Western Graduate&PostdoctoralStudies

Western University
Scholarship@Western

Electronic Thesis and Dissertation Repository

5-11-2012 12:00 AM

Software Size and Effort Estimation from Use Case Diagrams Using Regression and Soft Computing Models

Ali Bou Nassif
The University of Western Ontario

Supervisor
Dr. Luiz Fernando Capretz
The University of Western Ontario

Graduate Program in Electrical and Computer Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy
© Ali Bou Nassif 2012

Follow this and additional works at: https://ir.lib.uwo.ca/etd

 Part of the Other Electrical and Computer Engineering Commons

Recommended Citation

Bou Nassif, Ali, "Software Size and Effort Estimation from Use Case Diagrams Using Regression and Soft Computing Models" (2012). *Electronic Thesis and Dissertation Repository*. 547.
https://ir.lib.uwo.ca/etd/547

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlswadmin@uwo.ca.

SOFTWARE SIZE AND EFFORT ESTIMATION FROM USE CASE DIAGRAMS USING REGRESSION AND SOFT COMPUTING MODELS

(Spine title: Software Effort Estimation From Use Case Diagrams)

(Thesis Format: Integrated Article)

by

Ali Bou Nassif

Graduate Program in Electrical and Computer Engineering

A thesis submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

The School of Graduate and Postdoctoral Studies

Western University

London, Ontario, Canada

© Ali Bou Nassif 2012

WESTERN UNIVERSITY
SCHOOL OF GRADUATE AND POSTDOCTORAL STUDIES

CERTIFICATE OF EXAMINATION

Supervisor	Examiners
_____	_____
Dr. Luiz Fernando Capretz	Dr. Abdallah Shami
	_____
Co-Supervisor	Dr. Abdelkader Ouda
_____	_____
Mr. Danny Ho	Dr. Nazim Madhavji
Supervisory Committee	_____
	Dr. Khalil El-Khatib
_____	

The thesis by

Ali Bou Nassif

entitled:

Software Size and Effort Estimation from Use Case Diagrams Using Regression and Soft Computing Models

is accepted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Date: May 11, 2012

_____
Chair of the Thesis Examination Board

Abstract

In this research, we propose a novel model to predict software size and effort from use case diagrams. The main advantage of our model is that it can be used in the early stages of the software life cycle, and that can help project managers efficiently conduct cost estimation early, thus avoiding project overestimation and late delivery among other benefits. Software size, productivity, complexity and requirements stability are the inputs of the model. The model is composed of six independent sub-models which include non-linear regression, linear regression with a logarithmic transformation, Radial Basis Function Neural Network (RBFNN), Multilayer Perceptron Neural Network (MLP), General Regression Neural Network (GRNN) and a Treeboost model. Several experiments were conducted to train and test the model based on the size of the training and testing data points. The neural network models were evaluated against regression models as well as two other models that conduct software estimation from use case diagrams. Results show that our model outperforms other relevant models based on five evaluation criteria. While the performance of each of the six sub-models varies based on the size of the project dataset used for evaluation, it was concluded that the non-linear regression model outperforms the linear regression model. As well, the GRNN model exceeds other neural network models. Furthermore, experiments demonstrated that the Treeboost model can be efficiently used to predict software effort.

Keywords: Software Size and Effort Estimation, Use Case Diagrams, Regression Analysis, MLP Model, RBFNN Model, GRNN Model, Treeboost Model.

Acknowledgements

I am heartily thankful to my supervisors, Luiz Fernando Capretz and Danny Ho for their encouragement, guidance and support through my entire Ph.D. program. Their constructive feedback motivated me to conduct my research efficiently and challenged me to publish my work in reputable conferences and journals.

I would also like to thank my wife Adeeba for her patience, support and tireless effort during my studies.

Table of Contents

List of Tables

List of Figures

Glossary of Terms

ANOVA	Analysis of Variance: It provides statistical tests such as p-test and f-test to learn the significance of the independent variables
CI	Confidence Interval: It is a statistical term to measure the reliability of a result. In statistics, A 95% confidence level is used frequently. This means if an experiment is conducted over and over, 95% of the time the true parameter will fall in the interval
FP	Function Points. It is a unit of measurement to express the business functionalities of an information system. This method was introduced by Allan Albrecht at IBM in 1979
GRNN	General Regression Neural Network. It is a type of artificial neural network models that has four layers. The GRNN model was proposed by DF Specht in 1991
ISBSG	International Software Benchmarking Standards Group: It is a non-profit organization that maintains a repository of IT projects
MAE	Mean Absolute Error: It is an evaluation criterion which is the mean of the absolute error between the difference of the predicted effort and the actual effort

ME	Mean Error: It is an evaluation criterion which is the mean of the error between the difference of the predicted effort and the actual effort
MLP	Multilayer Perceptron: It is one of the traditional artificial neural network models. It is composed of an input layer, output layer and one or more hidden layers
MMER	Mean of the Magnitude of Error Relative to the estimate. It is an evaluation criterion which is the mean of the absolute value of the difference between the actual effort and the predicted effort divided by the predicted effort
MMRE	Mean of the Magnitude of Relative Error: It is an evaluation criterion which is the mean of the absolute value of the difference between the actual effort and the predicted effort divided by the actual effort
NFR	Non-Functional Requirements: These are also called quality attributes. In this thesis, NFR are used as independent variables such as productivity, complexity and requirements uncertainty
PRED	Prediction Level. It is an evaluation criterion which was used in conjunction with MMRE, MMER and CI. PRED(x) calculates the

	ratio of a project's MMRE (or MMER) that falls into the selected range (x) out of the total projects
RBF	Radial Basis Function: It is a real-valued function that satisfies the condition $\phi(x) = \phi(\|x\|)$
RBFNN	Radial Basis Function Neural Network. It is a type of artificial neural network models that has three layers. The RBFNN model was proposed by Broomhead and Lowe. The hidden layer contains a set of neurons that use Radial Basis Function (RBF) as activation functions
Spread	The spread is the radius or width of a RBF function which is denoted by "σ"
UCP	Use Case Points. It is a model introduced by G. Karner in 1993 to estimate software effort from use case diagrams

Chapter 1 [1]

1. Introduction

1.1 Motivation

Estimation is part of our daily lives. When we plan to go to work, we estimate the time needed to get there. This estimated time fluctuates according to some external factors, such as the weather conditions, traffic jams, and so forth. If we want to build a house, we estimate the cost and the schedule needed to complete its construction. Sometimes we conduct estimation intentionally, but often it occurs naturally. We instinctively enhance our estimation based on past experience and historical data.

Likewise, software estimation has become a crucial task in software engineering and project management. Old estimation methods that have been used to predict project costs

---

[1] Part of this chapter was published in the International Conference on Emerging Trends in Computer Science, Communications and Information Technology, and in the Journal of Global Research in Computer Science.

1. **Ali Bou Nassif**, Luiz Fernando Capretz and Danny Ho: Software Estimation in the Early Stages of the Software Life Cycle, *International Conference on Emerging Trends in Computer Science, Communications and Information Technology (CSCIT 2010),* January 2010, Nanded, India (Published)
2. **Ali Bou Nassif,** Luiz Fernando Capretz and Danny Ho: Enhancing Use Case Points Estimation Method using Soft Computing Techniques, *Journal of Global Research in Computer Science,* Volume 1, No. 4, November 2010, PP. 12-21 (Published).

developed using procedural languages are becoming inappropriate methods of estimation for the more recent projects being created with object-oriented languages. This in turn, may lead to project failures and has spawned the need for developing new approaches to software estimation.

The Standish Group [1] states that 44% of IT projects were delivered late and over budget. This indicates that the role of project management has become increasingly more important [2][3]. The International Society of Parametric Analysis (ISPA) identified the main reasons behind project failures [4]. These reasons can be summarized as follows:

- Lack of estimation of the staff's skill level
- Lack of understanding the requirements
- Improper software size estimation

Another study was conducted by the Standish Group International [1] to determine the main factors that lead to project failures. These factors include:

- Uncertainty of system and software requirements
- Unskilled estimators
- Budget limitation
- Optimism in software estimation
- Ignoring historical data
- Unrealistic estimation

In a nutshell, many software projects fail because of the inaccuracy of software estimation and misunderstanding or incompleteness of the requirements. This fact motivated researchers to conduct research on software estimation for better software size and effort assessment. One of the early stages of project management is planning; and in that stage, software developers begin to perform software size and effort estimation to calculate the budget, schedule and number of people required to develop the software.

According to Kotonya and Sommerville [3], the requirements engineering stage is mainly composed of four interleaved activities. These activities include Requirements Elicitation, Requirements Analysis and Negotiation, Requirements Documentation and Requirements Validation. Figure (1-1) shows the requirements engineering process [3]. As software estimation becomes critical to prevent or reduce project failures, estimation in the early stages of the software life cycle has become imperative. The earlier the estimation is, the better project management will be. The importance of early estimation is exposed when it is required to bid on a project or commit to a contract between a customer and a developer. The early software estimation is conducted at a point when the details of the problem are not yet disclosed; this is called the size estimation paradox [2] . The software size should first be estimated in the early stages. In general, the early stage of the software life cycle is the requirements phase.

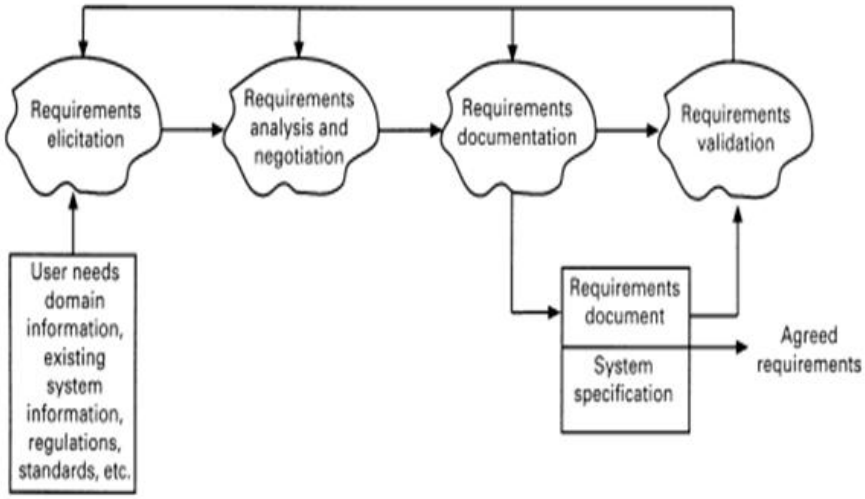


Figure 1-1 Requirements Engineering process [3]

Software estimation can be conducted at any activity within the requirements engineering process. However, performing estimation in the early activities stage, such as Requirements Elicitation means that the requirements of the software are not complete and more assumptions will need to be made in the estimation process. This could lead to poor results. On the other hand, if software estimation is done during or after the validation activity, fewer assumptions are needed and consequently, estimation results will be more accurate.

UML diagrams, proposed by Jacobson et al. in 1992 [5], such as use case diagrams, activity diagrams, collaboration diagrams, class diagrams and sequence diagrams are used in the requirements, analysis and design stages in the software life cycle. As UML diagrams have become popular in the last decade, software developers have become more interested in conducting software estimation based on UML models, and especially the

use case diagrams. The use case diagram as shown in Figure (1-2), is a set of use cases and actors that represents the functional requirements of a system and it is usually included in the Software Requirements Specification (SRS) documents.

This thesis focuses on developing a novel model to calculate software size and effort from use case diagrams. Our model can be used in the early stages of the software life cycle (requirements stage) and results show that the proposed model is a competitive one to alternative models that predict software effort from use cases.

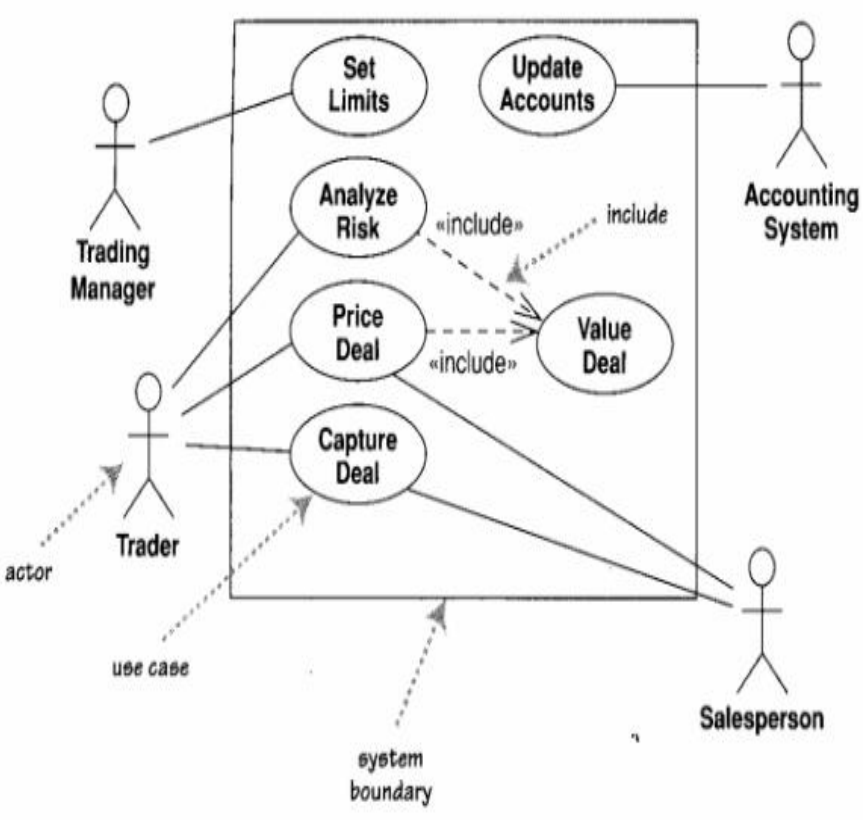


Figure 1-2 Use case diagram [6]

The model introduced in this thesis is geared toward estimating software effort of UML-based projects. For projects that do not contain use case diagrams and only contain textual representation of the functional requirements, we propose the following algorithm to map textual representation of the functional requirements to use case descriptions. After the mapping, our model can be used for software effort estimation. Please note that the validation of this algorithm is out of the scope of this thesis. The mapping algorithm is presented as follows:

1- Each main functional requirement is mapped to a use case
2- Each sub-requirement that deals with a condition or alternative flow is mapped to a transaction in the Extension (aka Alternative) scenario
3- Each sub-requirement that deals with a simple statement which represents an interaction between an actor and the system is mapped to a transaction in the Success (aka Main Flow) scenario
4- Each sub-requirement which is a mix between the above second and third steps is mapped to Success as well as Alternative transactions

Table 1-1 is an example for a textual functional requirement in a University Course Online Registration System project written using the RequisitePro tool. In this example, the main functional requirement is FEAT28 and there are five sub-requirements which include FEAT28.1, FEAT28.2, FEAT28.3, FEAT28.4 and FEAT28.5.

Table 1-1 Functional Requirement Example

Requirements	Priority	Difficulty	Stability	Risk	Origin
FEAT28: Students can enroll in any listed course	High	Medium	Medium	Schedule – Medium	End Users
FEAT28.1: After course registration deadline students can no longer enroll	High	Medium	Medium	Schedule – Medium	End Users
FEAT28.2: Cannot enroll in more than one course during a given time period	High	Medium	Low	Schedule – Low	End Users
FEAT28.3: The system should check that student has proper prerequisites	Medium	Medium	Medium	Schedule – Medium	End Users
FEAT28.4: Cannot enroll into a course that has reached max capacity	Low	Low	Medium	Technology – Low	End Users
FEAT28.5: Cannot enroll into more than five courses in the same term	Low	Medium	Low	Schedule – Low	End Users

With respect to the functional requirement listed in Table 1-1, the main requirement FEAT28 is mapped to a use case named "Enroll a Course". The sub-requirement FEAT28.1 describes three main transactions. The first transaction is that the student should select the course he or she wishes to enroll in. This is mapped to a transaction in the Success scenario. The second transaction is that student enrolls in the course which is also a transaction in the success scenario. The third transaction describes a condition that students should register before the deadline which should be listed under the Extensions (Alternative Scenario). The sub-requirement FEAT28.2 states a condition that students cannot enroll in two or more courses that run on the same time period. FEAT28.2 should be treated as a transaction under the Extensions. FEAT28.3 is mapped to a transaction in the Extensions scenario which checks if the prerequisites of the course are fulfilled. FEAT28.3 can also be mapped to a transaction under the Extensions (Alternative

Scenario) to describe the condition if the course prerequisites are not satisfied. FEAT28.4 describes a condition to check the maximum capacity of a course, which will be mapped to a transaction under the Extensions. Finally, FEAT28.5 also states a condition to check the number of courses registered by a student. Based on the above mapping description, the use case description (aka use case scenario) of the use case "Enroll a course" is presented in Table 1-2.

Table 1-2 Use case description

Use Case Title: Student Enrolls in a Course
Actors: Student, Admin
Precondition: The student is not enrolled in a course
Main Success Scenario (Main Flow):

1. The student chooses the course he or she wishes to enroll in
2. The student enrolls in the course

Extensions (Alternative)
2a: The student does not have permission (e.g. the student has not paid the tuition)
    2a1: Notify the student to contact the administrator
2b: The deadline has passed
    2b1: An error message will be displayed
2c: The prerequisite of the course is not fulfilled
    2c1: The student is advised to contact the professor to obtain permission
2d: Two courses have the same schedule
    2d1: The student is advised to choose one or the other
2e: The number of the enrolled courses has been exceeded
    2e1: An error message will be displayed
2f: The course is full
    2f1: An error message will be displayed
Post condition: The student has enrolled in a course

1.2 Research Questions

This research focuses on predicting software effort from use case diagrams. The use case point model [7] was the first model to deal with software effort prediction from use case diagrams. There are many limitations to the use case point model such as the complexity weights assigned to use cases and the description of these weights are not satisfactory, and the weights of the technical and environmental factors are outdated. There is several related work that addressed the issues of the use case model. Authors in [8] and [9] worked on adjustment factors, while others in [9] and [10] highlighted the discrepancies in designing use case models. Researchers in [11], [12] and [13] proposed different size metrics such as Transactions, TTPoints and Paths, while others [14], [15], [16], [17], [18], [19] and [20] went further to extend the UCP model by providing new complexity weights or by modifying the method used to predict effort.

Based on the above literature, we highlighted some research gaps. First, none of the above work used neural network models to predict software effort from use case diagrams. Second, the above work used linear regression for software effort estimation. Third, the size of the projects used in most datasets is small (less than 4,000 person-hours). As well, the influence of non-functional requirements was not addressed adequately. Thus, we ask seven relevant questions:

1. How can we measure the size of a use case and how can we estimate the size of a use case diagram?
2. How can team productivity contribute to software effort prediction?

3. To what degree can software effort estimation be influenced by project complexity?
4. How will unstable requirements affect the accuracy of software effort estimation?
5. To what degree can software effort prediction from use case diagrams be affected by non-functional requirements (productivity, complexity and requirements stability combined)?
6. What is the nature of the relationship between software effort and size?
7. What type of models can be used to predict software effort from use case diagrams?

Regarding the first question, we conducted two experiments. In the first experiment described in Chapter 3, we used the method proposed by the use case point (UCP) model (this model is described in Chapter 2). We found that this model is inadequate, specifically regarding large use cases. In the second experiment, which is presented in Chapter 4, we proposed a new method to calculate the size of a use case, and consequently the size of the use case diagram.

The second question is addressed in Chapters 3 and 4. In Chapter 3, we used the environmental factors with their default weights proposed by the UCP model to calculate productivity. However, these factors were filtered and new weights were proposed in Chapter 4. Moreover, we used a fuzzy logic technique to calibrate the proposed productivity values.

The third question is tackled in Chapter 4, as we proposed a new method to calculate the complexity of a project.

The fourth question is addressed in Chapter 3 and Chapter 4. In Chapter 3, we used requirements stability as one of eight factors that contribute to productivity. However, we found that the requirements stability factor plays an important role in estimating software effort. For this reason, we eliminated the requirements stability factor from the eight factors that contribute to productivity and proposed requirements stability as one of the independent factors that affect software estimation, which is also presented in Chapter 4.

The fifth question deals with the influence of non-functional requirements (NFR) on software estimation. Many published work ignore the impact of NFR on effort estimation. The UCP model [7] states that the NFR can increase the effort by about 30%. However, others argue that NFR can represent more than 50% of the total effort [21]. This indicates that NFR can double the predicted effort. In our research, we found that NFR can increase software effort by a factor of 2.6 (160%). In our model, we represent NFR through three main factors, which include productivity, complexity and requirements uncertainty. The productivity factor itself can increase the effort by 42% which corresponds to the lowest degree of team productivity. However, the complexity factor and requirements stability factors can increase software effort by 30% and 40%, respectively which correspond to the highest complexity degree and to the highest requirements uncertainty degree. As a combination of productivity, complexity and

requirements uncertainty factors (this combination represents the NFR), the effort can be increased to a factor of 2.6 (1.42*1.3*1.4) or by 160%.

In research question six, we ask about the relationship between software size and effort. All researchers agree that software effort is correlated to software size. This means, when software size increases, software effort will increase. However, many models including the UCP claim that the relationship between software effort and size is linear. Other prominent cost estimation models such as COCOMO claims that this relationship is log-linear and it is represented as $Effort = a * Size^b$. In Chapters 3 and 4, we argue that this relationship is non-linear. Specifically, we introduced three types of non-linear models in Chapter 4 and we showed by experiments that these models outperform the log-linear model especially for large projects. This is a breakthrough in the field of software estimation.

In question seven, we investigate different models to see which one is suitable for software effort prediction from use case diagrams. We show in Chapters 3 and 4 that linear and non-linear regression models can be used for software effort estimation. Furthermore, we assert that neural network models and especially MLP, RBFNN and GRNN can also be used as alternatives to regression models. In Chapter 5, we present a Treeboost model to predict software effort from use case diagrams based on three predictors which include software size, productivity and complexity.

1.3 Research Contributions

This thesis focuses on creating a model to predict software size and effort from use case diagrams. Research contribution can be mainly summarized as follows:

1- Several experiments were conducted to figure out the nature of the relationship between software effort and size. Results concluded that this relationship is non-linear, although the degree of non-linearity varies based on how large the software size is. For instance, this non-linear relationship is insignificant with small projects. However, this non-linearity becomes evident with mid-sized projects and stands out with large projects.

2- Six different levels of complexity for use cases were identified. These include Very Low, Low, Normal, High, Very High and Extra High. This classification is based on the number of transactions of each use case by giving the Success scenario more weight than the Extension scenario.

3- A new method to calculate the productivity of the team developing a project was proposed. The overall productivity factor is based on five factors; each has five levels (Level-1 which corresponds to very low, to Level-5 which corresponds to very high). These factors include team experience about the problem domain, team motivation, experience in the programming language used, experience in the object oriented language and the level of the analytical skills of the team. Additionally, we propose a weight to each of these five factors that contribute to productivity. The final productivity weight is calculated based on the level of each

of the five factors. Furthermore, we used a fuzzy logic technique to calibrate the proposed productivity factor.

4- A new method to calculate the project complexity factor was put forward based on five levels. A weight was assigned to each complexity level.

5- Five levels of requirements uncertainty were proposed. Requirements uncertainty includes the increase in the number of requirements as well as the change of the requirements during the software development life cycle.

6- Six different models were put forward to estimate software effort from software size, productivity, complexity and requirements uncertainty. These models include linear regression, non-linear regression, Multilayer Perceptron neural network, Radial Basis Function Neural Network, General Regression Neural Network and Treeboost. Four experiments were carried out to evaluate and test the proposed models with two other models that conduct software estimation from use case diagrams. In the first experiment, all models were tested using 65 data points of effort ranging between 120 person-hours and 224,890 person-hours. After that, the 65 testing data points were divided into three categories: Small Dataset, which contains 25 projects of effort ranging between 120 person-hours and 3,000 person-hours; Medium Dataset which contains 21 projects of effort ranging between 3,000 person-hours and 10,000 person-hours; and Large Dataset which contains 19 projects of effort greater than 10,000 person-hours. In the second experiment, all models were tested using the Small Dataset; however, in the third and the fourth experiments, all models were tested using the Medium and

the Large Datasets respectively. A thorough comparison among all models was carried out based on each experiment and recommendations on how to use each model were proposed. Additionally, the proposed model was evaluated against models that conduct software estimation from use case diagrams. The experiments show that the proposed model outperforms other models based on different evaluation criteria.

1.4 Thesis Structure

This thesis is organized as follows. Chapter 2 defines the terms used in this work, and then presents a literature review, followed by related work. Chapter 3 introduces the linear regression model and the Multilayer Perceptron neural network model. In Chapter 4, we elaborate on the linear and non-linear regression models, as well as the Radial Basis Function Neural Network model and the General Regression Neural Network Model. Chapter 5 proposes a Treeboost model to estimate software effort based on three predictors. Finally, Chapter 6 summarizes the thesis and proposes future work.

References

[1] J. Lynch. Chaos manifesto. The Standish Group. Boston. 2009[Online]. Available: http://www.standishgroup.com/newsroom/chaos_2009.php.

[2] O. Demirors and C. Gencel, "A Comparison of Size Estimation Techniques Applied Early in the Life Cycle," *Software Process Improvement,* vol. 3281, pp. 184-194, 2004.

[3] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques.* Chichester; New York: John Wiley, 1998.

[4] D. Eck, B. Brundick, T. Fettig, J. Dechoretz and J. Ugljesa, "Parametric estimating handbook," The International Society of Parametric Analysis, Fourth Edition. 2009.

[5] I. Jacobson, M. Christerson, P. Jonsson and G. Overgaard, *Object-Oriented Software Engineering: A use Case Driven Approach.* Addison Wesley, 1992.

[6] J. Rumbaugh, I. Jacobson and G. Booch, "Use cases," in *UML Distilled*, 3rd ed., M. Fowler, Ed. Pearson Higher Education, 2004, pp. 103.

[7] G. Karner, "Resource Estimation for Objectory Projects," *Objective Systems,* 1993.

[8] S. Diev, "Use cases modeling and software estimation: applying use case points," *SIGSOFT Softw. Eng. Notes,* vol. 31, pp. 1-4, 2006.

[9] B. Anda, H. Dreiem, D. I. K. Sjoberg and M. Jorgensen, "Estimating software development effort based on use cases-experiences from industry," *4th International Conference on the Unified Modeling Language, Modeling Languages, Concepts, and Tools,* 2001, pp. 487-502.

[10] M. Arnold and P. Pedross, "Software size measurement and productivity rating in a large-scale software development department," in *Proceedings of the 20th International Conference on Software Engineering,* 1998, pp. 490-493.

[11] G. Robiolo and R. Orosco, "Employing use cases to early estimate effort with simpler metrics," *Innovations in Systems and Software Engineering,* vol. 4, pp. 31-43, 2008.

[12] G. Robiolo, C. Badano and R. Orosco, "Transactions and paths: Two use case based metrics which improve the early effort estimation," in *International Symposium on Empirical Software Engineering and Measurement,* 2009, pp. 422-425.

[13] M. Ochodek and J. Nawrocki, "Automatic transactions identification in use cases," in *Balancing Agility and Formalism in Software Engineering*, B. Meyer, J. R. Nawrocki and B. Walter, Eds. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 55-68.

[14] K. Periyasamy and A. Ghode, "Cost estimation using extended use case point model," in *International Conference on Computational Intelligence and Software Engineering,* 2009.

[15] F. Wang, X. Yang, X. Zhu and L. Chen, "Extended use case points method for software cost estimation," in *International Conference on Computational Intelligence and Software Engineering,* 2009.

[16] G. Schneider and J. P. Winters, *Applied use Cases, Second Edition, A Practical Guide.* Addison-Wesley, 2001.

[17] M. R. Braz and S. R. Vergilio, "Software effort estimation based on use cases," in *COMPSAC '06,* 2006, pp. 221-228.

[18] A. B. Nassif, L. F. Capretz and D. Ho, "Estimating software effort based on use case point model using sugeno fuzzy inference system," in *23rd IEEE International Conference on Tools with Artificial Intelligence,* Florida, USA, 2011, pp. 393-398.

[19] P. Mohagheghi, B. Anda and R. Conradi, "Effort estimation of use cases for incremental large-scale software development," in *Proceedings of the 27th International Conference on Software Engineering,* St. Louis, MO, USA, 2005, pp. 303-311.

[20] M. Ochodek, J. Nawrocki and K. Kwarciak, "Simplifying effort estimation based on Use Case Points," *Information and Software Technology,* vol. 53, pp. 200-213, 2011.

[21] Y. Ossia. IBM haifa research lab. *IBM Haifa Research Lab* [Online]. 2011. Available: https://www.research.ibm.com/haifa/projects/software/nfr/index.html.

Chapter 2

2. Background

In this chapter, we define the terms used in this thesis such as fuzzy logic, neural network and its types, as well as the criteria used to evaluate our work. Moreover, a literature review and the related work are presented.

2.1 Fuzzy Logic

Fuzzy logic is derived from the fuzzy set theory that was proposed by Lotfi Zadeh in 1965 [1]. As a contrary to the conventional binary (bivalent) logic that can only handle two values *True* or *False* (1 or 0), fuzzy logic can have a truth value which is ranged between 0 and 1. This means that in the binary logic, a member is completely belonged or not belonged to a certain set, however in the fuzzy logic, a member can partially belong to a certain set. Mathematically, a fuzzy set *A* is represented by a membership function as follows:

$$F_z[x \in A] = \mu_A(x) : \mathbb{R} \rightarrow [0,1]. \qquad (2.1)$$

Where μ_A is the degree of the membership of element *x* in the fuzzy set *A*.

A fuzzy set is represented by a *membership function*. Each element will have a grade of membership that represents the degree to which a specific element belongs to the set. Membership functions include *Triangular, Trapezoidal* and *S-Shaped*. In fuzzy logic,

linguistic variables are used to express a rule or fact. For example, “the temperature is thirty degrees” is expressed in fuzzy logic by “the temperature is low” or “the temperature is high” where the words *low* and *high* are linguistic variables. In fuzzy logic, the knowledge based is represented by if-then rules. For example, if the temperature is high, then turn on the fan. The fuzzy system is mainly composed of three parts. These include *Fuzzification*, *Fuzzy Rule Application* and *Defuzzification*. Fuzzification means applying fuzzy membership functions to inputs. Fuzzy Rule Application is to make inferences and associations among members in different groups. The third step in the fuzzy system is to defuzzify the inferences and associations, make a decision and provide an output that can be understood. In this thesis work, fuzzy logic is used to calibrate the productivity factor of the regression model.

2.2 Neural Network

Artificial Neural Network (ANN) is a network composed of artificial neurons or nodes which emulate the biological neurons [2]. ANN can be trained to be used to approximate a non-linear function, to map an input to an output or to classify outputs. There are several algorithms available to train a neural network but this depends on the type and topology of the neural network. The most prominent topology of ANN is the feed-forward networks. In a feed-forward network, the information always flows in one direction (from input to output) and never goes backwards. An ANN is composed of nodes organized into layers and connected through weight elements. At each node, the

weighted inputs are aggregated, thresholded and inputted to an activation function to generate an output of that node. Mathematically, this can be represented by:

$$y(t) = f[\sum_{i=1}^{n} w_i x_i - w_0]. \quad (2.2)$$

Where x_i are neuron inputs, w_i are the weights and f[.] is the activation function. There are many types of activation functions as shown in Figure (2-1) [3].

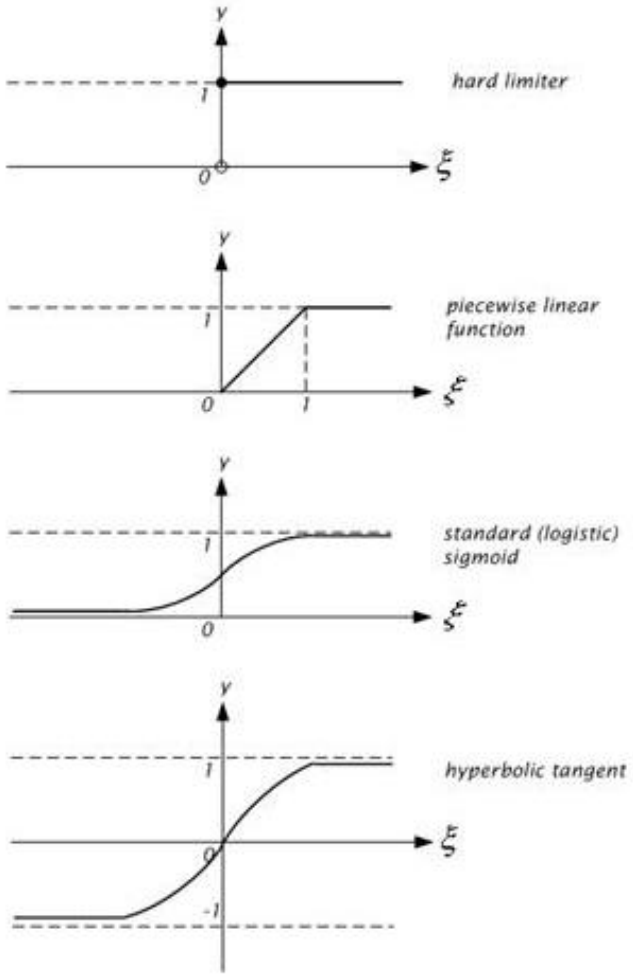


Figure 2-1 Activation functions [3]

Feed-Forward ANN layers are usually represented as *input*, *hidden* and *output* layers. If the hidden layer does not exist, then this type of the ANN is called *perceptron*. The perceptron is a linear classifier that maps an input to an output provided that the output falls under two categories. The perceptron can map an input to an output if the relationship between the input and output is linear. If the relationship between the input and output is not linear, one or more hidden layers will exist between the input and output layers to accommodate the non-linear properties. Several types of feed-forward neural networks with hidden layers exist. These include Multilayer Perceptron (MLP), Radial Basis Function Neural Network (RBFNN) and General Regression Neural Network (GRNN).

2.2.1 Multilayer Perceptron (MLP)

A MLP is a feed-forward neural network model that contains at least one hidden layer and each input vector is represented by a neuron. The main difference between the MLP and the Perceptron is that in the Perceptron, there are no hidden layers. In general, the neurons in the hidden layer use non-linear activation function such as the sigmoid function (logistic). The output layer node usually uses a linear activation function. The number of hidden neurons varies and can be determined by trial and error so that the error is minimal. MLPs are usually trained using the backpropagation algorithm which is a type of gradient decent algorithm. Another algorithm can be used to train a MLP which is the conjugate gradient algorithm [4]. The applications of the MLP model include image recognition, speech recognition, curve fitting and machine translation. Figure (2-2) shows

the schematic diagram of a MLP that has five input vectors, seven neurons and one output.

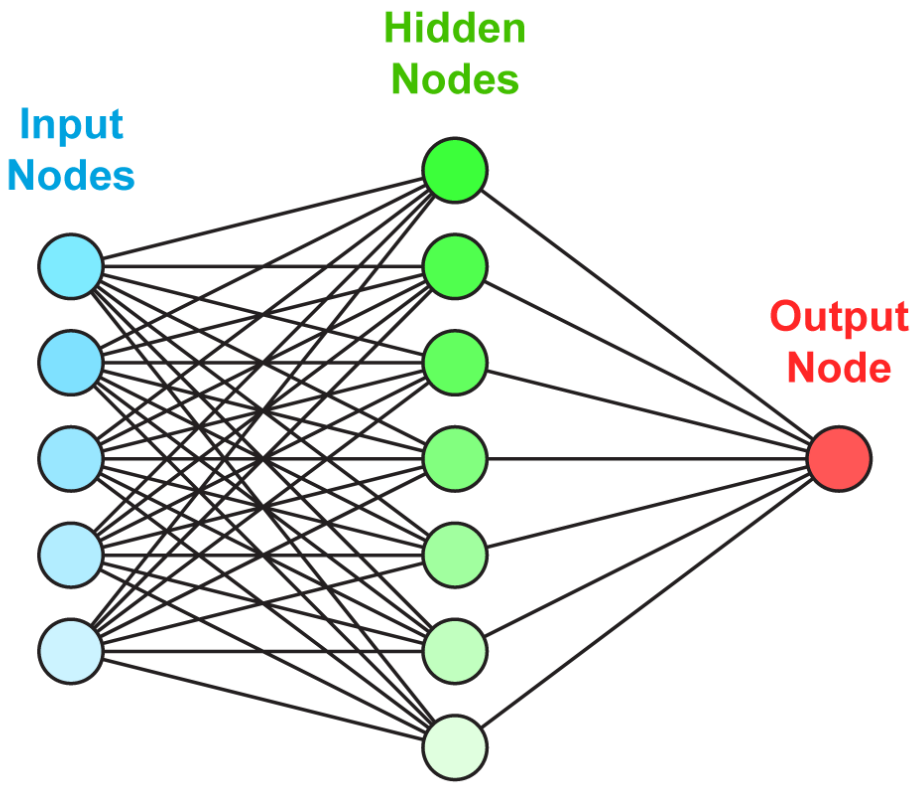


Figure 2-2 Schematic diagram of a MLP model

2.2.2 Radial Basis Function Neural Network

A Radial Basis Function Neural Network (RBFNN) was introduced by Broomhead and Lowe [5]. A RBFNN is a feed-forward network that has three layers; an input layer, a hidden layer and an output layer. Figure (2-3) shows the architecture of the RBFNN.

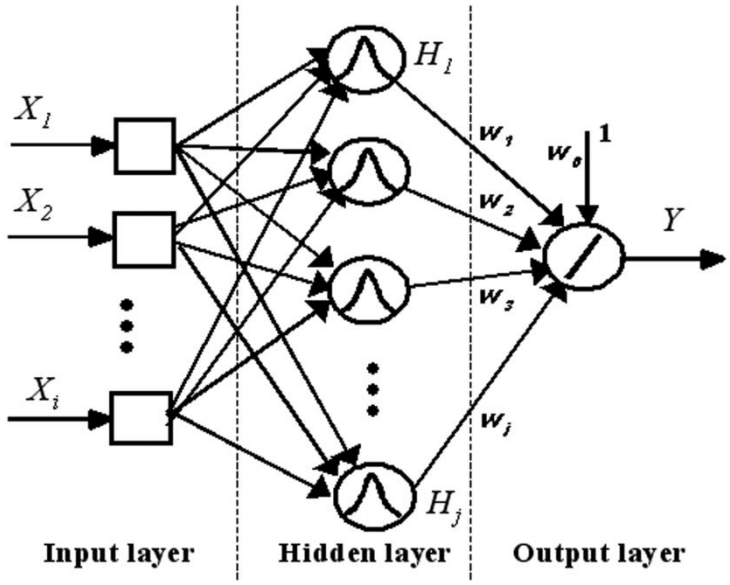


Figure 2-3 Schematic diagram of a RBFNN model [5]

The first layer is the input layer that represents the input vectors (in this chapter, there are four input vectors; software size, team productivity, project complexity and requirements stability). The hidden layer contains a set of neurons that use Radial Basis Function (RBF) as activation functions. An RBF function depends on the distance from its center C_i to the input X. Each RBF function has a radius or width (also called spread) which is denoted by "σ". The width might be different for each neuron. The Gaussian function is the most commonly used in RBF as shown in Equation (2.3):

$$f(x) = \exp(-\frac{\| X - C_i \|^2}{2\sigma_i^2}). \qquad (2.3)$$

Where Ci is the center and σ_i is the width of the i^{th} neuron in the hidden layer. The distance between X and the center is usually an Euclidean distance. The main advantages

of the RBFNN over other feed-forward neural networks include fast learning and not suffering from problems such as local minima and paralysis [6].

2.2.3 General Regression Neural Network

The General Regression Neural Network (GRNN) is a type of neural networks that performs regression on continuous target (output) variables. The GRNN was proposed by Specht in 1991 [7]. A GRNN is composed of four layers as depicted in Figure (2-4).

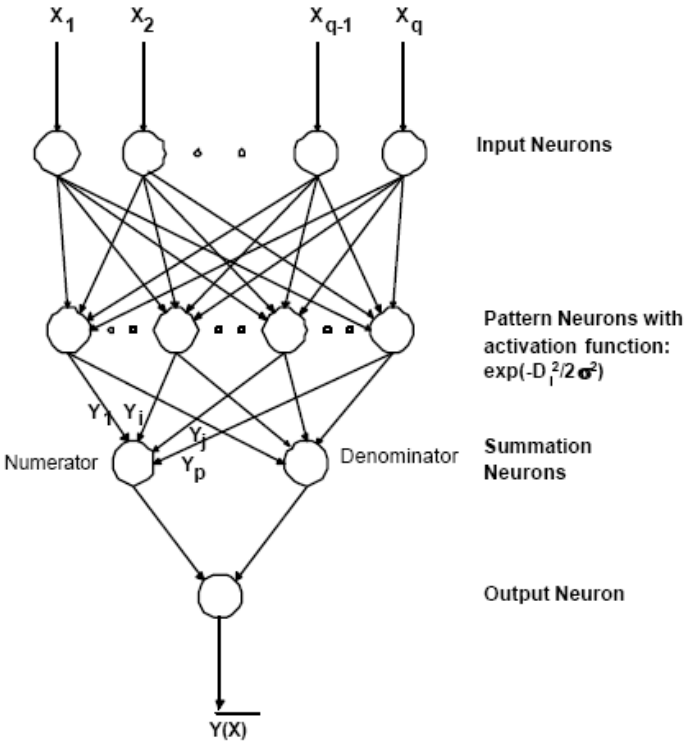


Figure 2-4 Schematic diagram of a GRNN model [7]

The first layer is the input layer in which each independent variable (predictor) has a neuron. The input neurons feed the values to the neuron in the second layer.

The second layer contains pattern neurons such that each training row in the training dataset has a neuron. Each neuron computes the Euclidean distance from the input vector X to the neuron's center, then applies the RBF function using the sigma "σ" values. The resulting value is then passed to neurons in the third layer (summation neurons).

The third layer only contains two neurons. One neuron is called the denominator summation which adds the values of the weights coming from each of the pattern neurons (second layer). The other neuron is the numerator summation that adds the weights multiplied by the actual output (target) value of each pattern neurons.

The fourth layer contains the output neuron in which the value stored in the numerator neuron is divided by the value stored in the denominator neuron. The output is the predicted target value.

The GRNN has several advantages such as they learn faster and are more accurate than other neural network models. Moreover, GRNN models are fairly insensitive to outliers. The main disadvantage of GRNN is that it requires more memory space to store the model and it becomes inapplicable if the number of the training project datasets is very huge.

2.3 Evaluation Criteria

Several methods exist to compare cost estimation models. Each method has its advantages and disadvantages. In our work, five different evaluation methods have been used. These methods include the Mean of the Magnitude of Relative Error (MMRE), the

Mean of Magnitude of error Relative to the Estimate (MMER) the Prediction Level (PRED), the Mean Error at 95% Confidence Interval (CI) and the Mean Absolute Error (MAE).

- MMRE*:* This is a very common criterion used to evaluate software cost estimation models [8]. The Magnitude of Relative Error (MRE) for each observation *i* can be obtained as:

$$MRE_i = \frac{|Actual\ Effort_i - Predicted\ Effort_i|}{Actual\ Effort_i}. \quad (2.4)$$

MMRE can be achieved through the summation of MRE over *N* observations:

$$MMRE = \frac{1}{N}\sum_{1}^{N} MRE_i. \quad (2.5)$$

- MMER: Another method can be used as an alternative to the MMRE which is the Magnitude of Error Relative to the estimate (MER) [9]. MER is similar to MRE with a difference that the denominator is the predicted effort instead of the actual effort. Consequently, the equations for MER and MMER are:

$$MER_i = \frac{|Actual\ Effort_i - Predicted\ Effort_i|}{Predicted\ Effort_i}. \quad (2.6)$$

$$MMER = \frac{1}{N}\sum_{1}^{N} MER_i. \quad (2.7)$$

As seen from Equations (2.4) and (2.6), improving one method might adversely affect the other method. This is because the denominator of the MRE is the actual effort where the

denominator of MER is the predicted effort. Nevertheless, it is important that MMRE and MMER are both used for evaluation. For instance, if the MMRE is large and the MMER is small, this indicates that the average actual effort of the projects is less than the average estimated effort. On the contrary, large MMER values indicate that the average estimated effort is less than the average actual effort.

- PRED(x): PRED *(x)* can be described as:

$$PRED(x) = \frac{k}{n}. \tag{2.8}$$

where x is the maximum MMRE (or MMER) of a selected range, n is the total number of projects, and k is the number projects in a set of n projects whose MMRE (or MMER) <= x. PRED calculates the ratio of a project's MMRE (or MMER) that falls into the selected range (x) out of the total projects. For example, PRED (30) gives the percentage of software projects that were estimated with MMRE (or MMER) less than or equal to 0.3. The estimation accuracy is proportional to PRED (x) and inversely proportional to MMRE or MMER.

- CI: The equation of the mean error confidence interval is:

$$CI = \bar{x} \pm t * \frac{SD}{\sqrt{N}}. \tag{2.9}$$

Where $\bar{x}$ is the mean error, *SD* is the standard deviation, *N* is the number of projects and *t* is a constant called the test statistic that depends on the number of the samples (projects) and the degree of the confidence level. The value of *t* is obtained from the *t-distribution*

table. The 95% confidence level becomes standard to many disciplines. For example, the value of *t* is 2.042, 2, 1.98 and 1.96 if the number of projects is 30, 60, 100 and 1,000 respectively at the 95% confidence level. For instance, the value (SD/√N) is called the standard error of the mean.

The equation for the mean error for each observation i and total number of observations N is:

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i . \tag{2.10}$$

Where $x_i = (Actual\ Effort_i - Predicted\ Effort_i)$

The equation of the standard deviation can be seen as:

$$SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2} \ . \tag{2.11}$$

- MAE: The Mean Absolute Error (MAE) is the average of the absolute errors between the actual and the predicted effort as shown in Equation (2.12).

$$MAE = \frac{1}{N}\sum_{i=1}^{N} | E_a - E_p | . \tag{2.12}$$

  Where E_a is the actual effort and E_p is the predicted effort.

2.4 Literature Review

Software estimation can be affected by several parameters [10] . These parameters include:

- Size: The effort and cost of a software project depends on the size of the project. The larger the size is, the higher effort and cost will be needed. Software size estimation will first be performed if the size of the project is unknown upon conducting effort estimation. The size of a project can be measured in Source Lines of Codes (SLOC) or Function Points (FP).
- Category: The category of a project is important in software estimation. Examples of project categories include Development, Maintenance, Migration, etc.
- Personnel Attributes: The experience and the productivity of a team affect software estimation.
- Domain: The domain of the project might affect software estimation. The effort to build a human resources management system is different from the effort needed to develop an accounting and stock management system. Examples of domain categories include finance, insurance, retail and manufacturing.
- Complexity: the complexity of a project plays an important role in software estimation. Examples of complexity include mission-critical (will the application be used in a healthcare system to monitor the heartbeats and the blood pressure of a person?), architecture (is the architecture 2 tiers, 3 tiers or multi-tiers?) and Service Level Agreement (will there be a strict SLA that should be met?).

There are several models for software effort and cost estimation. These include algorithmic models, expert judgement models, estimation by analogy models and soft computing models.

2.4.1 Algorithmic Models

This is still the most popular category in the literature [11]. These models include COCOMO [12], SLIM [13], Function Point, Use Case Points [20] and SEER-SEM [14]. The main cost driver of these models is the software size. In COCOMO and SLIM models, the size is measured in Source Lines of Code (SLOC). However, the function point and the use case point models take software size in function points (FP) and use case points (UCP) respectively. Algorithmic models either use a linear regression equation, the one used by Kok et al. [15] or non-linear regression equations, those which are used by Boehm [12].

2.4.1.1 COCOMO

The COnstructive COst MOdel (COCOMO) is an algorithmic model used to predict software cost. It was developed by Barry Boehm in 1981 [12], and it was known as COCOMO 81. COCOMO uses a simple regression formula. The model's parameters are derived from historical projects and current project characteristics. There are three main types of COCOMO 81. These include Basic COCOMO, Intermediate COCOMO and Detailed COCOMO. The Basic COCOMO equations are as follow:

$$Effort = a \times Size^{b}. \tag{2.13}$$

Where Effort is measured in person-months and Size is measured in KSLOC. The constants "a" and "b" are determined based on the project type as seen in Table (2.1). Equation (2.14) is used to calculate the time required to develop the project.

$$Development\_Time = c \times Effort^{d}. \qquad (2.14)$$

Where Development_Time is measured in months. The constants "c" and "d" are also shown in Table (2.1). Equation (2.15) shows the number of people required for the project development.

$$People\_\mathrm{Re}quired = \frac{Effort}{Development\_Time}. \qquad (2.15)$$

The constants "a", "b", "c" and "d" are determined based on three categories of projects which are Organic, Semi-detached and Embedded as shown in Table (2-1). Organic projects are projects where small teams with good experience are working with non-strict requirements. Projects are classified as Semi-detached when medium teams with mixed experience are working with requirements which are mixed between strict and non-strict. Embedded projects are those that have tight constraints.

Intermediate COCOMO is an advanced model of the Basic COCOMO where software effort is a function of software size and 15 other cost-driver attributes. These attributes represent the non-functional requirements of the project. Each attribute has a rate on a six-point scale ranging from "very low" to "extra high".

Detailed COCOMO incorporates the characteristics of the Intermediate COCOMO with an assessment of the cost drivers according to each phase of the software life cycle.

Table 2-1 Software project types [12]

Software Project	a	b	c	D
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Boehm introduced COCOMO II model [16] which is an advanced model of COCOMO 81. COCOMO II is more suitable for estimating modern software development projects. The main differences between COCOMO II and COCOMO'81 can be summarized as:

- COCOMO II takes into account requirements volatility.
- Estimation is adjusted for software reuse and re-engineering when automated tools are used.
- Cost drivers were updated.
- COCOMO II has more data points (161 data points as opposed to 63 in COCOMO'81.
- COCOMO II uses logical SLOC where COCOMO'81 uses physical SLOC. One logical SLOC (if-then-else) might contain several physical SLOC.

2.4.1.2 SLIM

The Software LIfecycle Management (SLIM) model, which is also known as the Putnam model was developed by Lawrence Putnam in 1978 [13]. The SLIM describes the effort and time required to finish a project of a certain size. The time-effort curve of Putnam

follows the Rayleigh distribution as shown in Figure (2-5). The effort required to develop a project is as follows:

$$Effort = \left[\frac{Size}{\Pr oductivity \times Time^{4/3}} \right]^3 \times B. \qquad (2.16)$$

Where Effort is measured in person-years and Size in SLOC. Productivity is the process productivity which is the ability of a software organization to develop software of a given size at a certain defect rate. Time is measured in years where B is a scaling factor and it is a function of project size.

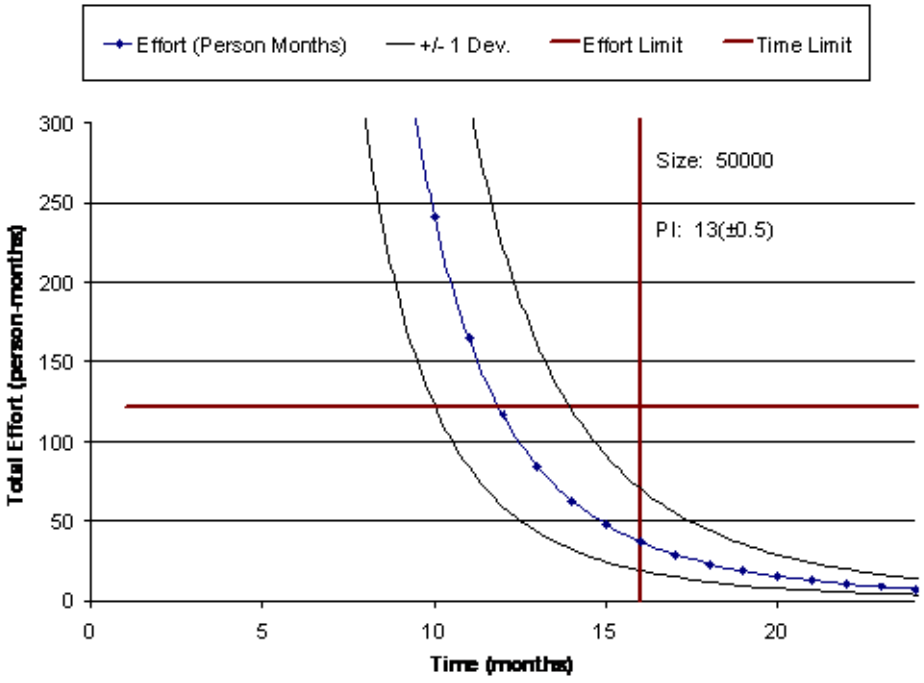


Figure 2-5 Putnam's time-effort graph based on Rayleigh distribution [13]

2.4.1.3 Function Point Model

Function Points measure the functionality of software as opposed to SLOC which measures the physical components of software. The function point method was proposed by Allan Albrecht in 1979 [17] [18]. There are a few methods to count function points but the standard method is the one that is maintained by the Function Points Analysis (FPA) which is based on the International Function Point Users Group (IFPUG) [19].

FPA defines five parameters that the size of software depends on. These parameters include inputs, outputs, inquiries, internal files and external interfaces. It is clear that these parameters are touchable by the end user. Figure (2-6) shows the function points parameters within an application [18]. These parameters are discussed as the following:

- Inputs: These are inputs from the user to the application. For example, create, delete, update and read are considered as inputs.
- Outputs: This is an output of a certain process in the application. For example a financial report in an organization. The financial report is considered as an output if it is printed, or stored in a database or external media storage, or even if it is just displayed on the screen.
- Inquiries: These are queries executed by the user to fetch some data stored in the database. The output of an inquiry is similar to the output discussed above, except that business information is not processed in this case. Information is sorted or rearranged based on the query issued by the user.

- Internal files: These files store all the data of the application. Internal files belong to the application and are maintained by the application owner or the administrator.
- Interfaces: This is the interface of external applications by which transactions can be made to the main application. The function point model defines interface as files that belong to external applications and are supported by those applications, however these files contribute to the size of the main application. For example, the main application might request a file that contains important information and this file is maintained and updated by other applications.

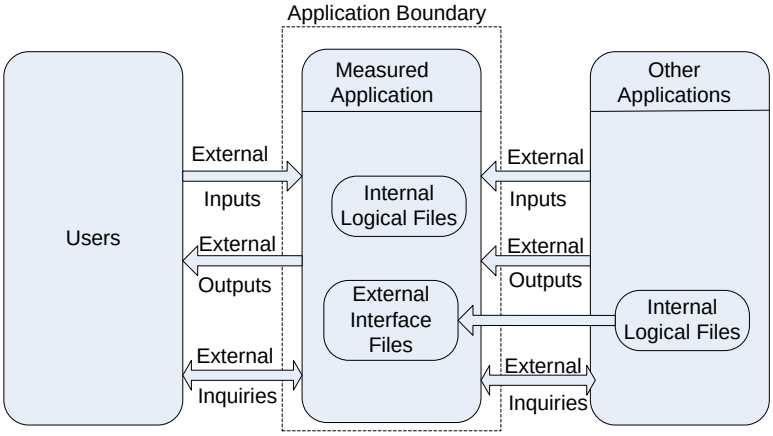


Figure 2-6 High level view of the function point model

2.4.1.4 Use Case Point Model

The Use Case Point (UCP) model [20] is based on mapping a use case diagram to a size metric called use-case points. A use case diagram shows how users interact with the system. A use case diagram is composed of use cases and actors. Use cases represent the

functional requirements where an actor is a role played by a user. Figure (1-2) is an example of a use case diagram. Each use case is represented by a use case scenario (description). The use case scenario (description) is mainly composed of a Success scenario and an Extension (Alternative) scenario as shown in Table (2-2).

The use case point model was first described by Gustav Karner in 1993 [20]. This model is used for software cost estimation based on the use case diagrams. First, the software size is calculated according to the number of actors and use cases in a use case diagram multiplied by their complexity weights. The complexity weights of use cases and actors are presented in tables (2-3) and (2-4) respectively.

As shown in Table (2-3), the complexity of a use case is determined by the number of its transactions as shown in the use case description of each use case. The software size is calculated through two stages. These include the Unadjusted Use Case Points (UUCP) and the Adjusted Use Case Points (UCP). UUCP is achieved through the summation of the Unadjusted Use Case Weight (UUCW) and Unadjusted Actor Weight (UAW). UUCW is represented in Equation (2.17).

Table 2-2 Use case scenario (description)

Use Case Title: Student Enrolls in a Course
Actors: Student, Admin
Precondition: The student is not enrolled in a course
Main Success Scenario (Main Flow):

1. The student chooses the course he or she wishes to enroll in
2. The student enrolls in the course

Extensions (Alternative)
2a: The student does not have permission (e.g. the student has not paid the tuition)
2a1: Notify the student to contact the administrator
2b: The deadline has passed
2b1: An Error message will be displayed
2c: The prerequisite of the course is not fulfilled
2c1: The student is advised to contact the professor to obtain permission
2d: Two courses have the same schedule
2d1: The student is advised to choose either one
2e: The number of the enrolled courses has been exceeded
2e1: An error message will be displayed
2f: The course is full
2f1: An error message will be displayed
Post condition: The student has enrolled in a course

Table 2-3 Complexity weights of use cases [20]

Use Case Complexity	**Number of Transactions**	**Weight**
Simple	Less than 4 (should be realized by less than 5 classes)	5
Average	Between 4 and 7 (should be realized between 5 and 10 classes)	10
Complex	More than 7 (should be realized by more than 10 classes)	15

Table 2-4 Complexity weights of actors [20]

Actor Complexity	**Description**	**Weight**
Simple	Through an API	1
Average	Through a text-based user interface	2
Complex	Through a Graphical User Interface	3

$$UUCW = \sum_{i=1}^{3} n_i \times W_i. \quad (2.17)$$

where n_i is the number of items of variety i of the use cases and $W\underline{i}$ is the complexity weight of the corresponding use case. Similarly, UAW is represented as follows:

$$UAW = \sum_{j=1}^{3} m_j \times C_j. \quad (2.18)$$

where m_j is the number of items of variety j of the actors and C_j is the complexity weight of the corresponding actor. Consequently, UUCP can be defined as follows:

$$UUCP = UUCW + UAW \quad (2.19)$$

After calculating the UUCP, the Adjusted Use Case Points (UCP) is calculated. UCP is achieved by multiplying UUCP by the Technical Factors (TF) and the Environmental Factors (EF). TF and EF represent the non-functional requirements of the software. TF contributes to the complexity of the project while EF contributes to the team efficiency and productivity. The technical and environmental factors are depicted in tables (2-5) and (2-6) respectively. The technical factor is detailed as follows:

$$TF = 0.6 + 0.01 \sum_{i=1}^{13} T_i \times W_i. \quad (2.20)$$

where T_i is a factor that takes values between 0 and 5. The value "0" indicates that the factor is unrelated while the value "5" indicates that the factor is indispensable. The value "3" specifies that the technical factor is not very important, nor irrelevant (average). For instance, if all of the factors have a value of "3", the technical factor (TF) will be 1. W_i represents the weight of technical factors (Table 2-5).

On the other hand, the environmental factor (EF) can be described as follows:

$$EF = 1.4 - 0.03\sum_{i=1}^{8} E_i \times W_i. \tag{2.21}$$

where E_i is the Environmental Factor (which is similar to T_i in Equation 2.20), taking values between 0 and 5. Finally, the Adjusted Use Case Points (UCP) can be defined as follows:

$$UCP = UUCP \times TF \times EF. \tag{2.22}$$

By incorporating TF and EF, the value of UCP will be more or less than the value of UUCP by 30%. For effort estimation, Karner proposed 20 person-hours to develop each UCP. This is expressed in Equation (2.23):

$$Effort = Size \times 20. \tag{2.23}$$

where Effort is measured in person-hours and Size is measured in UCP.

Table 2-5 Technical factors

T_i	**Complexity Factors**	W_i
T_1	Easy installation	0.5
T_2	Portability	2
T_3	End user efficiency	1
T_4	Reusability	1
T_5	Complex internal processing	1
T_6	Special security features	1
T_7	Usability	0.5
T_8	Application performance	1
T_9	Special user training facilities	1
T_{10}	Concurrency	1
T_{11}	Distributed systems	2
T_{12}	Provide direct access for third	1
T_{13}	Changeability	1

Table 2-6 Environmental factors

E_i	**Efficiency and Productivity Factors**	W_i
E_1	Familiar with Objectory	1.5
E_2	Object oriented experience	1
E_3	Analyst capability	0.5
E_4	Stable requirements	2
E_5	Application experience	0.5
E_6	Motivation	1
E_7	Part-time workers	-1
E_8	Difficult programming language	-1

There are several limitations regarding the UCP model. These include:

- The complexity of a use case is based on the number of transactions in the use case scenario. A complex use case is defined when the number of transactions is

more than seven. In the industry, some use cases might contain more than twenty transactions. According to the UCP, a use case with eight transactions has the same complexity rate as the one of twenty transactions. However, the effort required to build a use case of twenty transactions is more.

- The UCP assumes that the effort required to develop the Main Success Scenario of a use case is the same as the Extensions, if both the Success scenario and the Extensions have the same number of transactions. In fact, the effort required to develop the Main Success Scenario should be more because it is the core of the use case scenario.
- In the UCP model, NFR can increase software effort by 30%. However, According to IBM, the NFR might increase the software effort of a software project by 100% [21].
- The UCP ignores the *Include* and *Extend* use cases in the use case diagram. However, developing these types of use cases require effort and thus, they should not be ignored when calculating software effort.
- The equation used to calculate software effort is a simple linear regression, which is the multiplication of software size by twenty. Here, there are two main concerns. First, this equation is applied on any software size. Our experiments show that a software equation used with large projects should be different from the one used with small projects. Secondly, this equation assumes that the relationship between software size and effort is linear. Longstreet [22] stated that when estimation is based on the Function Points method, the effort required to

develop one Function Point is between 0.5 and 5 hours for small projects (less than 100 function points) and between 20 to 60 hours for large projects (greater than 7,000 function points). The UCP is similar to the Function Point model in the way that both methods can be applied in the Requirements stage of the software life cycle and both are independent of the programming language and the topology used to develop the project. We believe that this non-linearity between software effort and size in the Function Point model is valid as well as in the UCP. For instance, if the effort required in building a software project of size 250 UCP is 5,000 person-hours, the effort needed to build the same project type of size 500 UCP would be more than 10,000 person-hours. This is because the larger the project is, the larger the team required to build this project [23]. When the number of the team members increases, the number of the communication paths among this team will dramatically increase as shown in Equation (2.24) [24], and consequently, this requires more effort for the team communication and project management.

$$Communication\_Paths = \frac{N(N-1)}{2}. \qquad (2.24)$$

Where "N" is the number of people in the team.

Although many related work tried to address some of the limitations of the UCP model, many issues still exit and these issues are tackled by our model.

2.4.2 Expert Judgement

Expert judgement involves consulting a group of experts to use their experiences to propose an estimation of a given project [25]. The Delphi technique is used to provide communication and cooperation among the experts. The Delphi technique is summarized as follows [26]:

1. A coordinator provides each expert with a project's specifications and a form to be filled.
2. The coordinator calls for a group meeting with the experts to discuss any issues.
3. The experts will anonymously fill the forms.
4. The coordinator receives the forms and prepares a summary for the estimation.
5. The coordinator calls for a meeting to discuss with the experts the proposed estimation values, and especially when these values vary dramatically among experts.
6. The experts fill the estimation forms again. Steps 4 to 6 are repeated until a satisfaction has been reached.

The main advantage of this method is that the final estimation report can be reached in a reasonable period. Moreover, this method is relatively inexpensive and can be accurate in comparison with other models especially, when the experts have a solid knowledge of the problem domain of the proposed project.

The main limitation of the expert judgement model is that this method is very subjective and it lacks standardizations and thus, cannot be reusable. Another drawback of this method is the lack of analytical argumentation because of the frequent use of phrases such as "*I believe that …*" or "*I feel that …*" [27].

2.4.3 Estimation by Analogy

Estimation by analogy is a method in which the proposed project is compared to similar historical projects where all required information about the historical projects is documented. Estimation by analogy is actually a systematic form of expert judgement since experts look for analogies. The main steps to conduct analogy by estimation include:

1. The characteristics of the proposed project are identified.
2. Similar completed projects are selected.
3. Estimation of the proposed project is conducted.

The main advantage of this method is that estimators are using their expertise to estimate new projects based on actual completed projects. Furthermore, this method is relatively fast and reliable.

The main disadvantage of estimation by analogy is that companies are required to maintain a well-designed knowledge repository. Moreover, companies should have a good number of historical projects; however, this method cannot be applied in new companies.

2.4.4 Soft Computing Models

Soft computing models include neural network models, fuzzy logic models, genetic algorithm models and hybrid models such as, neuro-fuzzy and neuro-genetic models. These models can be applied in two main situations. First, these models can be applied as standalone models that take several inputs such as software size and productivity, then provide an output such as software effort. These models can be trained using historical projects. Another usage of these models is that they can be used to calibrate some parameters or weights of algorithmic models such as COCOMO parameters and function point model weights. Soft computing models can also be used with estimation by analogy to increase the accuracy of estimation.

2.5 Related Work

In addition to the above literature in software estimation, some related work for software estimation is listed as follows:

Periyasamy et al. [28] extended the UCP model by classifying actors into seven groups. The weight proposed for actors varies between 0.5 and 3.5. Moreover, the authors proposed four types of use cases and assigned new weights for each use case. The weight of a use case is determined based on the number of associations between actors and the use case. The authors also proposed a new method to calculate software size from use cases; however, the authors have not evaluated their method against any related models.

Wang et al. [29] extended the UCP model by constructing a probabilistic cost model by integrating a fuzzy set theory with Bayesian Belief Networks with the UCP model. The proposed method was evaluated using two financial projects of efforts 3,016 and 4,459 person-hours respectively. These projects are located in China and developed using Java programming language. The evaluation of the extended UCP shows slim improvement in comparison with the original UCP.

Schneider et al. [30] mentioned that when calculating software effort, instead of multiplying the size by 20 (as the original UCP model), Environmental factors should be evaluated because these factors contribute to the efficiency of the team developing the project. If the efficiency is fair, then 20 person-hours per UCP should be used. If the efficiency is low, then 28 person-hours per UCP should be used. If the efficiency is very low, then the project team should be reconstructed because very low efficiency indicates that the project is at significant risk of failure with this team. Another approach can be considered when the efficiency is very low by taking 36 person-hours for 1 UCP. The main limitation of Schneider's approach is that the effort required to develop one UCP is either 20, 28 or 36 person-hours.

Azzeh et al. [31] and [32] proposed two models for software effort estimation. The first one is an estimation- by-analogy model based on the integration of fuzzy set theory with grey relational analysis and fuzzy numbers. However, the second model is based on analogy estimation with fuzzy numbers and can be used in the early stages of the software life cycle. Both models were evaluated using five different datasets such as

International Software Benchmarking Standards Group (ISBSG), Desharnais, Kemerer, Albrecht & Gaffney and COCOMO 81. MMRE, MdMRE, MMER and PRED(25) were used as evaluation criteria. Results proved that the proposed models are competitive when compared with other models such as case-based reasoning, multiple linear regression, stepwise regression and artificial neural networks.

Pendharkar et al. [33] developed a Bayesian network to predict software development effort. The proposed model can incorporate decision making risks. The model was evaluated using 33 industrial projects and was compared with other neural network and regression tree forecasting models. The authors proved that their model can be a competitive model for software effort prediction based on the absolute error criterion.

Jiang et al. [34] and Xia et al. [35] built linear regression models with a logarithmic transformation based on function points using ISBSG data. Xia et al. used the regression model as an activation function in a neural network to calibrate the weights in the function point model. However, Jiang et al. used the regression model to study the effect of software size on development effort and software quality. The main concern of these models is that they ignore the influence of the non-functional requirements on estimation.

Park et al. [36] proposed a neural network for software effort estimation. This model takes six inputs and the accuracy of the proposed model was compared with the accuracy of human expert judgments and two traditional regression models. The evaluation was conducted on 148 IT projects and results proved that the proposed neural network gives

better results than existing regression models based on the magnitude of relative error (MRE).

Idri et al. [37] proposed two Radial Basis Neural Network model for software effort estimation. Each of the RBFNN models uses different formula to calculate the width of the RBF functions. The model was trained using COCOMO 81 and Tukutuku datasets and evaluated based on MMRE and PRED criteria.

Idri et al. [38] investigates the use of the RBFNN models in software estimation and especially the role of the hidden layer. In their paper, the authors use two clustering techniques; the C-means and the APC-III. A comparison between these techniques was conducted using COCOMO 81 and Tukutuku datasets.

Reddy et al. [39] proposed a RBFNN model for software effort estimation. The model was trained based on the k-mean clustering algorithm and was evaluated using the COCOMO 81 dataset.

Shin et al. [40] presented an objective modeling methodology to determine the RBFNN model parameters using their SG algorithm. The model was then used to predict software effort using the NASA dataset.

Heiat [41] compared a neural network model with regression models. The evaluation was conducted on 67 projects from three different sources. The author concluded that the neural network model was competitive to regression models when third generation language was used. However, regression models gave better results when combinations

of third and fourth generation language projects were used. The evaluation criterion used was the mean absolute percentage error (MAPE).

Tan et al. [42] proposed a new LOC estimation method for information systems based on their conceptual data models through a multiple linear regression model. The authors evaluated their work using open source and industrial projects.

Anvik et al. [43] used machine learning techniques to create recommenders to triage bug reports that can be useful to streamline the development process.

Lopez-Martin [44], [45], [46] and [47] created regression models from short scale programs and from ISBSG repository. The author also developed fuzzy logic and neural network models such as Feed-Forward and General Regression Neural Networks. The authors proved that these models can be used as alternatives to regression models to predict software effort. The evaluation criteria used were the Mean of the Magnitude of Relative Error (MMRE) and the Mean of Magnitude of error Relative to the Estimate (MMER).

Shepperd and Schofield [48] proposed a software estimation model using analogy. The model was evaluated based on 275 projects from nine different industrial datasets. The authors argue that estimation model based on analogy surpasses other algorithmic models based upon stepwise regression.

Jørgensen et al. [49] applied regression toward the mean (RTM) method with analogy for software effort estimation. The proposed model was evaluated based on 5 different

datasets. The authors argued that the accuracy of software effort estimation using analogy would be improved when using RTM.

Other machine learning models exist and are used to improve the accuracy of software estimation. Examples of these models include [50], [51], [52], [53], [54], [55] and [56].

None of the above work developed neural network models to predict software effort from use case diagrams. Furthermore, none has thoroughly investigated the non-linear relationship between software size and effort the way it is addressed in this thesis.

References

[1] L. A. Zadeh, "Fuzzy sets," *Information and Control,* vol. 8, pp. 338-353, 1965.

[2] R. P. Lippman, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine,* vol. 3, no.2, pp. 4-22, 1987.

[3] O. C. Celebi, "Tutorial: Neural Networks and Pattern Recognition Using MATLAB," 2011.

[4] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks,* vol. 6, pp. 525-533, 1993.

[5] D. S. Broomhead and D. Lowe, "Multivariable Functional Interpolation and Adaptive Networks," *Complex Systems,* vol. 2, pp. 321-355, 1988.

[6] Chien-Cheng Lee, Pau-Choo Chung, Jea-Rong Tsai and Chein-I Chang, "Robust radial basis function neural networks," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on,* vol. 29, pp. 674-685, 1999.

[7] D. F. Specht, "A general regression neural network," *Neural Networks, IEEE Transactions on,* vol. 2, pp. 568-576, 1991.

[8] L. C. Briand, K. E. Emam, D. Surmann, I. Wieczorek and K. D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," in *International Conference on Software Engineering,* 1999, pp. 313-322.

[9] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell and M. J. Shepperd, "What Accuracy Statistics Really Measure," *IEE Proc. -Softw,* vol. 148, no. 3, pp. 81-85, June, 2001.

[10] M. A. Parthasarathy, *Practical Software Estimation: Function Point Methods for Insourced and Outsourced Projects.* Upper Saddle River, NJ: Addison-Wesley, 2007.

[11] L. C. Briand and I. Wieczorek, "Resource Estimation in Software Engineering," *Encyclopedia of Software Engineering,* vol. 2, pp. 1160-1196, 2002.

[12] B. W. Boehm, *Software Engineering Economics.* Prentice-Hall, 1981.

[13] L. H. Putnam, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering,* vol. SE-4, pp. 345-361, 1978.

[14] D. D. Galorath and M. W. Evans, *Software Sizing, Estimation, and Risk Management.* Boston, MA, USA: Auerbach Publications, 2006.

[15] P. Kok, B. A. Kitchenham and J. Kirakowski, "The MERMAID approach to software cost estimation," in *Esprit Technical Week,* 1990.

[16] B. Boehm, C. Abts, W. Brown and S. Chulani, *Software Cost Estimation with COCOMO II*. Upper Saddle River, New Jersey: Addison Wesley, 2000.

[17] A. Albrecht, "Measuring application development productivity," in *IBM Application Development Symp.* 1979, pp. 83-92.

[18] A. J. Albrecht and J. E. Gaffney Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, vol. SE-9, pp. 639-648, 1983.

[19] IFPUG. IFPUG counting practices manual. 1986 [Online]. Available: www.ifpug.org.

[20] G. Karner, "Resource Estimation for Objectory Projects," *Objective Systems*, 1993.

[21] Y. Ossia. IBM haifa research lab. *IBM Haifa Research Lab* [Online]. 2011. Available: https://www.research.ibm.com/haifa/projects/software/nfr/index.html.

[22] D. Longstreet, "Estimating Software Effort," *Software Metrics*, 2008.

[23] P. C. Pendharkar and J. A. Rodger, "The relationship between software development team size and software development cost," *Commun ACM*, vol. 52, pp. 141-144, January, 2009.

[24] P. Kuthiala. PMBOK – PMP project management. *JustPM Blog* [Online]. 2009. Available: http://www.justpmblog.com/2009/03/05/pmp-project-management-2/.

[25] R. T. Hughes, "Expert judgement as an estimating method," *Information and Software Technology,* vol. 38, pp. 67-75, 1996.

[26] N. Dalkey and O. Helmer, "An Experimental Application of the Delphi Method to the Use of Experts," *Management Science,* vol. 9, pp. pp. 458-467, 1963.

[27] M. Jørgensen, "Forecasting of software development work effort: Evidence on expert judgement and formal models," *International Journal of Forecasting,* vol. 23, pp. 449-462, 2007.

[28] K. Periyasamy and A. Ghode, "Cost estimation using extended use case point (e-UCP) model," in *International Conference on Computational Intelligence and Software Engineering,* 2009.

[29] F. Wang, X. Yang, X. Zhu and L. Chen, "Extended use case points method for software cost estimation," in *International Conference on Computational Intelligence and Software Engineering,* 2009.

[30] G. Schneider and J. P. Winters, *Applied use Cases, Second Edition, A Practical Guide.* Addison-Wesley, 2001.

[31] M. Azzeh, D. Neagu and P. Cowling, "Fuzzy grey relational analysis for software effort estimation," *Empirical Software Engineering,* vol. 15, pp. 60-90, 2010.

[32] M. Azzeh, D. Neagu and P. I. Cowling, "Analogy-based software effort estimation using Fuzzy numbers," *Journal of Systems and Software,* vol. 84, pp. 270-284, 2011.

[33] P. C. Pendharkar, G. H. Subramanian and J. A. Rodger, "A probabilistic model for predicting software development effort," *Software Engineering, IEEE Transactions on,* vol. 31, pp. 615-624, 2005.

[34] Z. Jiang, P. Naudé and B. Jiang, "The effects of software size on development effort and software quality," *International Journal of Computer and Information Science and Engineering,* vol. 1, pp. 230-234, 2007.

[35] W. Xia, L. F. Capretz, D. Ho and F. Ahmed, "A new calibration for Function Point complexity weights," *Information and Software Technology,* vol. 50, pp. 670-683, 2008.

[36] H. Park and S. Baek, "An empirical validation of a neural network model for software effort estimation," *Expert Systems with Applications,* vol. 35, pp. 929-937, 10, 2008.

[37] A. Idri, A. Zakrani and A. Zahi, "Design of Radial Basis Function Neural Networks for Software Effort Estimation," *International Journal of Computer Science Issues,* vol. 7, pp. 11-17, 2010.

[38] A. Idri, A. Zahi, E. Mendes and A. Zakrani, "Software Cost Estimation Models Using Radial Basis Function Neural Networks," *Software Process and Product Measurement,* vol. 4895, pp. 21-31, 2008.

[39] C. S. Reddy, P. S. Rao, K. Raju and V. V. Kumari, "A New Approach For Estimating Software Effort Using RBFN Network," *International Journal of Computer Science and Network Security*, vol. 8, pp. 237-241, 2008.

[40] M. Shin and G. A.L., "Empirical data modeling in software engineering using radial basis functions," *Software Engineering, IEEE Transactions on*, vol. 26, pp. 567-576, 2000.

[41] A. Heiat, "Comparison of artificial neural network and regression models for estimating software development effort," *Information and Software Technology*, vol. 44, pp. 911-922, 2002.

[42] H. B. K. Tan, Y. Zhao and H. Zhang, "Conceptual data model-based software size estimation for information systems," *ACM Transactions on Software Engineering and Methodology*, vol. 19, pp. 4:1-4:37, oct, 2009.

[43] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology*, vol. 20, pp. 10:1-10:35, aug, 2011.

[44] C. Lopez-Martin, "A fuzzy logic model for predicting the development effort of short scale programs based upon two independent variables," *Applied Soft Computing*, vol. 11, pp. 724-732, 1, 2011.

[45] C. Lopez-Martin, "Applying a general regression neural network for predicting development effort of short-scale programs," *Neural Computing & Applications,* vol. 20, pp. 389-401, 2011.

[46] C. Lopez-Martín, C. Yanez-Marquez and A. Gutierrez-Tornes, "Predictive accuracy comparison of fuzzy models for software development effort of small programs," *Journal of Systems and Software,* vol. 81, pp. 949-960, 2008.

[47] C. Lopez-Martin, C. Isaza and A. Chavoya, "Software development effort prediction of industrial projects applying a general regression neural network," *Empirical Software Engineering,* vol. 17, pp. 1-19, 2011.

[48] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *Software Engineering, IEEE Transactions on,* vol. 23, pp. 736-743, 1997.

[49] M. Jørgensen, U. Indahl and D. Sjøberg, "Software effort estimation by analogy and "regression toward the mean"," *J. Syst. Software,* vol. 68, pp. 253-262, 12/15, 2003.

[50] W. L. Du, D. Ho and L. F. Capretz, "Improving Software Effort Estimation Using Neuro-Fuzzy Model with SEER-SEM," *Global Journal of Computer Science and Technology,* vol. 10, pp. 52-64, 2010.

[51] Y. Li, M. Xie and T. Goh, "Adaptive ridge regression system for software cost estimating on multi-collinear datasets," *Journal of Systems and Software,* vol. 83, pp. 2332-2343, 2010.

[52] G. Kousiouris, T. Cucinotta and T. Varvarigou, "The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks," *Journal of Systems and Software*, vol. 84, pp. 1270-1291, 2011.

[53] X. Huang, D. Ho, J. Ren and L. F. Capretz, "Improving the COCOMO model using a neuro-fuzzy approach," *Appl. Soft Comput.*, vol. 7, no. 1, pp. 29-40, 2007.

[54] A. Mittal, K. Parkash and H. Mittal, "Software cost estimation using fuzzy logic," *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 1, pp. 1-7, 2010.

[55] I. Attarzadeh and S. H. Ow, "Software Development Cost and Time Forecasting Using a High Performance Artificial Neural Network Model," *Intelligent Computing and Information Science*, vol. 134, pp. 18-26, 2011.

[56] I. Attarzadeh and S. H. Ow, "Improving the accuracy of software cost estimation model based on a new fuzzy logic model," *World Applied Sciences Journal*, vol. 8, pp. 177-184, 2010.

Chapter 3

3. MLP and Linear Regression Models[2]

3.1 Introduction

This chapter presents our preliminary research in creating a linear regression with a logarithmic transformation model, as well as a Multilayer Perceptron (MLP) neural network. In this chapter, we introduce two main factors that contribute to software effort estimation which include software size and team productivity. Software size is estimated using the method proposed by the use case point (UCP) (section 2.4.1.4). Team productivity is calculated based on the Environmental Factors (EF) (Table 2-6) proposed

[2] Part of this chapter was published in the 2011 International Conference on Computer and Software Modeling, in the 23rd IEEE International Conference on Tools with Artificial Intelligence and in the 2011 IEEE International Conference on Intelligent Computing and Intelligent Systems. An extended version of these papers has been submitted to the Journal of Systems and Software (Elsevier).

1. **Ali Bou Nassif**, Danny Ho and Luiz Fernando Capretz: Regression Model for Software Effort Estimation Based on the Use Case Point Model, *2011 International Conference on Computer and Software Modeling (ICCSM 2011),* September 2011, Singapore (Published).
2. **Ali Bou Nassif**, Luiz Fernando Capretz, Danny Ho, "Estimating Software Effort Based on Use Case Point Model Using Sugeno Fuzzy Inference System," ictai, pp.393-398, 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence, Boca Raton, Florida, USA, 2011 (Published).
3. **Ali Bou Nassif**, Luiz Fernando Capretz, Danny Ho, " A Regression Model with Mamdani Fuzzy Inference System for Early Software Effort Estimation Based on Use Case Diagrams," icis, pp.615-620, 2011 IEEE International Conference on Intelligent Computing and Intelligent Systems, Guangzhou, Guangdong, China, 2011 (Published).
4. **Ali Bou Nassif**, Danny Ho and Luiz Fernando Capretz, "Towards an Early Software Estimation Using Log-Linear Regression and a Multilayer Perceptron Model", Journal of Systems and Software (Elsevier), 2012 (Under Review).

by the UCP model. The MLP model takes nine inputs which include software size and the eight environmental factors.

Section 3.2 proposes the linear regression model with fuzzy logic as well as the MLP approach. Section 3.3 demonstrates an assessment of the proposed models and provides some discussion about the results. Section 3.4 lists some threats to validity. Finally, Section 3.5 concludes the chapter.

3.2 Research Methodology and Models' Evaluation

This section presents the proposed regression model, the calibration of this model using fuzzy logic and the proposed neural network model. Moreover, evaluations of these models are demonstrated.

3.2.1 Regression Model

Equation (2.23) shows how software effort is calculated from software size based on the original Use Case Point (UCP) model. There are two main shortcomings of this equation. First, the relationship between software effort and size is linear and this assumption does not reflect the actual situation in the software industry. Longstreet [1] stated that when estimation is based on the Function Points method, the effort required to develop one Function Point is between 0.5 and 5 hours for small projects (less than 100 function points) and between 20 to 60 hours for large projects (greater than 7,000 function points). The UCP is similar to the Function Point model in the way that both methods can be applied in the Requirements stage of the software life cycle and both are independent of the programming language and the topology used to develop the project. We believe that

this non-linearity between software effort and size in the Function Point model is valid as well as in the UCP. McConnell [2] states that "People naturally assume that a system that is 10 times as large as another system will require something like 10 times as much effort to build. But the effort for a 1,000,000 LOC system is *more* than 10 times as large as the effort for a 100,000 LOC system. Using software industry productivity averages, the 10,000 LOC system would require 13.5 staff months. If effort increased linearly, a 100,000 LOC system would require 135 staff months. But it actually requires 170 staff months". This shows that when software size increases, software effort would increase but with a non-linear relationship. The second shortcoming is that this equation does not take into consideration the productivity of the team that is developing the software. In the proposed model, a novel regression analysis is applied to generate a new equation to calculate software effort. The new equation takes into account the non-linear relationship between software effort and size as well as the productivity factor of the team. Furthermore, the value of the productivity factor is proposed using a multiple linear regression model of two independent variables.

The general equation of software effort can be represented as [3]:

$$Effort = \frac{Complexity}{Productivity} \times Size. \tag{3.1}$$

where *Complexity* is the complexity factor of a project and *Productivity* is the productivity factor of the team that is developing this project. To find the non-linear relationship between software size and software effort, regression analysis was applied on 125 educational and industrial projects (see Appendix C) that have similar projects

complexity and team productivity. Thus, at this point, complexity and productivity factors are ignored and software effort is a function of software size only. Questionnaire I (Appendix A) was used to collect data. To obtain accurate results in regression analysis, data should be normally distributed [4]. If data were normally distributed, the regression equation would be:

$$Effort = a \times Size + b. \tag{3.2}$$

where a and b are constants.

Several experiments were conducted using Minitab version 16 to determine how data were distributed. The histograms of software size (Figure 3-1) and software effort (Figure 3-2) show that data are not normally distributed. Generating regression models from data based on Figures (3-1) and (3-2) is possible but this will lead to poor results. For this reason, data were normalized using logarithmic transformation. After normalization, data (ln size and ln effort) became normally distributed (Figures 3-3 and 3-4). The regression equation after logarithmic transformation is:

$$\ln(Effort) = c \times \ln(Size) + d. \tag{3.3}$$

Where c and d are constants. Equation (3.3) can be rewritten as:

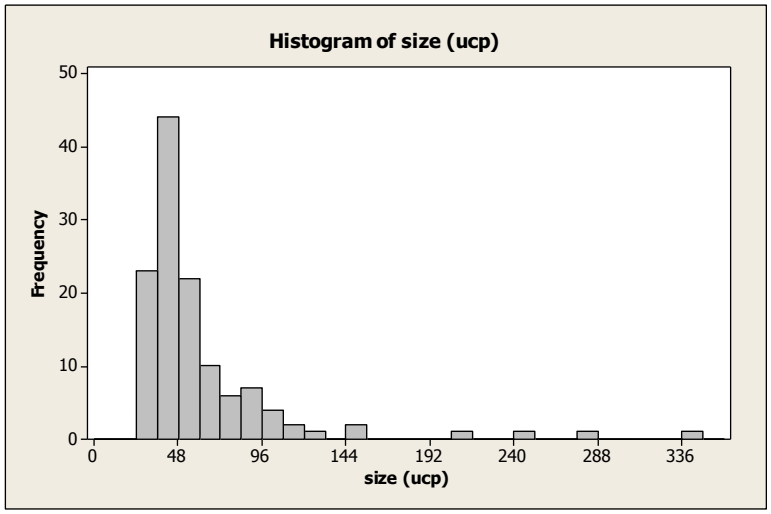


Figure 3-1 Histogram of size

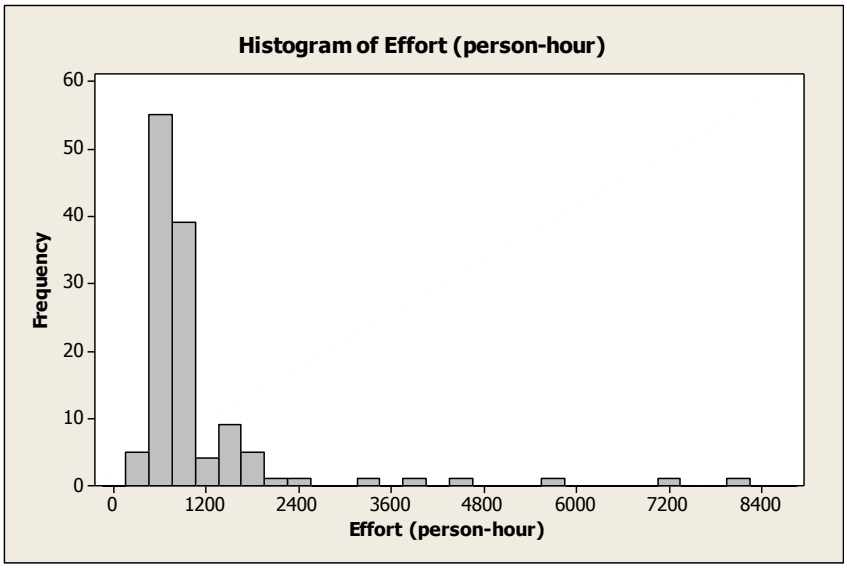


Figure 3-2 Histogram of effort

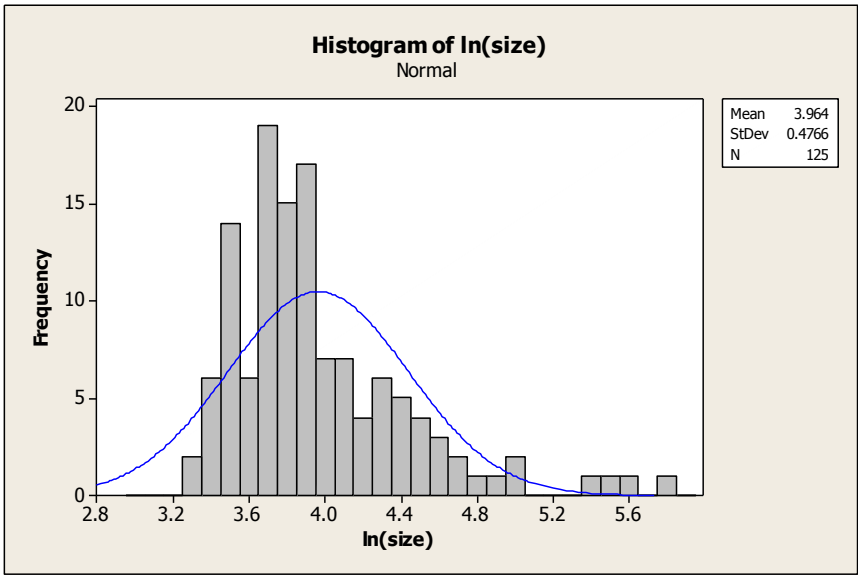


Figure 3-3 Histogram of ln(Size)

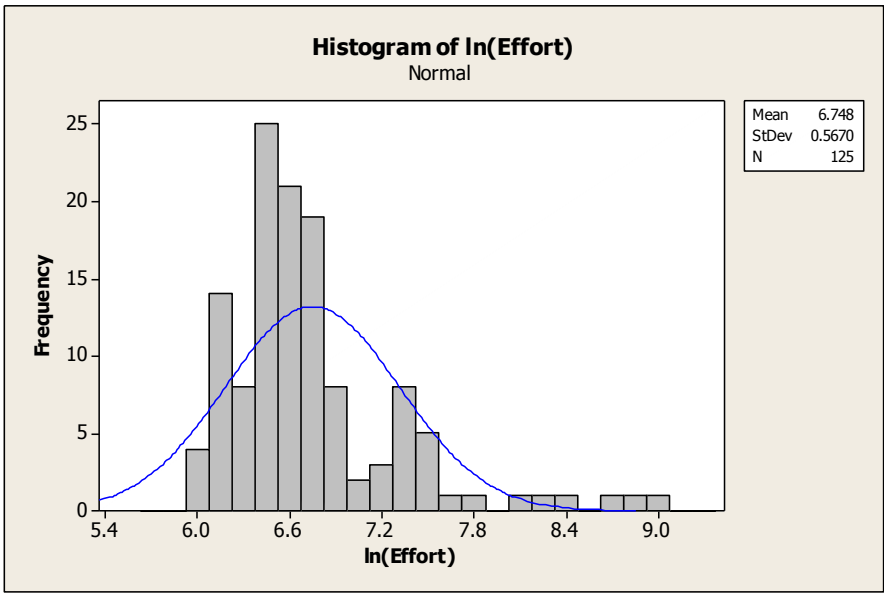


Figure 3-4 Histogram of ln(Effort)

$$Effort = A \times (Size)^{B}. \tag{3.4}$$

Using Minitab, the values of A and B are 8.16 and 1.17 respectively. The proposed regression equation is:

$$Effort = 8.16 \times (Size)^{1.17}. \quad (3.5)$$

Where *Size* is the software size in UCP and *Effort* is the software effort in person-hours. For instance, Equation (3.5) shows the non-linear relationship between Effort and Size and ignores the Complexity and Productivity factors. The main equation of software effort is expressed in Equation (3.6).

Figure (3-5) shows the relationship between software size and effort based on the original UCP model (Equation 2.23) and the proposed regression model (Equation 3.5). The straight line (blue line) represents Karner's model (original UCP model) and the dotted line represents the proposed regression model. This comparison shows that the non-linear relationship is not significant for small projects (less than 200 UCP). On the other hand, the non-linear relationship stands out for mid-size and large projects. The proposed regression model also shows that when software size becomes larger and larger, software effort is exponentially increasing. For instance, when software size is 1,000 UCP, software effort based on the regression model is larger than the software effort based on the original UCP model by 30%. For instance, Figure (3-5) answers the sixth research question raised in Section 1.2.

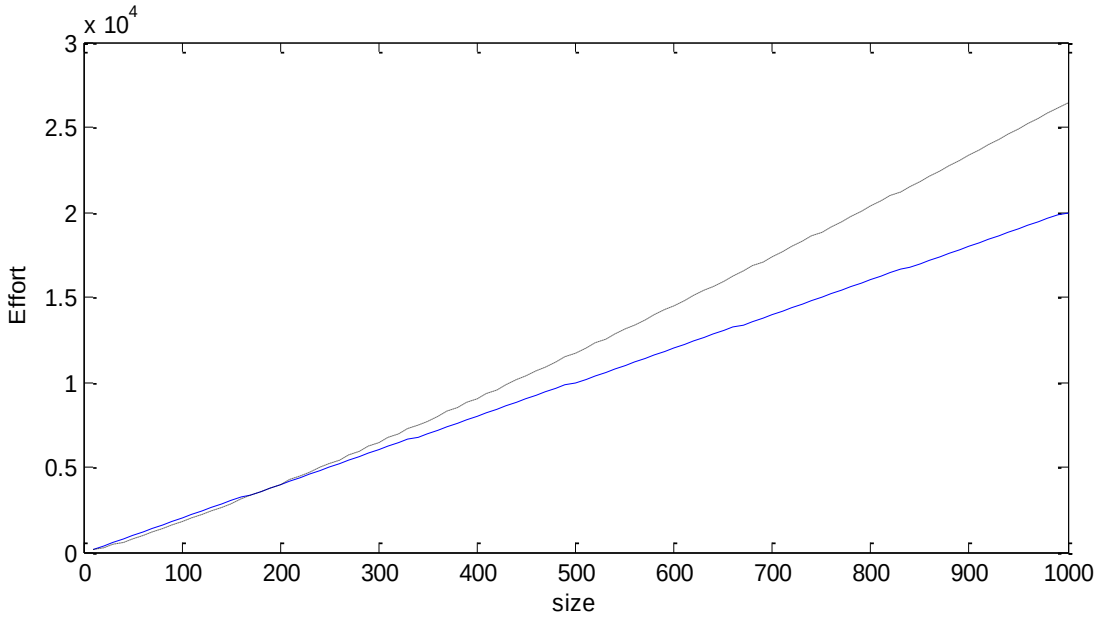


Figure 3-5 Comparison between software size and software effort

It is very important to test and validate the proposed regression equation (Equation 3.5) because this equation will be the core of the regression model (Equation 3.6 shows the main regression equation of the model). To thoroughly validate this equation, several techniques were used. These include the probability plot (aka Q-Q plot), the coefficient of determination R^2, Spearman and Pearson coefficients, Analysis of Variance (ANOVA) and the model's parameters. The probability plot (Q-Q plot) compares two probability distributions by plotting their quantiles against each other. It shows if the relationship between these two distributions is linear or not. Since the regression analysis was applied after the logarithmic transformation, the Q-Q graphs of normalized size and normalized effort were performed as shown in Figures (3-6) and (3-7) respectively. The figures

show that 95% of normalized data (size and effort) are linearly distributed and thus, the regression equation (Equation 3.5) is valid.

Another method was applied to measure the accuracy of the regression equation (Equation 3.5). For this purpose, the value of the coefficient of determination R^2 was measured. R^2 is the percentage of variation in Effort explained by the variable Size. An acceptable value of R^2 is ≥ 0.5 [5]. The value R^2 reported for the regression model in Equation (3.5) is 0.972. Approximately 97 % of the variation in Effort can be explained by the variable Size. This shows a strong relation between Size and Effort.

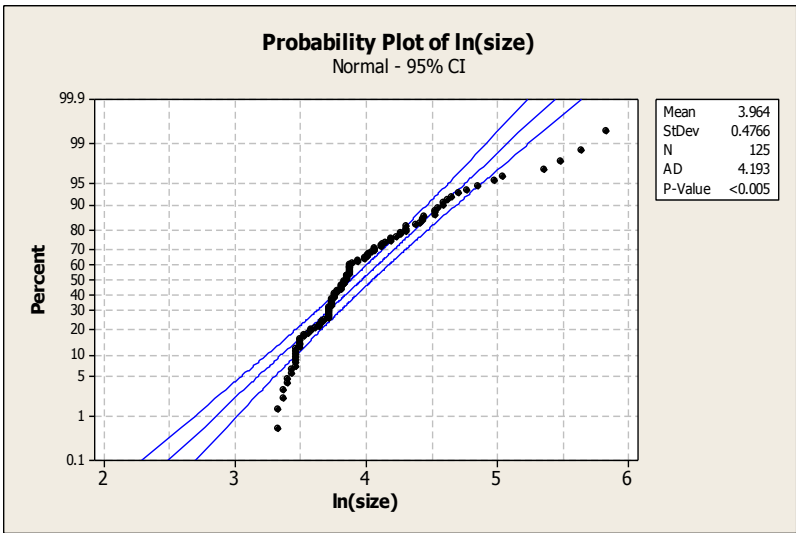


Figure 3-6 Q-Q plot for normalized size

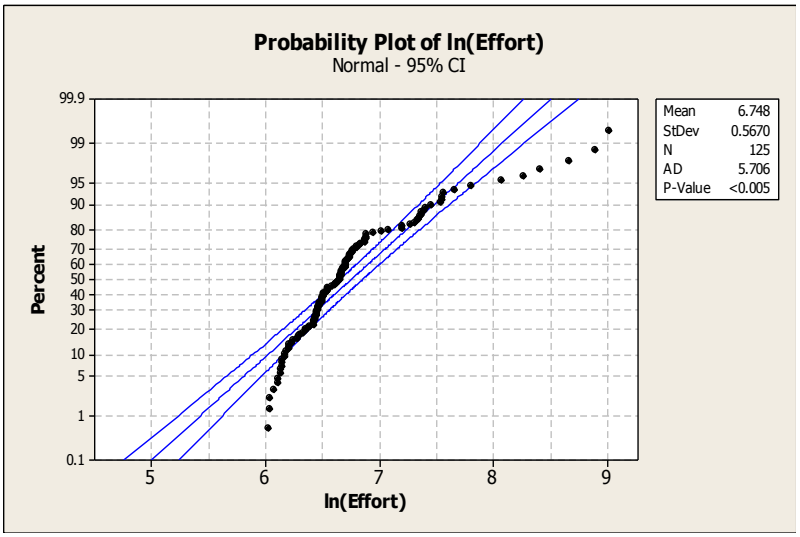


Figure 3-7 Q-Q plot for normalized effort

To thoroughly test the regression model, Spearman [6] and Pearson [7] coefficients were determined to measure the correlation strength between the Effort and Size. The coefficients range of both Spearman and Pearson is between [-1, 1]. The value 0 means that these two variables are not correlated. A positive value represents a positive correlation. Larger coefficient values correspond to stronger correlations. On the contrast, negative values mean negative correlations. In our experiments, the Spearman and Pearson coefficients are 0.98 and 0.97 respectively. This shows that the two variables Effort and Size have a strong positive relationship.

Table (3-1) depicts the ANOVA for the regression equation. From the "p" value of ANOVA, we notice that there is a significant relationship among the variables at the 99% confidence level. For instance, DF, SS, MS, F and P correspond to Degrees of Freedom, Sum of Squares, Mean Square, F Ratio and P Ratio respectively. However, Table (3-2) shows the model's parameters to determine if the model can be simplified. The highest

"p" value in Table (3-2) is 0.000. Since the "p" value of each variable is less than 0.05, all independent variables are significant at the 95% confidence level.

Based on the above experiments and results, the regression equation represents the non-linear relationship between software size and effort with high percentage of accuracy. By taking into consideration Equation (3.5), the main equation for software effort in the proposed model can be expressed as follows:

$$Effort = 8.16 \times \frac{\text{Pr}oject\_Complexity}{\text{Pr}oductivity} \times (Size)^{1.17}. \qquad (3.6)$$

The second step of the proposed model is to calculate the values of Project_complexity and Productivity. Table (2-5) presents some technical factors that represent the complexity of a project. We will assume that Karner's technical factor TF can represent the Project_Complexity factor during the estimation of UCP and consequently, the Project_Complexity factor in Equation (3.6) can be ignored. The main effort equation will become:

$$Effort = \frac{8.16}{\text{Pr}oductivity} \times (Size)^{1.17}. \qquad (3.7)$$

Equation (3.7) shows that Effort is inversely proportional to productivity. For instance, Equation (3.7) answers the second research question proposed in Section 1.2. With respect to productivity, Table (2-6) lists some productivity attributes. In the original UCP model, productivity factor is only included when estimating the adjusted UCP size. Schneider et al. [8] included the productivity factor while calculating software effort as discussed in Section 1.5. We believe that the productivity factor should be included in the

software effort equation. Based on Table (2-6), the highest productivity factor is achieved when the value of the factors E1 to E6 is 5 and the value of the factors E7 and E8 is 0. If we assume that prod_sum = $\sum_{i=1}^{8} E_i \times W_i$, this implies that the value of prod_sum is 32.5. On the other hand, the lowest productivity factor is achieved when the value of E1 to E6 is set to 0 and the value of E7 and E8 is set to 5. This implies that the value of prod_sum is -10. In the proposed approach, the productivity factor in Equation (3.7) is determined based on the value of prod_sum. To discover the influence of prod_sum on software effort, a multiple linear regression equation was generated using Minitab version 16 with two independent variables (*Size* and *prod_sum*) as shown in Equation (3.8).

$$Effort = 409 + (24.9 \times Size) - (52.8 \times prod\_sum). \qquad (3.8)$$

This equation shows that when software size increases, software effort increases. However, when the productivity of the team (prod_sum) increases, software effort decreases. This interpretation makes sense in the software industry and it is compatible with the influences of software size and team productivity proposed in Equation (3.7).

The value of the coefficient of determination R^2 of Equation (3.8) is 0.861. This indicates that approximately 86 % of the variation in Effort can be explained by the independent variables *size* and *prod_sum*. Tables (3-3) and (3-4) show the ANOVA and model parameters of Equation (3.8). ANOVA shows that there is a significant relationship among the variables at the 99% confidence level. The model's parameters show that the least value of "p" is 0.009 which is less than 0.05 that indicates that all independent variables are significant at the 95% confidence level.

Table 3-1 ANOVA for Equation 3.5

Source	DF	SS	MS	F	P
Regression	1	38.756	38.756	4319.13	0.000
Residual	123	1.104	0.009		
Total	124	39.860			

Table 3-2 Model parameters for Equation 3.5

Predictor	Coef	SE Coef	T	P
Constant	2.09835	0.07126	29.45	0.000
ln(size)	1.17314	0.01785	65.72	0.000

Table 3-3 ANOVA for Equation 3.8

Source	DF	SS	MS	F	P
Regression	2	174055066	87027533	300.51	0.000
Residual Error	97	28090762	289595		
Total	99	202145827			

Table 3-4 Model parameters for Equation 3.8

Predictor	Coef	SE Coef	T	P
Constant	408.5	154.1	2.65	0.009
Size	24.939	1.120	22.26	0.000
Prod_sum	-52.75	11.77	-4.48	0.000

From the above results, we deduce that the proposed multiple linear regression equation is valid and it is used to determine the productivity factor in Equation (3.7) based on the value of the variable *prod_sum*. Since the value of prod_sum varies between [-10, 32.5], it is difficult to predict the value of *productivity* in Equation (3.7) based on each value of

prod_sum. For this reason, the *productivity* variable is depicted based on four main ranges of *prod_sum*. This is analogous to the representation of cost drivers in the COCOMO model where each cost drivers are classified according to five or six levels (from very low, to very high). After that, fuzzy logic is used to adjust the values of the *productivity* variable. Since the *prod_sum* variable falls between [-10, 32.5], the main four regions of this variable are selected as between [-10, 0], between [1, 10], between [11, 20] and between [21, 32.5]. To find the influence of *prod_sum* on *Effort* in Equation (3.8), four values of *prod_sum* are selected such that each value belongs to each of the aforementioned main regions. To minimize the influence of the *size* variable on *Effort* and only focus on the influence of *prod_sum*, the value of the *size* variable is the same for each value of *prod_sum*. The selected value of *size* is 80 UCP because the value "80" is considered as a medium-size project with respect to the pool of the projects used to generate the regression equation. Based on this information and according to Equation (3.8), the following rules can be deduced:

- If *size* is 80 and *prod_sum* is -7 then *Effort* is 2770. (-7 falls between [-10, 0])
- If *size* is 80 and *prod_sum* is 5 then *Effort* is 2137. (5 falls between [1, 10])
- If *size* is 80 and *prod_sum* is 16 then *Effort* is 1556. (16 falls between [11, 20])
- If *size* is 80 and *prod_sum* is 26 then *Effort* is 1028. (26 falls between [21, 32.5])

If we substitute the values of *size* and *Effort* of the aforementioned four rules in Equation (3.7), the values of the *productivity* variable are 0.4, 0.7, 1 and 1.3 respectively.

Equation (3.7) represents the main proposed regression model for software effort estimation, where *Effort* is the software effort in person-hours, *size* is the software size in UCP and the value of *productivity* is depicted in Table 3-5.

3.2.2 Fuzzy Logic Approach

Table (3-5) shows the values of the *productivity* variable of Equation (3.7). The productivity factors were predicted using the multiple linear regression model. Each productivity factor value was given a description. The main drawback of the productivity factor is that the values are crisp and there is no graduation in the productivity factor values as the value of prod_sum increases. For instance, if the value of prod_sum is 10, the productivity factor is 0.7, however, if the value of prod_sum is 11, the value of the productivity factor is 1. To tackle this drawback, a fuzzy logic approach has been used.

A fuzzy logic approach is applied on the proposed regression model to adjust the values of the productivity factor. In the proposed approach, we used two types of fuzzy systems. This includes Mamdani [9] and Sugeno [10]. Both Mamdani and Sugeno can have the same input (membership functions). However, the main difference between these two models is that the output of Mamdani can take any membership function like the input but the output of Sugeno can be either constant or a straight line. The input membership of the fuzzy logic system used is Trapezoidal because each input has a range of values (e.g. between 1 and 10). The output membership used is Triangular because each output has a fixed value which is represented by a triangle's vertex. The method used in the Defuzzification stage is the centroid since this is the default and most common used

method. Matlab version 2010b was used to conduct the experiments of the fuzzy logic approach. Figures (3-8) and (3-9) show the input and the output memberships of Mamdani fuzzy logic system, respectively.

Table 3-5 Productivity factor

prod_sum = $\sum_{i=1}^{8} E_i \times W_i$	**Productivity Description**	**Productivity Factor**
Less than 0	Very Low	0.4
Between 1 and 10	Low	0.7
Between 11 and 20	Average	1
Greater than 20	High	1.3

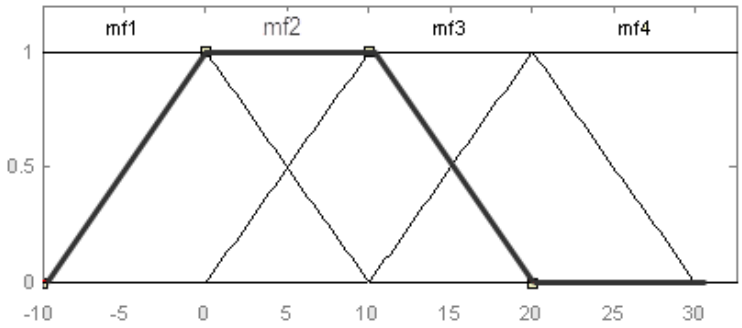


Figure 3-8 Memdani input membership function

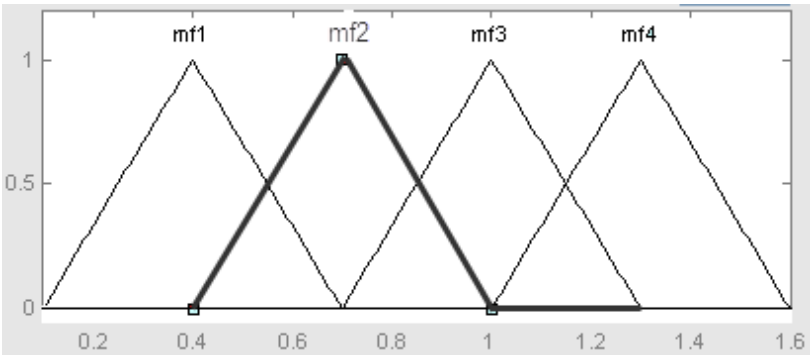


Figure 3-9 Mamdani output membership Function

There are two main approaches to elicit fuzzy rules [11]. These include:

1. The expert knowledge is translated into if-then rules. A structured model can be used to incorporate these rules. Membership functions and weights of rules can be calibrated using input and output data.
2. No prior knowledge about the system is initially used. A fuzzy model is constructed based on a certain algorithm. Fuzzy rules and membership functions are expected to describe the system behavior. An expert can modify the rules and the membership functions.

In this work, the first approach is used.

There are four fuzzy rules in the proposed approach. These include:

1- If prod_sum is less than 0, then productivity factor = 0.4.

2- If prod_sum is between 0 and 10, then productivity factor = 0.7.

3- If prod_sum is between 10 and 20, then productivity factor = 1.

4- If prod_sum is greater than 20, then productivity factor = 1.3.

The centroid method is used for Defuzzification which calculates the center of gravity of a surface.

After applying the fuzzy logic approach, the productivity factor has a specific value for each value of prod_sum. Table (3-6) shows some samples of the new values of the productivity factors using Mamdani fuzzy system and Table (3-7) shows the values of the values of the productivity factors using Sugeno fuzzy system. The labels IN, PO and PN correspond to prod_sum, old productivity factor and new productivity factor respectively.

Our experiments show that there is no noticeable difference between Mamdani and Sugeno systems so we compared the MLP model with the regression model which is based on Mamdani system and thus, the Sugeno system was ignored.

As seen in Table (3-6), the values of the new productivity factor (PN) are not as crisp as the values of the old productivity factor (PO). This leads to better estimation results. For instance, a complete list of the productivity factor values can be obtained using the proposed fuzzy logic inference system.

3.2.3 Neural Network Model

Neural network models have been widely used in software estimation as alternative solutions to regression models. In this chapter, a neural network model is developed based on a set of 120 projects, of which 100 projects were used in the training stage and 20 projects were used in the testing stage. In Section 3.3, a comparison is conducted between the proposed neural network model and the proposed regression model with fuzzy logic.

Each neural network model has input and output layers. If data are not linearly separable, which is the case in our problem, a hidden layer should exist between the input and output layers. The proposed neural network is classified as Multilayer Perceptron (MLP) that contains an input layer, one hidden layer and an output layer. The main inputs to the proposed neural network model are software size and team productivity represented by the eight environmental factors (E1 to E8 as shown in Table 2-6). The output of the model is software effort. The main reason of choosing the eight environmental factors to

represent the team productivity rather than choosing the *prod_sum* variable is to see the impact of each of these eight factors on software effort. The structure of the proposed neural network is depicted in Figure (3-10).

Table 3-6 New productivity factor using mamdani system

IN	PO	PN	IN	PO	PN
-10	0.4	0.4	8	0.7	0.78
-9	0.4	0.44	9	0.7	0.81
-8	0.4	0.47	10	0.7	0.85
-7	0.4	0.493	11	1	0.88
-6	0.4	0.511	12	1	0.91
0	0.4	0.55	20	1	1.15
1	0.7	0.583	21	1.3	1.15

Table 3-7 New productivity factor using sugeno system

IN	PO	PN	IN	PO	PN
-10	0.4	0.4	8	0.7	0.8
-9	0.4	0.42	9	0.7	0.83
-8	0.4	0.45	10	0.7	0.86
-7	0.4	0.46	11	1	0.89
-6	0.4	0.48	12	1	0.9
0	0.4	0.55	20	1	1.15
1	0.7	0.58	21	1.3	1.17

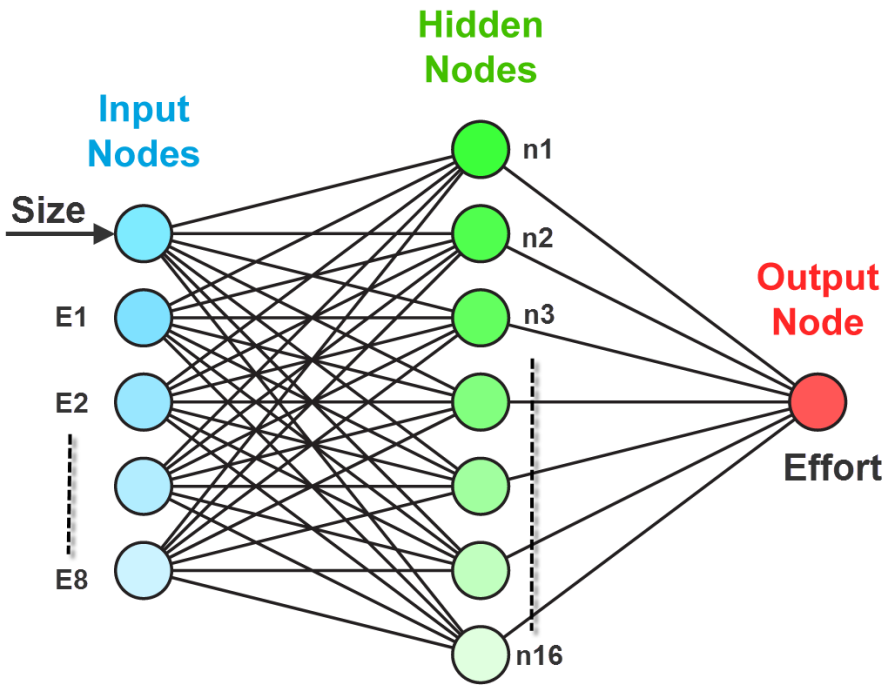


Figure 3-10 Neural network model

To generate the proposed neural network model, several steps must be considered. The first step is to determine the number of nodes in the hidden layer. This problem is highly controversial and there is no straightforward answer to it. If the number of hidden nodes is too few, there will be high training error and high generalization error due to underfitting. On the other hand, if the number of hidden nodes is too high, you may get low training error but still have high generalization error due to overfitting. Overfitting occurs when the model gives good results in training but bad results in the validation process. Blum [12] and Linoff et al. [13] argued that the number of nodes in the hidden layer should be between the number of nodes in the input layer and double that number. In other words, if the number of the nodes in the input layer is ni, the number of nodes in

the hidden layer should be between (ni+1) and 2ni. In our case, the number of hidden nodes falls between 10 and 18. Another consideration should be taken while developing the neural network model is how to train, validate and test the model. Here, the term "validation" is used during the training stage. The purpose of the validation is to see how the model is performing in the training phase. On the other hand, the term "testing" is used when data which were not included in the training stage are used to test and assess the model (this is discussed in Section 3.3). Taking these considerations into account is very critical and crucial since the model is deemed definitive when the training process has finished. After that, the model is used to predict software effort. The model is trained, validated and tested using a set of 120 available projects (100 projects for training and 20 projects for testing, see Appendix D). The size, environmental factors (E1 to E8) and the actual effort of each project are known. For better results, these projects should be shuffled before the training process. For this reason, a k-fold cross-validation technique is used. The value of "k" chosen is 10. This means that the training data is divided into 10 equal sets. The training process is repeated 10 times. In each time, 9 sets are used for training and 1 set to validate the training. The validation error of each round is computed as the average error of the projects within a set. After the training process has finished, all the sets will have been used in the training and validation processes. The round with minimal average error is selected. Keep in mind that testing data should be selected before applying the cross-validation method because testing data should not be included in the training or validation processes. The algorithm used to train the model was

Levenberg-Marquardt backpropagation. To demystify the process of training the neural network model, the following algorithm is used:

1- Prepare the data projects to be used in training, validation and testing processes.
2- Randomly pick 20 projects to be used for testing after the training/validation process has finished.
3- Randomly divide the remaining data (100 projects) into 10 equal sets (S1 to S10).
4- Set the number of nodes in the hidden layer to 10 ("nh" =10).
5- Set the number of training rounds (i) to 1 ("i" =1)
6- In Round "i" ("i" is a number between 1 and 10), use 9 sets for training and 1 set for validation (for each value of "i", 9 different sets are used for training and the remaining set for validation)
7- Record the validation error Vi-nh ("i" represents the number of the round, and "nh" the number of the nodes in the hidden layer. For instance, the first validation error is V1-10).
8- Increment the value of "i" by 1.
9- If the value of "i" is 11, then increment the value of "nh" by 1 and set the value of "i" to 1.
10- If the value of "nh" = 19, then stop training and exit.
11- Go to step "6"

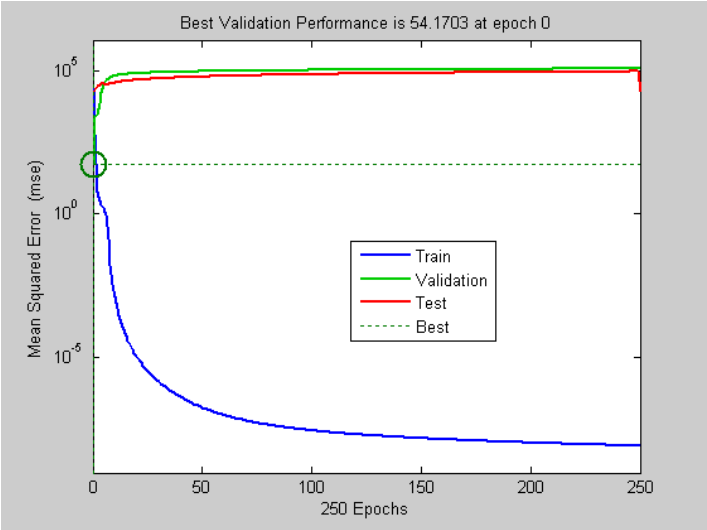


Figure 3-11 Performance graph

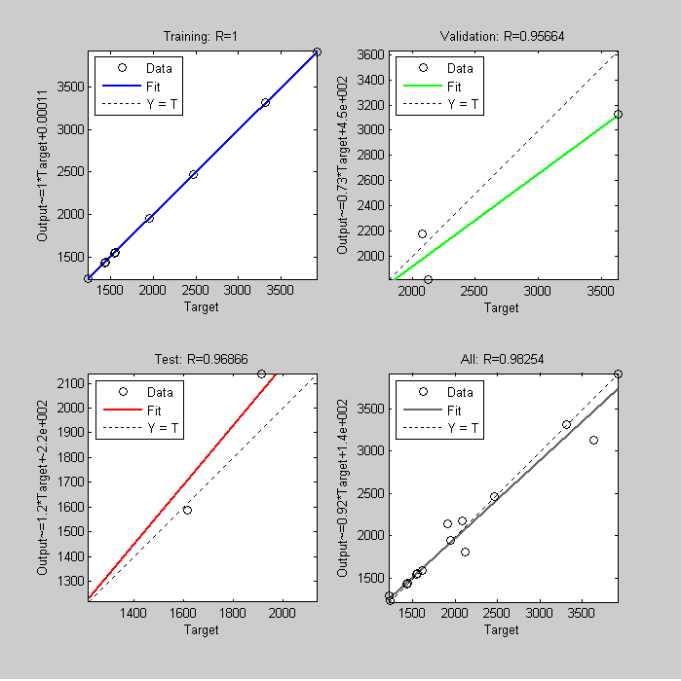


Figure 3-12 Regression graph

Ten rounds of training and validation were performed for each value of the number of hidden nodes “nh”. The values of “nh” were chosen between 10 and 18. The value 10 represents the number of the input nodes plus 1. The value 18 represents the number of hidden nodes multiplied by 2. This means that 90 values of Vi-nh were reported (from V1-10 until V10-18). Experiments showed that the minimal value of validation error occurred when the number of hidden nodes is 16.

To evaluate the proposed neural network model, performance and regression graphs were conducted after training as shown in Figures (3-11) and (3-12) respectively. There is no sign of overfitting in Figure (3-11).

3.3 Models Assessment and Discussion

This section presents the assessment of the proposed models. The set of projects that were selected before training the neural network model is used for testing. Moreover a comparison is performed between the proposed models and other models such as the original UCP model and Schneider’s model. Furthermore, a discussion is provided about the assessment of models.

3.3.1 Testing the Proposed Models

First, the set of testing projects that was excluded from the projects used to train the neural network model is divided into two main subsets. The first subset contains projects that are relatively small (< 100 UCP). The other subset contains projects that are relatively large (> 100 UCP). Three main experiments were conducted to test the

proposed models. First, the proposed models are tested using the whole set. Secondly, the proposed models are tested using the subset that contains the small projects. Thirdly, the subset that contains the large projects is used. The main purpose of conducting three experiments is to see how the neural network model performs with small and large projects. The evaluation criteria used for testing are MMER as well as PRED (25), PRED (35), PRED (50) and PRED (75). Table (3-8) shows the results when the whole set of testing is used. The columns Kar, Sch, Reg and Neu correspond to Karner's model (original UCP model), Schneider's model, the proposed regression model with fuzzy logic (Equation 3.7, the value of the productivity factor is depicted in Table 3-7) and the neural network model respectively. Figure (3-13) shows the Interval plot at 95% confidence level of MMER against Karner, Schneider, Regression and neural network models.

Table 3-8 Results using whole dataset

Critera	**Kar**	**Sch**	**Reg**	**Neu**
MMER	29.6	25.2	21.7	32.2
PRED	70	80	75	65
PRED	70	80	90	65
PRED	90	100	95	75
PRED	100	100	100	100

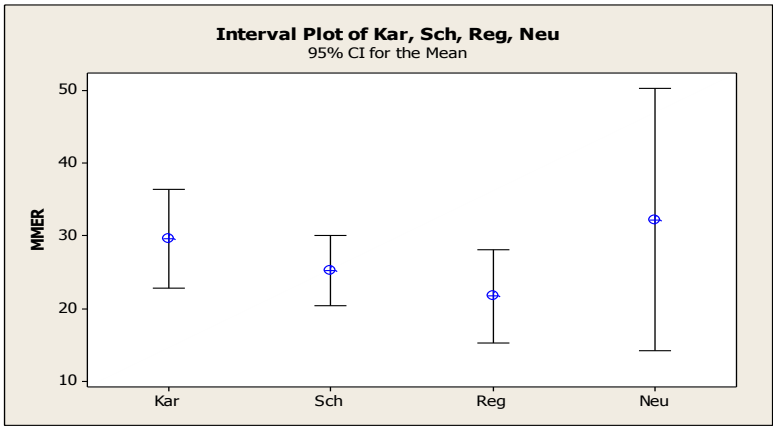


Figure 3-13 MMER interval plot

The second experiment is conducted by using the subset of data projects that contains small projects. Similarly, Table (3-9) and Figure (3-14) show the results.

Table 3-9 Results using small projects

Critera	**Kar**	**Sch**	**Reg**	**Neu**
MMER	31.25	22.4	26.52	21.3
PRED	70	90	50	100
PRED	70	90	80	100
PRED	80	100	90	100
PRED	100	100	100	100

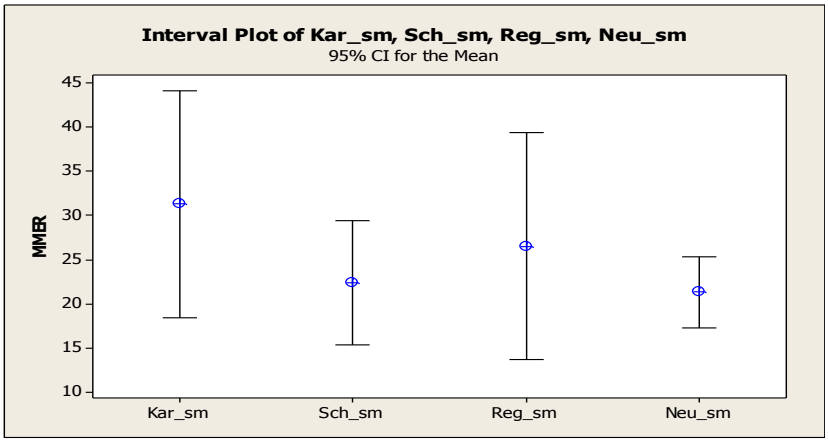


Figure 3-14 MMER interval plot for small projects

Consequently, Table (3-10) and Figure (3-15) show the results when large projects are used.

Table 3-10 Results using large projects

Critera	**Kar**	**Sch**	**Reg**	**Neu**
MMER	28	27.9	16.9	43.1
PRED	70	70	100	30
PRED	70	70	100	30
PRED	100	100	100	50
PRED	100	100	100	100

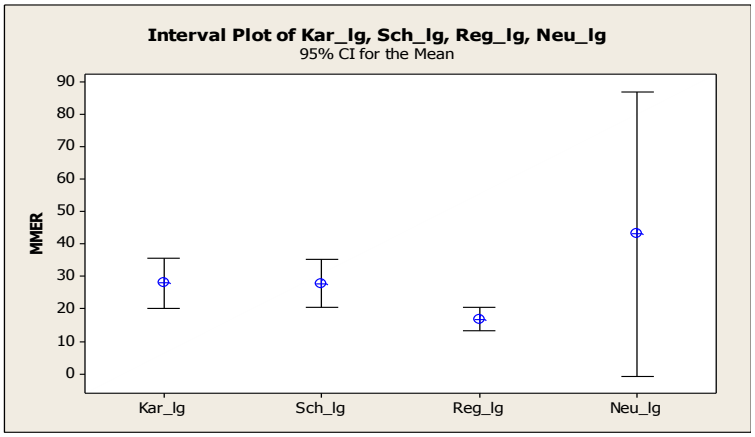


Figure 3-15 MMER interval plot for large projects

3.3.2 Comparison Among Different Models

Table (3-8) shows that the proposed regression model surpassed the original UCP model and Schneider's model by about 8% and 3.5% respectively when MMER is used. The regression model also gave good results when PRED (x) is used. However, the original UCP model and Schneider's model slightly surpassed the neural network model. Moreover, Figure (3-13) shows that the neural network model has the largest variation in the MMER which is not good. On the other hand, the neural network model gave promising results when small projects are used for testing as it surpasses all the models when MMER and PRED (x) are used. Furthermore, Figure (3-14) shows that the neural network model has the least variation in the MMER. Lastly, when large projects are used for testing, the regression model surpassed all the models when MMER and PRED (x) evaluation criteria are used. On the other hand, the neural network model did not perform well with large projects. As a conclusion, we noticed that the linear regression and the

MLP models can be used for software effort estimation and this answers the seventh research question proposed in Section 1.2.

3.3.3 Discussion

This chapter proposed a novel regression model to calculate software effort based on the use case diagrams. The regression model takes into consideration the non-linear relationship between software size and effort as well as the influence of team productivity. A Multilayer Perceptron (MLP) neural network model was also proposed in this work. A comparison between these two models shows that the neural network model can be used as an alternative to the proposed regression model. It is obvious from Table (3-10) that the regression model excels when large projects are used for testing. This might be because the regression model addresses the non-linear relationship between software size and effort as opposed to Karner's and Schneider's models. The non-linear relationship shows that when software size increases, software effort will increase exponentially.

3.4 Threats to Validity

Threats to validity can be summarized as follows:

- The regression model represented in Equation (3.7) was created using educational and industrial projects. Unfortunately, the majority of these projects are considered as small projects in the industry's point of view (less than 340 UCP). As seen in Figure (3-5), the proposed model shows the

influence of the non-linear relationship when software size increases. Nevertheless, the regression model has not been tested for projects whose efforts are larger than 8,000 person-hours.

- One of the reasons that the neural network model did not perform well with large projects is because the lack of the industrial projects. This model was trained using 100 projects. For this model to give better results, more projects should be used for training.
- It was difficult to elicit the environmental factors (Table 2-6) from the team that is developing software projects. For instance, developers might be optimistic when answering questions about their experiences and motivations. Moreover, the motivation of a developer/programmer might differ when placed in a different team, even in the same project. Furthermore, there is no straightforward rule to calculate the productivity of the team based on the productivity of each team member. In this work, the average of all team members was performed to calculate the productivity of the team.
- The UCP model mainly depends on the use case diagrams. If the use case diagrams were not properly designed, a huge error could be incurred.
- Because of the lack of obtaining industrial projects, some educational projects were used. Educational projects are mainly developed by students who work with these projects as part time. Projects developed by inexperienced students might incur errors when the actual software effort is estimated. Moreover, experiments show that most students only focus on the programming part

when developing software projects and thus, ignore the other stages of the software life cycle.

3.5 Conclusion

This chapter focused on software effort estimation from the use case diagrams using the use case point (UCP) model. In the UCP model, the unadjusted software size (UUCP) is calculated based on the number and complexity of the use cases as well as the actors. The adjusted use case point size (UCP) is then calculated by multiplying the UUCP by the technical and environmental factors. The technical factors represent the project complexity whereas the environmental factors represent the team productivity. After the UCP size is calculated, software effort can be estimated by multiplying the UCP by 20. There are two main shortcomings in the original UCP model. The first one is that the UCP model considers the relationship between software size and effort is linear. This is incorrect because when software size increases, the number of team members required to develop this software increases. When the team becomes larger, communication overhead will incur and this requires additional effort. This concludes that when software size increases, software effort will increase exponentially. Another shortcoming is that the influence of the team productivity is not taken into consideration while estimating effort. In this work, a novel simple regression model is proposed to tackle these limitations. A multiple linear regression model was developed to predict the productivity factor proposed in the simple linear regression. A Mamdani fuzzy logic approach was used to adjust the values of the productivity factor.

Another contribution in this work was to develop a Multi Layer Perceptron (MLP) neural network model. This model takes the software size and the team productivity represented by eight factors as inputs. The output of this model is the software effort. The proposed regression and neural network models were tested and evaluated. A comparison among the regression model, the neural network model, the original UCP model and the Schneider's model was conducted in three experiments. In the first experiment, all available data set was used to test and assess the models. In the second test, larger projects (>2,000 person-hours) were used for testing, while in the third experiment, smaller projects (<2,000 person-hours) were used for testing. Results show that the proposed regression model surpasses all the models when the first and the second experiments were used. On the other hand, the neural network model gives better results than the other models in the third experiment. This had led to the conclusion that an MLP neural network can be used as an alternative to regression models for projects of effort less than 2,000 person-hours.

The next step in this investigation will focus on improving the regression and the neural network models when new projects are available. The environmental and the technical factors of the UCP model should be updated. Moreover, the UCP model should be reconstructed to handle use cases of more than 7 transactions. Furthermore, the weights of the use cases should be calibrated.

References

[1] D. Longstreet, "Estimating Software Effort," *Software Metrics,* 2008.

[2] S. McConnell, *Software Estimation: Demystifying the Black Art.* Redmond, Washington: Microsoft, 2006.

[3] D. D. Galorath and M. W. Evans, *Software Sizing, Estimation, and Risk Management.* Boston, MA, USA: Auerbach Publications, 2006.

[4] A. C. Cameron and P. K. Trivedi, *Regression Analysis of Count Data.* Cambridge, UK: Cambridge University Press, 1998.

[5] W. Humphrey, *A Discipline for Software Engineering.* Addison Wesley, 1995.

[6] E. L. Lehmann, *Nonparametrics: Statistical Methods Based on Ranks.* Prentice Hall, 1998.

[7] A. Edwards, *An Introduction to Linear Regression and Correlation.* W. H. Freeman and Comp., 1976.

[8] G. Schneider and J. P. Winters, *Applied use Cases, Second Edition, A Practical Guide.* Addison-Wesley, 2001.

[9] E. H. Mamdani, "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis," *IEEE Transactions on Computers,* vol. C-26, pp. 1182-1191, 1977.

[10] M. Sugeno and T. Yasukawa, "A fuzzy-logic-based approach to qualitative modeling," *IEEE Transactions on Fuzzy Systems,* vol. 1, pp. 7-31, 1993.

[11] Z. Xu and T. M. Khoshgoftaar, "Identification of fuzzy models of software cost estimation," *Fuzzy Sets and Systems,* vol. 145, pp. 141-163, 2004.

[12] A. Blum, *Neural Networks in C++: An Object-Oriented Framework for Building Connectionist Systems.* NY: Wiley, 1992.

[13] G. S. Linoff and M. J. Berry, *Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management.* NY: Wiley, 2011.

Chapter 4

4. Regression, RBFNN and GRNN[3]

4.1 Introduction

This chapter presents four sub-models of our main model. These models include non-linear regression, linear regression with a logarithmic transformation, Radial Basis Function Neural Network and General Regression Neural Network. Moreover, the inputs of the model are introduced. They include software size, team productivity, project complexity and requirements stability. The main difference between this chapter and Chapter 3 is that in Chapter 3 we developed a neural network model of type Multilayer Perceptron (MLP) as well as linear regression model with a logarithmic transformation. The inputs of the MLP model were software size and team productivity. Software size was calculated based on the use case point (UCP) method; however, team productivity was calculated based on eight factors, also known as environmental factors as shown in

[3] Part of this chapter has been submitted to journal of Systems and Software (Elsevier) and an extended version was submitted to Empirical Software Engineering (Springer).

1. **Ali Bou Nassif**, Luiz Fernando Capretz, Danny Ho and Daniel Varona, "Software Effort Estimation from Use Case Diagrams Using Non-Linear Regression Analysis", Journal of Systems and Software, 2012 (Under review).
2. **Ali Bou Nassif**, Luiz Fernando Capretz and Danny Ho, "Regression and Neural Network Models for Software Effort Estimation from Use Case Diagrams", Empirical Software Engineering, 2012 (Under review).

Table (2-6). Furthermore, the project complexity factor was ignored in the previous chapter. This is because the project complexity was represented by the technical factors through the adjusted use case point size.

In this chapter, we propose a new method to calculate software size from use cases to overcome the limitations of the UCP model [1]. Team productivity was calculated based on five factors illustrated in Section 4.2.3 instead of the eight environmental factors Table (2-6) proposed in the use case point model. This is because Ochodek et al. [2] argued that the number of environmental factors can be reduced without deteriorating the estimation accuracy. Moreover, in this chapter, we introduce project complexity as a factor that affects software effort in Section 4.2.2. Most importantly, in Chapter 3, the requirements stability factor was one of the eight factors that contribute to productivity; however, we found that requirements stability is an essential factor and thus, we introduce it in Section 4.2.4 as one of four factors that affect software effort estimation. Section 4.2.5 shows the effort-size relationship. Sections 4.3 and 4.4 present the non-linear and linear regression models, respectively. Sections 4.5 and 4.6 present a Radial Basis Function Neural Network (RBFNN) and General Regression Neural Network (GRNN) models, respectively. In Section 4.7, we show how software effort can be estimated based on the regression and neural network models. Sections 4.8 and 4.9 present the verification and evaluation of models, respectively. Section 4.10 presents threats to validity and Section 4.11 concludes the chapter.

4.2 Model's Input Factors and Effort-Size Relationship

This section presents the four inputs of our proposed model. These include software size, project complexity, productivity and requirements stability. Moreover, the calibration of the productivity factor is introduced. Furthermore, we depict the actual relationship between software effort and size based on the industrial data points.

4.2.1 Size Estimation

In this work, a new approach to predict size estimation from use case descriptions is proposed to tackle the limitations of the use case model is proposed. In the use case point model [1] the total number of transactions in the use case description (scenario) is calculated as the number of transactions of the Success scenario plus the number of transactions in the Extension scenario. In our work, we investigated the weight of the transactions in the Success scenario versus the weight of the transactions in the Extensions Scenario. We have noticed through comparing industrial projects that if the Success and the Extensions have the same number of transactions, the effort required to develop the Success scenario is more than the effort required to develop the Extensions scenario. To support this claim, we have run three experiments. In each of the three experiments, a multiple linear regression model is developed that has four independent variables (productivity, complexity, requirements uncertainty and size) and one dependent variable (effort). The main difference among these three experiments is the why the size is estimated. In the first experiment, the complexity of a use case is determined by adding the number of transactions in the Success scenario with the number of transactions in the Extensions. In the second experiment, the use case complexity is determined by adding the number of transactions in the Success scenario with half the

number of transactions in the Extensions scenario. In the third experiment, the use case complexity is determined by adding the number of transactions in the Success scenario with third the number of transactions in the Extensions scenario. In each experiment, the coefficient of determination R^2 was calculated. We have noticed that the value of R^2 of the second experiment is the highest. Based on this, the use case complexity is determined by adding the number of transactions in the Success scenario with half the number of transaction in the Extensions scenario. Size estimation is based on the following rules:

- Consider all types of use cases in the use case diagram.
- In the use case scenario of each use case, count the number of transactions (based on the definition of transactions in the use case point model [1]) in the Main Success Scenario. This is noted by T_S.
- In the use case scenario of each use case, count the number of transactions in the Extensions part. This is noted by T_E.
- The total number of transactions of the use case is calculated as $T_S + T_E/2$.
- Assign a weight for each use case based on the rules proposed in Table (4-1).
- The total size of the project is conducted by adding the complexity weight of each use case. In other words,

$$Size = \sum_{i=1}^{6} n_i \times w_i. \qquad (4.1)$$

Where n is the number of use cases of variety *i* and w is its corresponding weight. For instance, the open-bracket representation "]4,8]" for LO indicates that "4" is not included. For instance, Table (4-1) answers the first research question proposed in Section 1.2.

Table 4-1 Use case complexity

Complexity level	Number of transactions	Complexity weight
VL (Very low)	[1,4]	5
LO (Low)	]4,8]	10
NM (Normal)	]8,12]	15
HI (High)	]12 to 16]	20
VH (Very High)	]16 to 20]	25
XH (Extra High)	> 20	30

4.2.2 Project Complexity

The complexity of the project is an important factor in software effort prediction. Complexity can be interpreted as an item having two or more elements [3] [4]. There are two dimensions of complexity. These include business scope such as schedule, cost, risk and technical aspect which is the degree of difficulty in building the product [4]. Technical complexity deals with the number of components of the product, number of technologies involved, number of interfaces and types of interfaces [4]. The project complexity can be classified as low complexity, medium complexity or high complexity [4]. Project complexity should be distinguished from other project characteristics such as size and uncertainty [3]. Complex projects require more effort to develop than simple

projects that have the same size as shown in Equation (4.2). The general equation of software effort can be represented as [5]:

$$Effort = \frac{Complexity}{Productivity} \times Size. \quad (4.2)$$

In our research, we identify the project complexity based on five levels (from Level-1 to Level-5). The reason behind defining five levels is to be compatible with other cost estimation models such as COCOMO where cost drivers are classified into five or six levels (such as Very Low, Low, Nominal, High, Extra High). Additionally, this classification is compatible to the project complexity classification in [4]. Regarding the complexity weights, we followed the UCP model where the highest level of complexity increases the effort by 30%. Moreover, as stated in the UCP model, normal complexity will not increase nor decrease the effort (factor = 1). The five complexity levels are defined as follows:

- Level-1: The complexity of a project is classified as Level-1 if the project team is familiar with this type of project and the team has developed similar projects in the past. The number and type of interfaces are simple. The project will be installed in normal conditions where high security or safety factors are not required. Moreover, Level-1 projects are those of which around 20% of their design or implementation parts are reusable (came from old similar projects). The weight of the Level-1 complexity is 0.7.

- Level-2: This is similar to level-1 category with a difference that only about 10% of these projects are reusable. The weight of the Level-2 complexity is 0.85.
- Level-3: This is the normal complexity level where projects are not said to be simple, nor complex. In this level, the technology, interface and installation conditions are normal. Furthermore, no parts of the projects had been previously designed or implemented. The weight of the Level-3 complexity is 1.
- Level-4: In this level, the project is required to be installed on a complicated topology/architecture such as distributed systems. Moreover, in this level, the number of variables and interface is large. The weight of the Level-4 complexity is 1.15.
- Level-5: This is similar to Level-4 but with additional constraints such as a special type of security or high safety factors. The weight of the Level-5 complexity is 1.3.

4.2.3 Productivity

Productivity is inversely proportional to effort as seen in Equation (4.2). The higher the productivity of a team is, the less effort required to develop a project. Team productivity was calculated based on five factors illustrated in Section 4.2.3 instead of the eight environmental factors Table (2-6) proposed in the use case point model. This is because Ochodek et al. [2] argued that the number of environmental factors can be reduced without deteriorating the estimation accuracy. Also, in the use case point model, Requirements Stability factor is one of the eighth environmental factors (Table 2-6).

According to NASA lab [6] and COCOMO II model [7], requirements uncertainty can increase software effort up to 40%. This is the reason of removing the requirements stability factor when calculating productivity and assigning it as an independent factor as shown in Section 4.2.4. Each factor is rated from "1" which represents "very low" to "5" which represents "very high" and this is analogous to the classification of the cost drivers in COCOMO model. Factors with average classifications are rated as "3". These factors and their corresponding weights are:

- Team experience regarding the problem domain. Weight is 2.
- Team motivation. Weight is 1.
- Programming language type and experience. Weight is 2.
- Object oriented experience (UML). Weight is 2.
- Analytical skills. Weight is 1.

Regarding the first factor, if the project team is acquainted with the problem domain of the project, the effort required to develop the project will be less than the one if the team is inexperienced with the problem domain. The motivation of the team also contributes to the productivity. People within the same team who get along with each other can achieve work faster. Team motivation is also influenced by several factors such as the environment where the project is deployed, working pace, and the number of working hours per day or per week. For instance, full-time employees tend to be more productive than part-time employees. Another important productivity factor is the team experience and the type of programming language used to implement the project. In general,

programmers who are expert in a certain language are those who have at least 5 years of experience. Moreover, the productivity would be higher when using 4th generation languages (4GL) such as Visual Basic and Matlab rather than using 3GL such as C++. The team experience in the object oriented concept is very important because the team is either drawing UML diagrams or implementing UML diagrams. This research is based on predicting software effort from UML use case diagrams. The final factor which contributes to the productivity is the analytical skills of the team. This is the team's ability to articulate, understand and solve both complicated and uncomplicated problems.

The second step after assigning a rate (from 1 to 5) to each of the above productivity factors, is to determine the value of the productivity. The productivity factor is calculated in two steps. First, calculate productivity_sum as follows:

$$\Pr oductivity\_Sum = \sum_{i=1}^{5} F_i * W_i. \qquad (4.3)$$

Where F is the productivity factor of variety i and W is its corresponding weight. Based on the rules introduced above, the minimum value of Productivity_Sum is when the rate of all factors is "1". Similarly, the maximum value would be when the rate of all factors is "5". This means that Productivity_Sum falls between 8 and 40. If all productivity factors are average (rate=3), then Productivity_Sum is 24. The second step is to find the final Productivity value which is based on the value of Productivity_Sum as shown in Table (4-2).

Table 4-2 Productivity factor

Productivity_Sum	Productivity
Less than or equal 14	0.7
Between 15 and 20	0.85
Between 21 and 27	1
Between 28 and 34	1.15
Greater than or equal 35	1.3

4.2.3.1 Calibration of Productivity Factor

Table (4.2) presents the values of the productivity factor. As seen in the table, these values are crisp and there is no graduation between each level. To avoid this problem, we use fuzzy logic to adjust the productivity values. The fuzzy system type used is Mamdani [8], the input membership of the fuzzy logic system used is Trapezoidal where the output membership is Triangular. Trapezoidal input membership was used because the input value (Productivity_Sum) is a range between two numbers; however, triangular output membership was used because the output (Productivity) has a single value. The method used in the Defuzzification stage is the centroid which is the default method used. Matlab version 2010b was used to conduct the experiments of the fuzzy logic approach. Figures (4.1) and (4.2) show the input and the output memberships respectively.

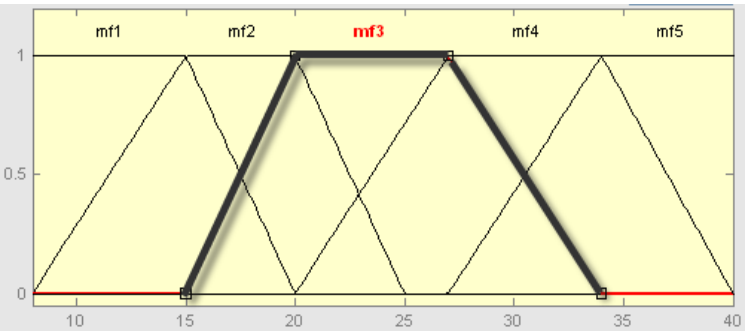


Figure 4-1 Mamdani input membership function

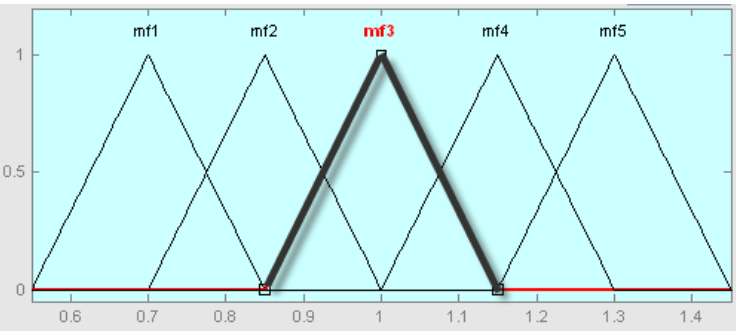


Figure 4-2 Mamdani output membership function

There are two main approaches to elicit fuzzy rules [9]. These include:

1. The expert knowledge is translated into if-then rules. A structured model can be used to incorporate these rules. Membership functions and weights of rules can be calibrated using input and output data.
2. No prior knowledge about the system is initially used. A fuzzy model is constructed based on a certain algorithm. Fuzzy rules and membership functions are expected to

describe the system behaviour. An expert can modify the rules and the membership functions.

In this work, the first approach is used.

There are five fuzzy rules in the proposed approach. These include:

1- If Productivity_Sum is less than or equal 14, then productivity factor = 0.7.

2- If Productivity_Sum is between 15 and 20, then productivity factor = 0.85.

3- If Productivity_Sum is between 21 and 27, then productivity factor = 1.

4- If Productivity_Sum is between 28 and 34, then productivity factor = 1.15.

5- If Productivity_Sum is greater than 34, then productivity factor = 1.3.

The centroid method is used for Defuzzification which calculates the center of gravity of a surface.

After applying the fuzzy logic approach, the productivity factor has a specific value for each value of Productivity_Sum. Table (4-3) shows the old values of the productivity factor as well as the adjusted values (after applying fuzzy logic). The labels P_S, O_F and N_F correspond to Productivity_Sum, old productivity factor and new productivity factor respectively.

As seen in Table (4-3), the values of the new productivity factor (N_F) are not as crisp as the values of the old productivity factor (O_F). This leads to better estimation values.

Table 4-3 New productivity factor

P_S	O_F	N_F	P_S	O_F	N_F	P_S	O_F	N_F	P_S	O_F	N_F
8	0.7	0.7	**16**	0.85	0.806	**24**	1	1.03	**32**	1.15	1.18
9	0.7	0.727	**17**	0.85	0.835	**25**	1	1.07	**33**	1.15	1.2
10	0.7	0.745	**18**	0.85	0.865	**26**	1	1.07	**34**	1.15	1.23
11	0.7	0.758	**19**	0.85	0.894	**27**	1	1.07	**35**	1.3	1.23
12	0.7	0.766	**20**	0.85	0.925	**28**	1.15	1.1	**36**	1.3	1.23
13	0.7	0.771	**21**	1	0.949	**29**	1.15	1.12	**37**	1.3	1.24
14	0.7	0.774	**22**	1	0.974	**30**	1.15	1.14	**38**	1.3	1.25
15	0.85	0.775	**23**	1	1	**31**	1.15	1.16	**39**	1.3	1.27

4.2.4 Requirements Stability

Another important factor when conducting software estimation is the degree of the requirements stability. In many projects, clients tend to change or increase the number of requirements and this will increase the effort. Figure (4-3) shows an example of 40% increase in the number of requirements over time (x-axis) which has led to 40% increase in software effort estimation (1.4x in y-axis). This approach has been used by leading organizations such as NASA's Software Engineering Laboratory [6]. COCOMO II uses a similar approach called Breakage (BRAK) to reflect the requirements volatility of the project [7].

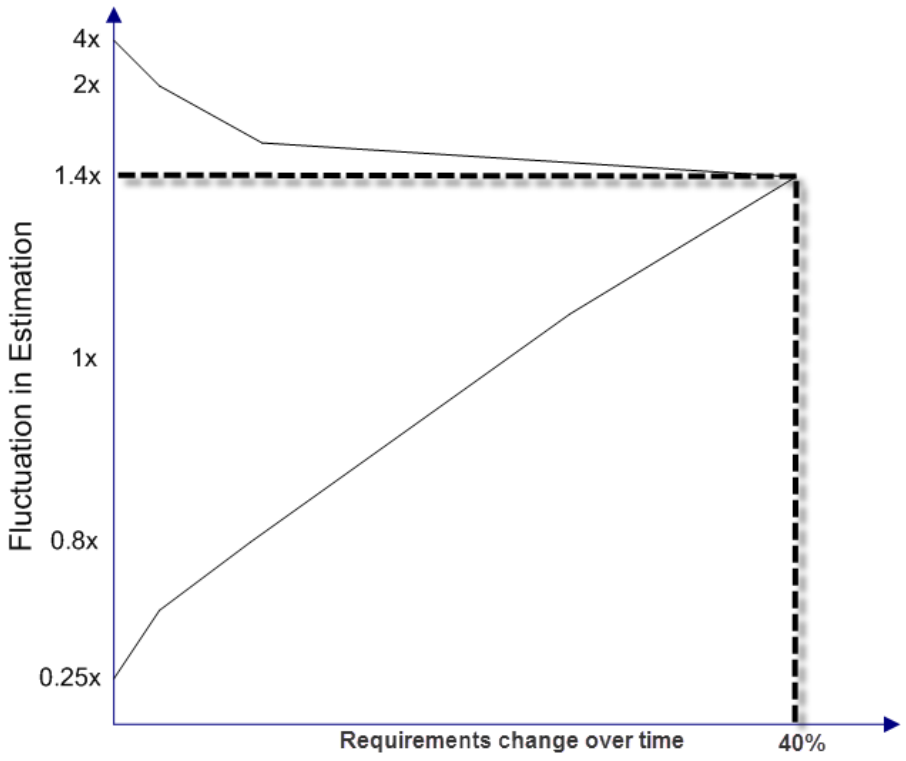


Figure 4-3 Requirements stability

We propose 5 levels of Requirements Stability from Level-1 (unstable requirements) to Level-5 (stable requirements). If the requirements are stable, there is no increase in software effort. Based on [6] and [7], 40% increase (or change) in the requirements can lead to 40% increase in effort. The weight for each level was defined as follows:

- Level-1: This indicates that there is an increase of 40% of the requirements during the project life cycle. This incorporates new requirements and changes in existing requirements. Weight is 1.4.

- Level-2: This indicates that there is an increase of 30% of the requirements during the project life cycle. Weight is 1.3.
- Level-3: This indicates that there is an increase of 20% of the requirements during the project life cycle. Weight is 1.2.
- Level-4: This indicates that there is an increase of 10% of the requirements during the project life cycle. Weight is 1.1.
- Level-5: This indicates that the requirements are stable during the project life cycle. Weight is 1.

4.2.5 Effort-Size Relationship

The original UCP model assumes that the relationship between software effort and size is linear as expressed in Equation (2.23). As discussed in Section 2.4.1.4, when the software size increases, software effort will increase but with non-linear relation. To support our hypothesis and to discover the type of this relationship (Effort – Size), among the 214 data projects that we have, 65 projects of software effort ranged between 122 person-hours and 129,353 person-hours were selected that have similar Complexity, Productivity and Requirements Stability (See Appendix E) . Figure (4-4) depicts the actual size and effort of these 65 projects as well as the original UCP Estimation. Figure (4-4) shows that the UCP method can be applied with acceptable error on small projects (size less than 250 UCP which is equivalent to 5,000 person-hours). Based on Figure (4-4), the UCP model cannot be applied on projects of effort more than 10,000 person-hours. Among the 214 data projects that we have, there are 58 projects (27%) that have effort more than

10,000 person-hours. This means that projects of greater than 10,000 person-hours cannot be ignored. The plot of the actual data projects in Figure (4-4) shows that the relationship between software effort and size in non-linear and this answers the sixth question proposed in Section 1.2.

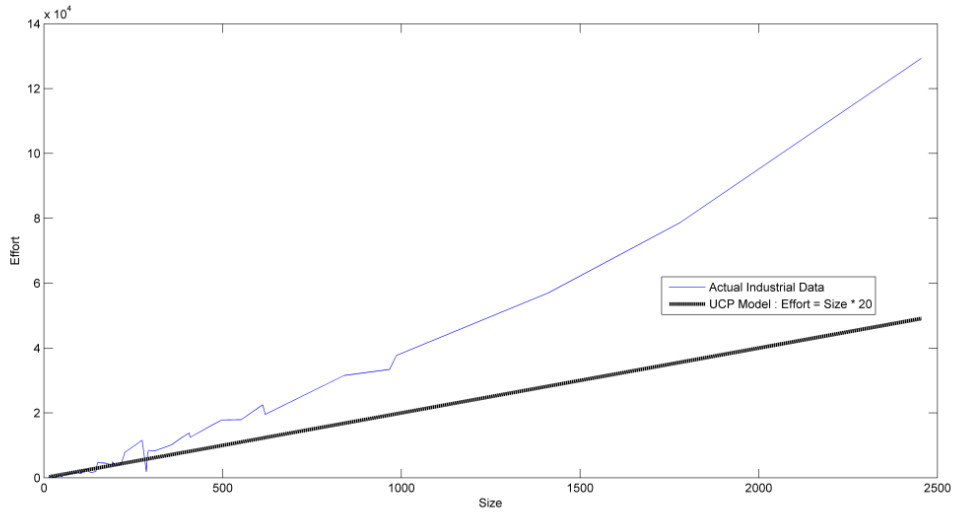


Figure 4-4 Comparison between UCP model and actual data

4.3 Non-linear Regression Model

In this section, we introduce the non-linear regression model that can best fit the non-linear relationship of the actual data shown in Figure (4-4). In statistics, regression analysis focuses on generating a relationship between a dependent variable (aka response) and one or more independent variables (aka predictors) [10]. Regression analysis studies show how the dependent variable responds to a change in the independent variables and it identifies which independent variable is related to the

dependent variable. Legendre [11] and Gauss [12] were among the first people who worked with regression models 200 years ago. There are many types of regression analysis. These include simple regression, multiple regression, linear regression and non-linear regression. Regression analysis has been widely used in software estimation. Software developers and project managers use historical data to build regression models. The regression models are then evaluated and compared with alternative models such as soft computing models.

In our previous publications [13] and [14] (Chapter 3), we proposed a linear regression model with a logarithmic transformation to predict software effort from use cases. In the work proposed in Chapter 3, we used the method used by the original UCP model to calculate software size. The factors calculating software effort were software size and team productivity. The model was evaluated using educational and industrial projects that are considered as relatively small projects. In this paper, a new approach to calculate software size is introduced. Moreover, we study and present factors that affect the prediction of software effort. These factors are the *Project Complexity, Team Productivity* and *Requirements Stability*. Most importantly, our model has been evaluated using industrial projects which are categorized from very small projects to very large projects (between 120 and 224,890 person-hours).

With respect to non-linear regression, many non-linear functions exist and it is not simple to just predict one. Based on the nature of the non-linear relationship in Figure (4-4), we used four different non-linear equations to see which equation can best fit the actual data. These equations include a second degree polynomial function and three exponential

functions as shown in Table (4-4), where the variable "x" corresponds to software size, the variable "y" corresponds to software effort, and "a", "b", "c" and "d" are constants.

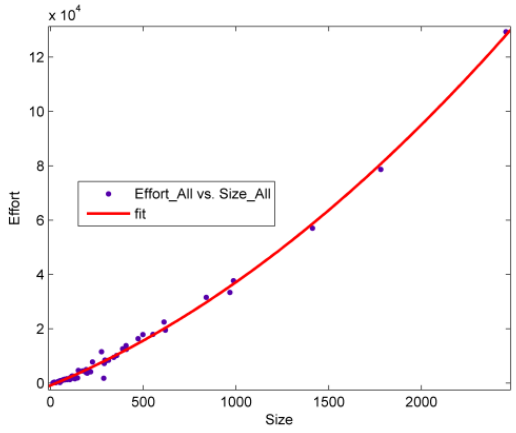
Table 4-4 Non-linear equations

Polynomial	Exponential 1	Exponential 2	Exponential 3
$y = a*x^2 + b*x + c$	$y = a + b*\exp(c*x)$	$y = a*\exp(b*x) + c*\exp(d*x)$	$y = \exp(a + b*x)$

In each non-linear equation type (Table 4-4), several experiments using Matlab 2010 were conducted using the whole dataset used in Figure (4-4) (65 projects) to calculate the values of the constants "a", "b", "c" and "d". In each experiment, the value of the coefficient of determination R^2 and the Root Mean Square (RMS) were measured. R^2 is the percentage of variation in Effort explained by the variable Size. An acceptable value of R^2 is ≥ 0.5 [15]. The RMS value shows how close the actual data are from the fitting curve. Both R^2 and RMS are important. Good regression models are those that have higher R^2 values and lower RMS values. Figures (4-5), (4-6), (4-7) and (4-8) show the regression graph, the value of the constants, R^2 and RMS of each function.

Based on the fitting graphs and on the R^2 and RMS values, the Polynomial (Figure 4-5) and the Exponential 2 (Figure 4-7) were candidates for the proposed regression model since they gave higher R^2 values and lower RMS values. However, after we have tested the Polynomial and Exponential 2 models, we found that they give inaccurate results when software size is less than 50 UCP. Although projects of size smaller than 50 UCP are considered small projects, small projects cannot be ignored. For this reason, the Polynomial and Exponential 2 models were eliminated, and consequently we found that none of these four non-linear models is appropriate to fit the whole project dataset.

Based on the above conclusion, the whole project dataset that is used to build the non-linear regression models (65 projects), was divided into three different ranges based on the software size. The first range is called Small, which includes 26 projects out of the 65 projects of software size less than 100 UCP (less than 2,000 person-hours). The second range is the Medium range that contains 21 projects of size ranged between 100 and 300 UCP (between 2,000 and 8,500 person-hours) and the third range is the Large one which contains 18 projects of size greater than 300 UCP (effort between 8,500 and 129,353 person-hours). Several experiments were performed to learn which of the four non-linear equations (Table 4-4) can best fit each range (Figures 4-9, 4-10, 4-11 and 4-12 for small dataset, Figures 4-13, 4-14, 4-15 and 4-16 for medium dataset and Figures 4-17, 4-18, 4-19 and 4-20 for large dataset). Experiments show that based on the fitting graphs, values of R^2 and RMS, the Polynomial model (Figure 4-9) can best fit the small dataset. However, the Exponential 3 (Figure 4-16) and Exponential 2 (Figure 4-19) models can best fit the Medium and the Large ranges, respectively.

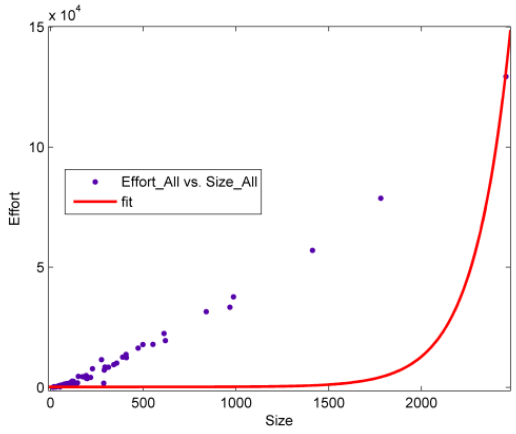


Polynomial

a=0.01
b=28.07
c=-870

R^2=0.99
RMS=1328

Figure 4-5 Polynomial, all data

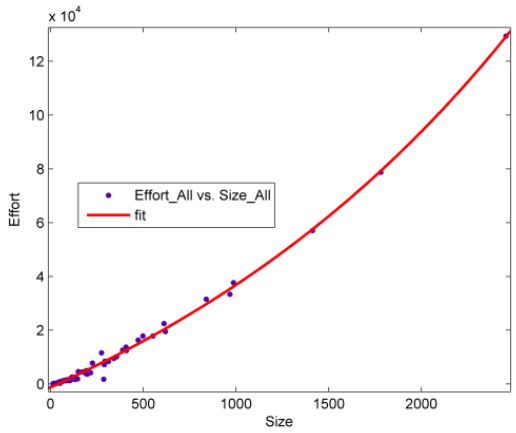


Exponential 1

a=225.3
b=0.43
c=0.005

R^2=0.40
RMS=15650

Figure 4-6 Exponential 1, all data

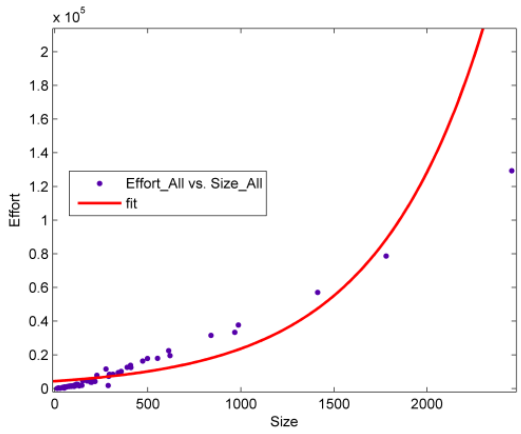


Exponential 2

a=54580
b=0.00046
c=-55770
d=-0.0001

R^2=0.99
RMS=1298

Figure 4-7 Exponential 2, all data

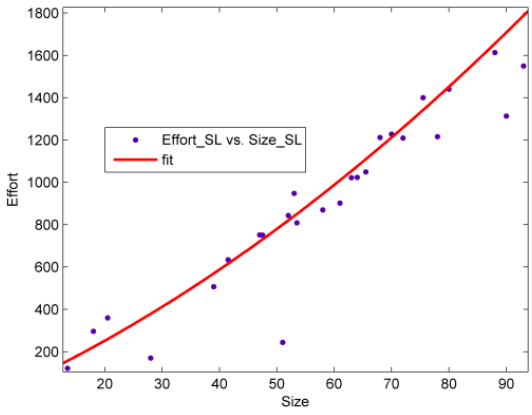


Exponential 3

a=8.37
b=0.0017

R^2=0.93
RMS=5349

Figure 4-8 Exponential 3, all data

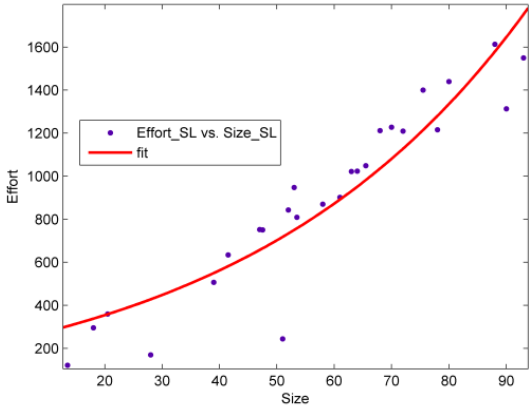


Polynomial

a=0.08
b= 12
c= -20

R^2=0.84
RMS= 167

Figure 4-9 Polynomial, small data

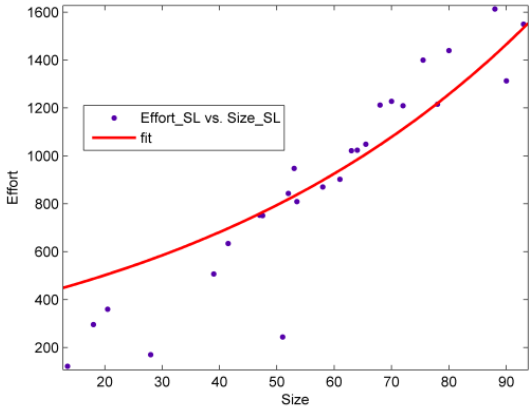


Exponential 1

a=-68
b=283.3
c=0.02

R^2=0.84
RMS=168.5

Figure 4-10 Exponential 1, small data

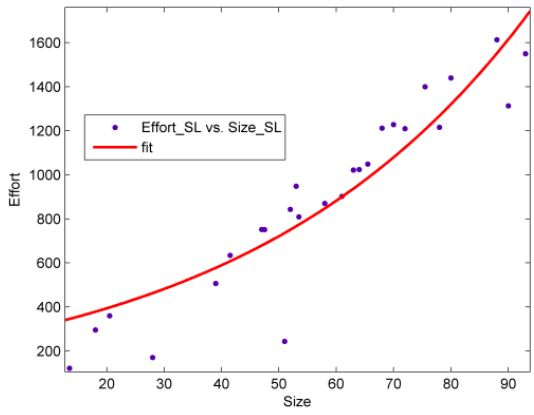


Exponential 2

a=0
b=0.015
c=369.5
d=0.015

R^2=0.77
RMS= 202

Figure 4-11 Exponential 2, small data

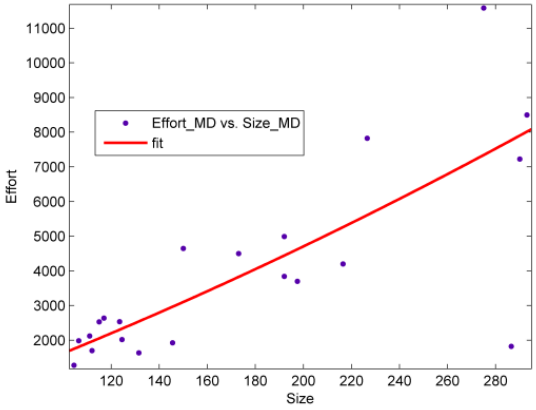


Exponential 3

a=5.57
b=0.02

R^2=0.84
RMS= 170

Figure 4-12 Exponential 3, small data

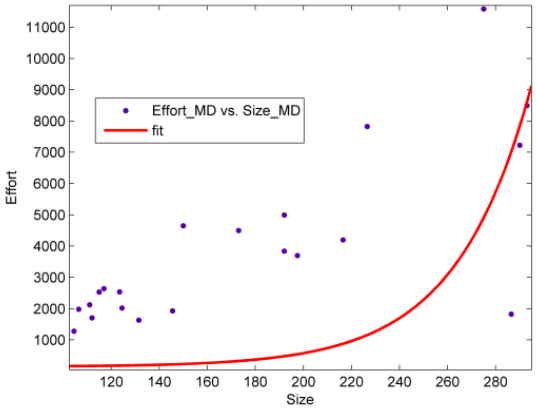


Polynomial

a=0.024
b= 23.5
c= -975

R^2=0.75
RMS= 1365

Figure 4-13 Polynomial, medium data

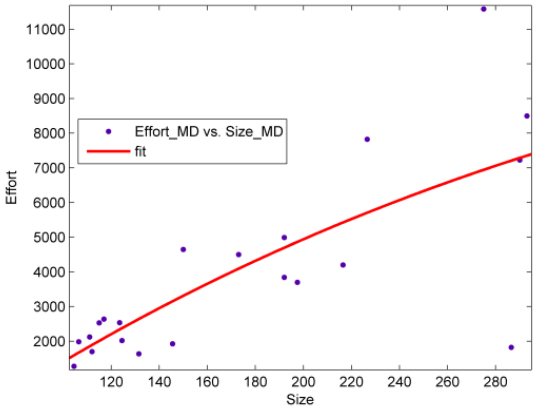


Exponential 1

a=140.1
b= 0.73
c= 0.031

R^2=-0.8
RMS= 3670

Figure 4-14 Exponential 1, medium data

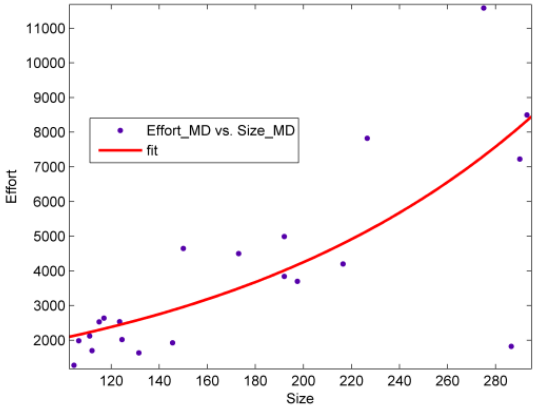


Exponential 2

a=-8.48e+10
b=-0.0014
c=8.48e+10
d=-0.0014

R^2=0.56
RMS=1963

Figure 4-15 Exponential 2, medium data

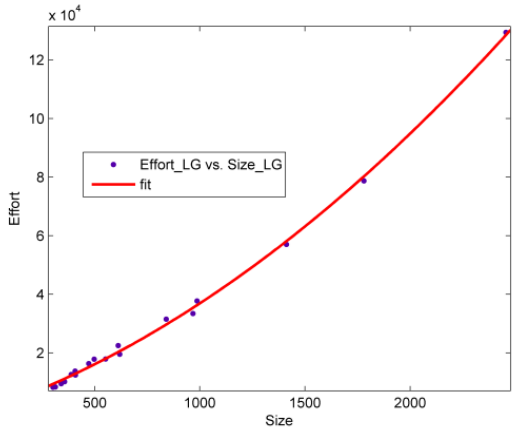


Exponential 3

a=6.9
b=0.0072

R^2=0.80
RMS=1225

Figure 4-16 Exponential 3, medium data

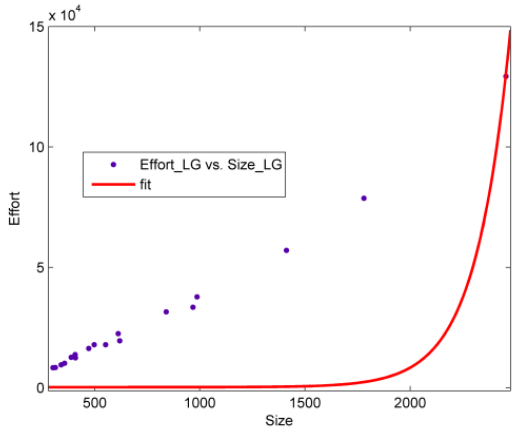


Polynomial

a=0.011
b= 25.19
c= 792.3

R^2=0.99
RMS=1439

Figure 4-17 Polynomial, large data

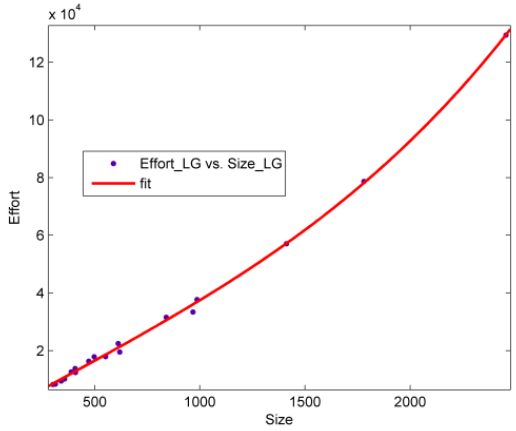


Exponential 1

a=225.3
b=0.041
c=0.0061

R^2=-0.04
RMS=31700

Figure 4-18 Exponential 1, large data

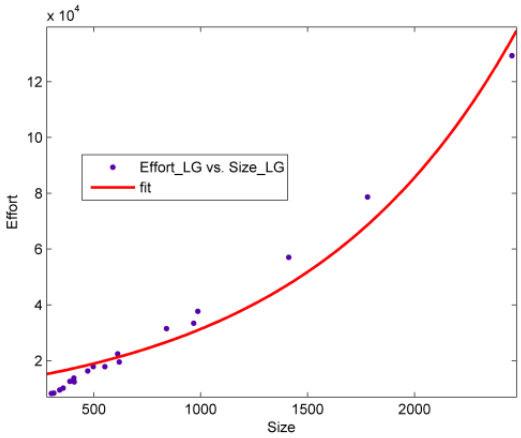


Exponential 2

a=25780
b=0.00067
c=-29570
d=-0.00083

R^2=0.99
RMS=1205

Figure 4-19 Exponential 2, large data



Exponential 3

a=9.34
b=0.001

R^2=0.96
RMS=5951

Figure 4-20 Exponential 3, large data

4.4 Linear Regression Model with a Logarithmic Transformation

In this section, we introduce the linear regression model. In linear regression, the best results are obtained if data are normally distributed [16]. Several experiments were conducted using Minitab version 16 to determine how data were distributed. For the purpose of consistency with the previous section (non-linear regression), the experiments were conducted on the whole project dataset, as well as the Small, Medium and Large ranges as defined in the previous section. Figures 4-21 to 4-28 show the histograms of software size and software effort, respectively when all dataset, small dataset, medium-sized dataset and large dataset are used respectively. Results show that data are not normally distributed. Generating regression models from data based on Figures 4-21 to 4-28 is possible but this will lead to poor results. For this reason, data were normalized using logarithmic transformation. After logarithmic transformation, data (ln size and ln effort) of all, small, medium, and large project dataset became normally distributed (Figures 4-29 to 4-36). If data were normally distributed, the regression equation would be:

$$y = a * x + b. \tag{4.4}$$

Where a and b are constants.

But since data were not normally distributed, linear regression is applied on ln(x) and ln(y) instead. The regression equation becomes as follows:

$$\ln(y) = c * \ln(x) + d. \quad (4.5)$$

Equation (4.5) can be written as follows:

$$y = A * x^{B}. \quad (4.6)$$

Where B=c and A=e^{d}.

Table 4-5 shows the values of the constants in Equations (4.5) and (4.6) as well as the values of R^2 and RMS in each of the four experiments (All Data, Small Data, Medium Data and Large Data).

Table 4-5 Linear model parameters

	Equation (4.5)					**Equation (4.6)**			
Category	**c**	**d**	**R^2**	**RMS**		**A**	**B**	**R^2**	**RMS**
All data	1.327	1.381	0.96	0.303		3.981	1.327	0.99	1506
Small Data	1.25	1.693	0.80	0.308		5.431	1.25	0.89	141.5
Medium	1.286	1.528	0.55	0.414		4.602	1.286	0.55	1832
Large Data	1.26	1.862	0.99	0.067		6.431	1.26	0.99	2497

The fitting graphs of Equation (4.5) are shown in Figures 4-37 (All Data), 4-38 (Small Data), 4-39 (Medium Data) and 4-40 (Large Data). However, the fitting graphs of Equation (4.6) are depicted in Figures 4-41 (All Data), 4-42 (Small Data), 4-43 (Medium Data) and 4-44 (Large Data). To better compare the non-linear equation with linear equation, Equation (4.6) is used for the linear regression. Despite the fact that the linear regression that represents the whole dataset (Figure 4-41) slightly surpasses the non-linear regression models that represent whole dataset, this model (Figure 4-41) does not perform well when the size of the input data points is too small or too large.

Consequently, this model is ignored and the other three models (Small, Medium and Large) are used instead. The comparison between the proposed models is presented in Section 4.9. Please note that Equations (4.5) and (4.6) as well as the non-linear equations (Section 4.3) only represent the non-linear relationship between software effort and size as shown in Figure 4-4 and not the final equations for predicting software effort. The final equation of software effort is represented in Section 4.7 (Equation 4.7).

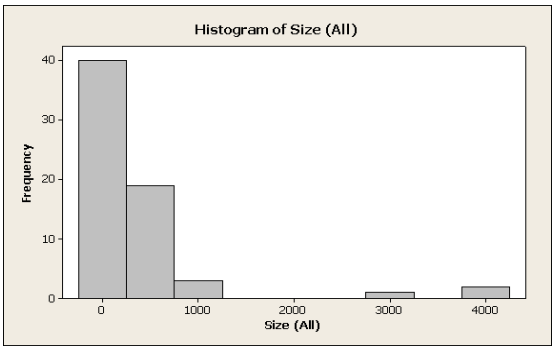


Figure 4-21 Size, all data

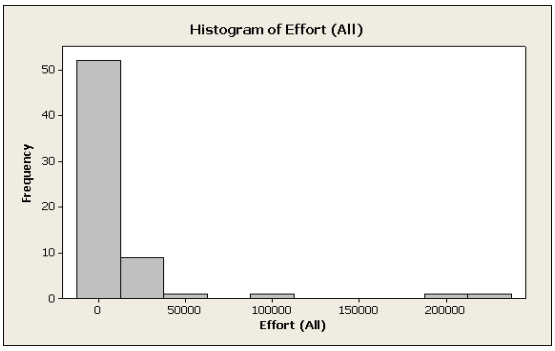


Figure 4-22 Effort, all data

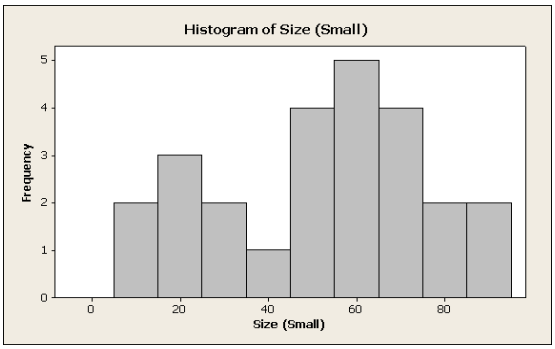


Figure 4-23 Size, small data

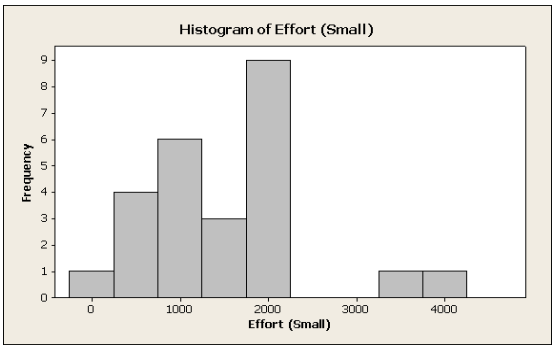


Figure 4-24 Effort, small data

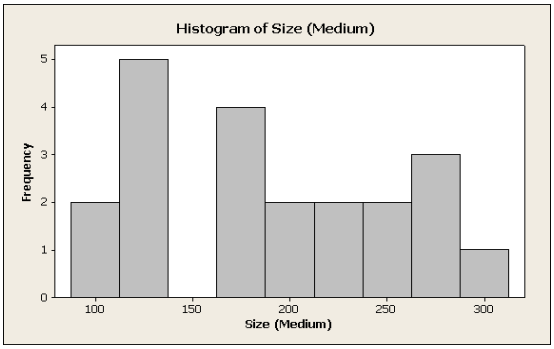


Figure 4-25 Size, medium data

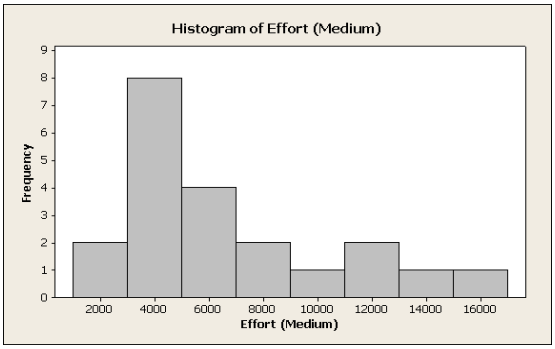


Figure 4-26 Effort, medium data

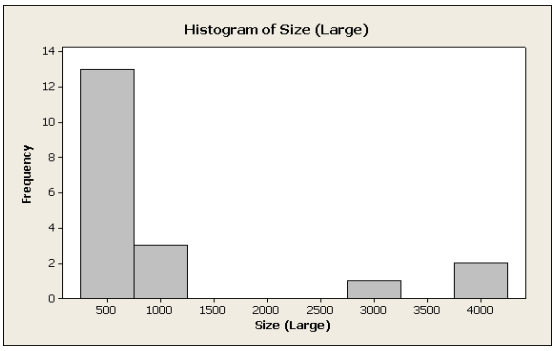


Figure 4-27 Size, large data

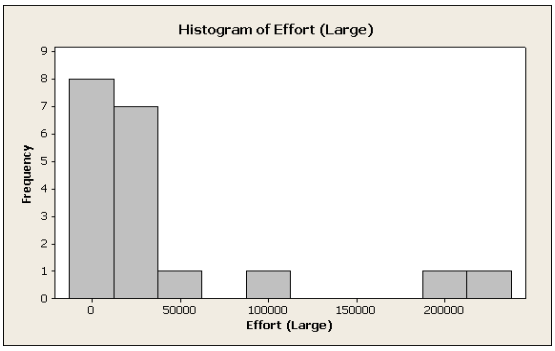


Figure 4-28 Effort, large data

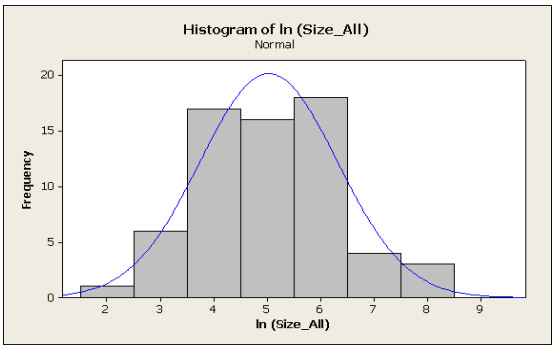


Figure 4-29 ln (Size_All_Data)

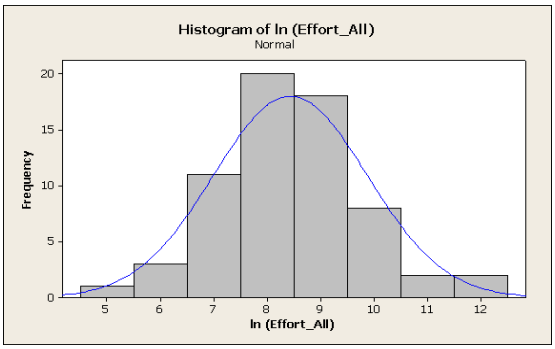


Figure 4-30 ln (Effort_All_Data)

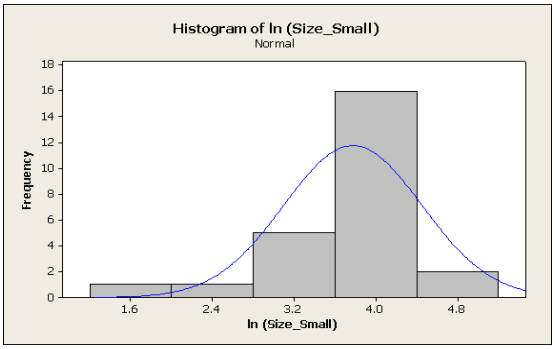


Figure 4-31 ln (Size_Small_Data)

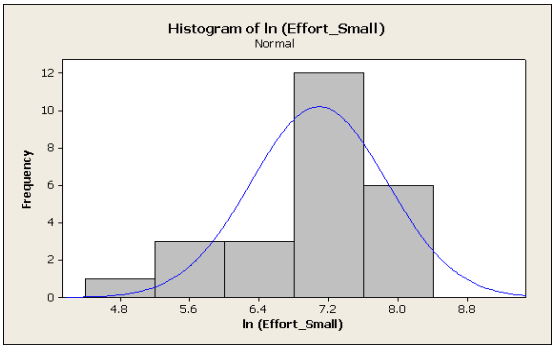


Figure 4-32 ln (Effort_Small_Data)

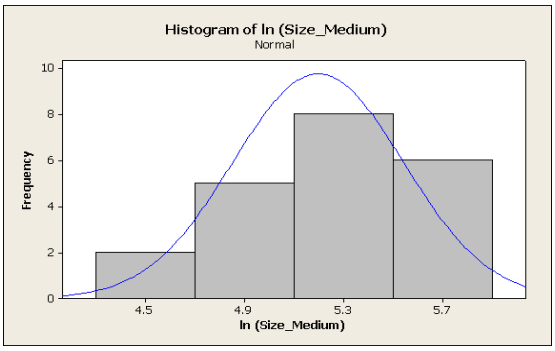


Figure 4-33 ln (Size_Medium_Data)

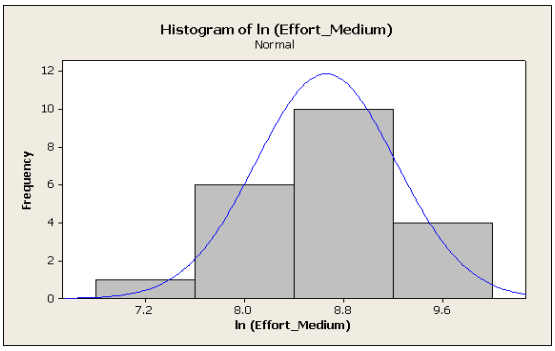


Figure 4-34 ln (Effort_Medium_Data)

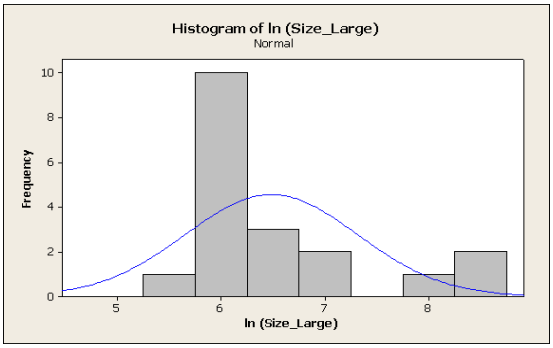


Figure 4-35 ln (Size_Large_Data)

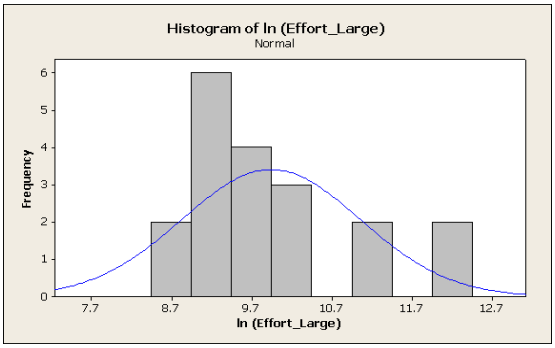


Figure 4-36 ln (Effort_Large_Data)

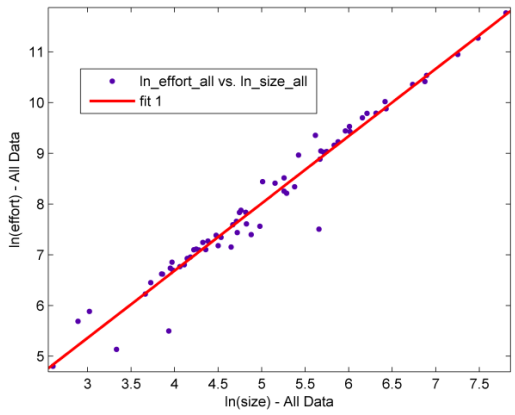


Figure 4-37 ln(size/effort), all data

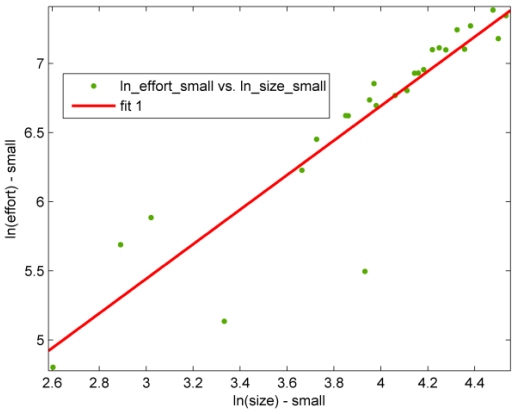


Figure 4-38 ln(size/effort), small data

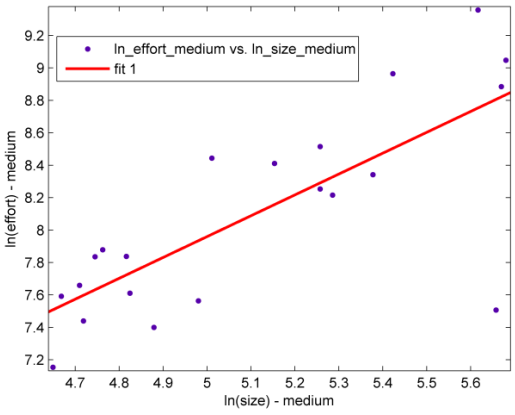


Figure 4-39 ln(size/effort), medium data

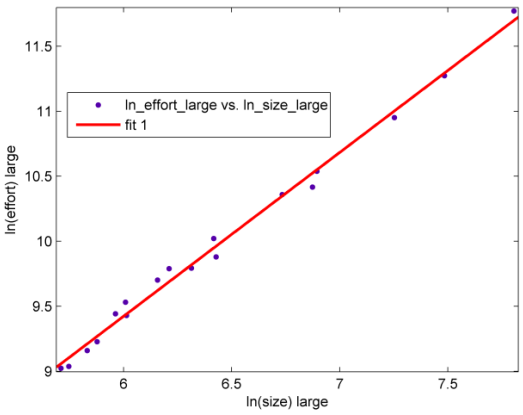


Figure 4-40 ln(size/effort), large data

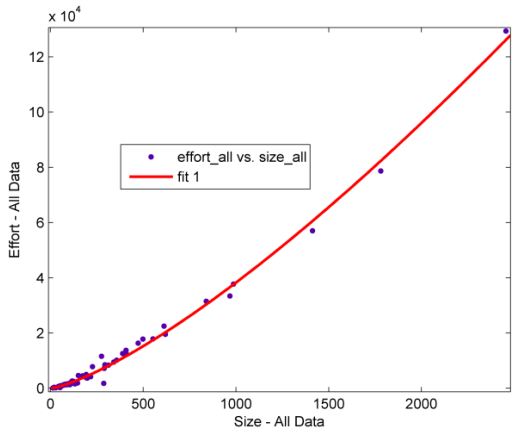


Figure 4-41 Size/effort, all data

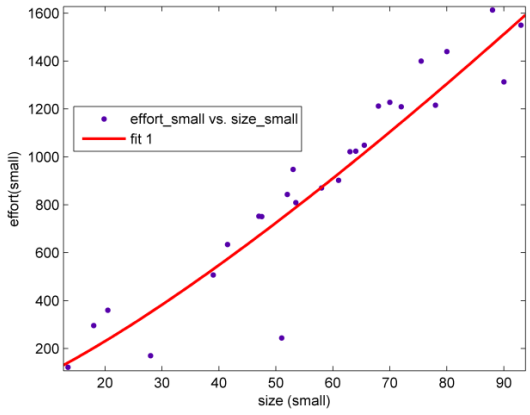


Figure 4-42 Size/effort, small data

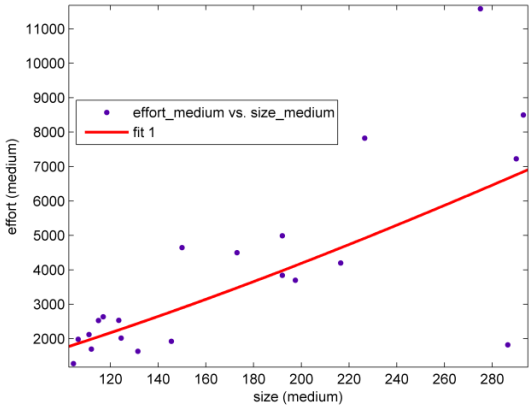


Figure 4-43 Size/effort, medium data

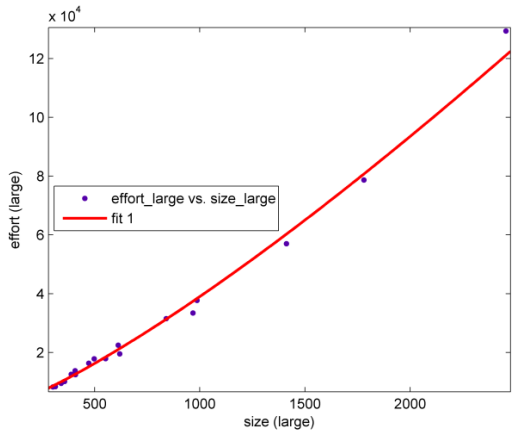


Figure 4-44 Size/effort, large data

4.5 Radial Basis Function Neural Network

The diagram of the Radial Basis Function Neural Network (RBFNN) model is presented in Figure 2-3. The input layer of our proposed model has four inputs which are software size, project complexity, team productivity and requirements stability. The training process performed in the neural network models is different from the regression models.

This is because in the regression models, the first step was to represent the non-linear relationship between software effort and size. The second step was to include the influence of the other three factors (project complexity, team productivity and requirements stability). Our proposed neural network models map non-linear relationships between the input and output of the model. For this reason, we trained our neural network models, the RBFNN and the GRNN (presented in Section 4.6) using actual projects by giving four inputs to the model and one output (software effort). Nonetheless, it is important that our models are trained based on different data sizes (small, medium and large). Among the whole project dataset (214 projects), there are 85 small projects (size less than 100 UCP), 69 medium-sized projects (size between 100 and 300 UCP) and 60 large projects (size greater than 300 UCP). In general, neural network models are trained using 70% of the whole data and tested (evaluated) using the remaining 30%. To guarantee that our model is trained and tested using dataset of different sizes, 70% of each range (small, medium and large) is used for training and the remaining 30% of each range are used for evaluating the models. So, the RBFNN and GRNN models are trained using 149 industrial projects that include 60 small projects, 48 medium-sized projects and 41 large projects. The software effort/size relationship of these 149 projects is depicted in Figure 4-45. The remaining data (25 small projects, 21 medium projects and 19 large projects) are used to evaluate not only the neural network models, but also the regression models. The purpose that the same data is used to evaluate the four models is to conduct a thorough and unbiased comparison among these four models as shown in Section 4.9.

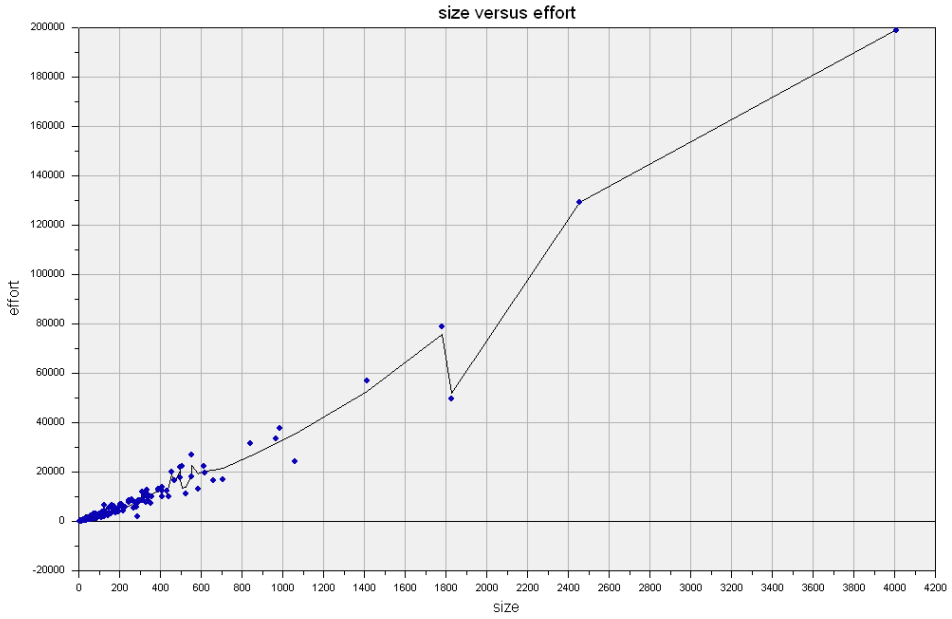


Figure 4-45 Size/ effort relationship

In general, RBFNN networks are trained using k-means clustering to find cluster centers that can be used as centers for the RBF functions. However, this method has been criticized because clustering results are not sensitive to initial conditions and ignore the influence of dependent variable [17], and thus this method does not provide the optimal centre for the RBF functions. In this work, the RBFNN model is trained using the algorithm proposed by Chen et al [18]. This algorithm uses the orthogonal forward selection method based on the leave one-out criterion. The number of the hidden neurons will start by one and is increased until the best training results are achieved. Best results are achieved when the average error in the training stage is minimal as well as the validation error. To avoid overfitting (when the error is very low during training but high in the validation), the k-fold (k=10) cross validation technique was used. The training

points were divided into 10 groups, such that 9 groups were used for training and 1 group for validation. The process was repeated 10 times so that all data points were used in training and validations. The average error of the 10 stages is reported. The training process stopped when the number of the hidden neurons reached 9 as shown in Figure 4-46. The proportion of variance explained by the model (R^2) in the training and validation processes is 0.99 and 0.51 respectively. The root mean squared errors (RMS) in the training and validation are 2,015 and 15,063 respectively. Figure 4-47 shows the relationship between the actual and predicted target values.

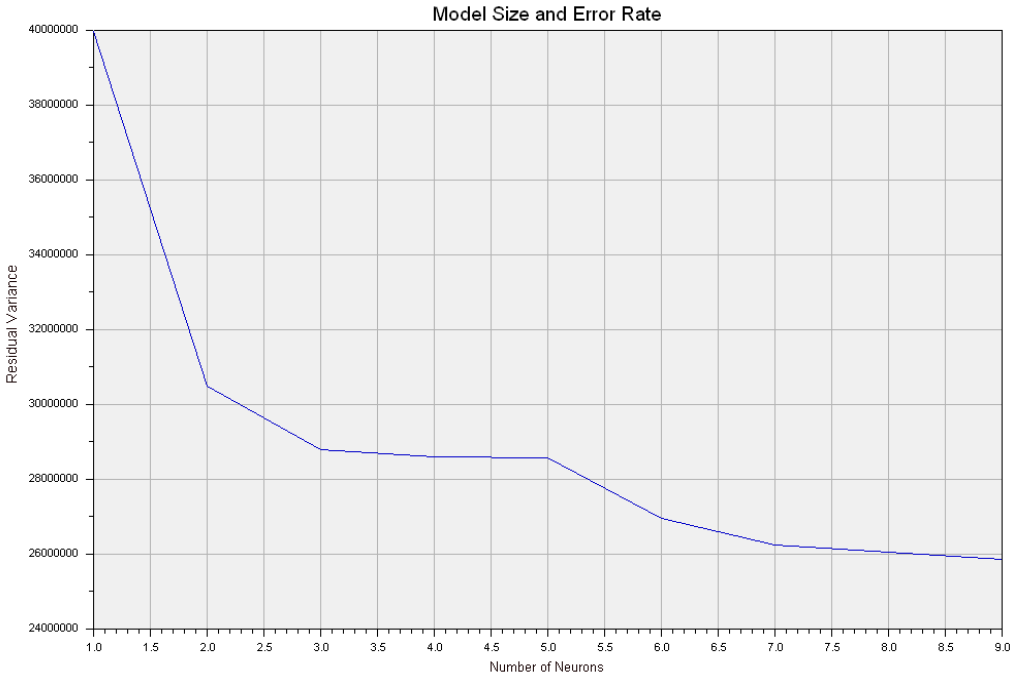


Figure 4-46 Number of neurons

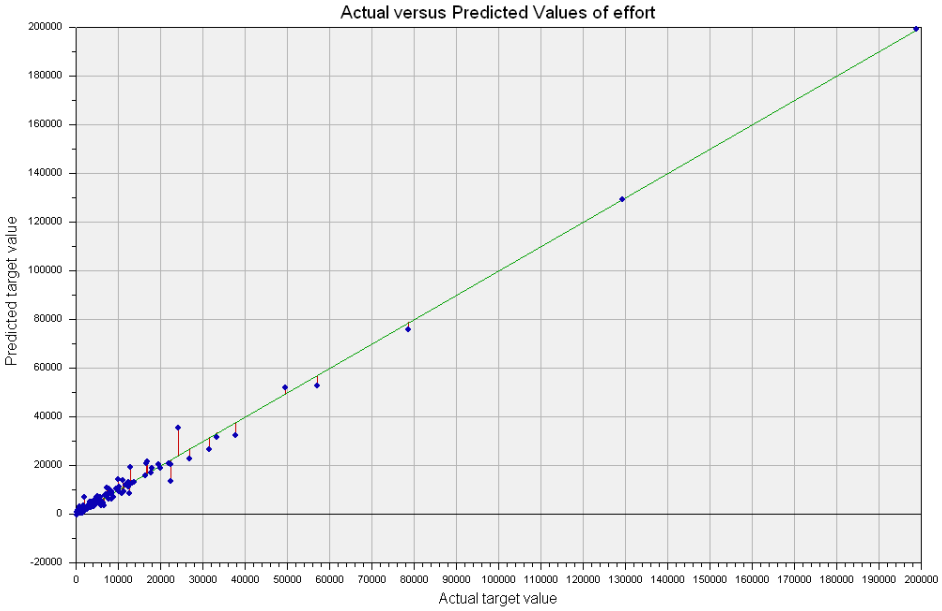


Figure 4-47 Actual versus predicted effort

The main parameters of the RBFNN model (center and spread of each neuron for each input variable) are shown in Table 4-6. The complete list of parameters is shown in Appendix G.

Table 4-6 RBFNN parameters

	Size		Project Complexity		Team		Requirements Stability	
Neuron	Center	Spread	Center	Spread	Center	Spread	Center	Spread
1	14.09	57.97	0.24	379.54	-3.46	268.98	0.66	193.23
2	-0.30	9.86	-0.07	0.11	-3.68	149.66	1.92	390.26
3	1.72	0.27	-0.25	141.66	0.46	344.43	0.26	31.22
4	0.20	329.25	-0.07	204.63	-0.85	300.51	0.58	0.06
5	14.56	144.28	-0.06	67.63	-1.61	151.21	0.53	0.15
6	1.22	1.01	0.12	355.93	0.95	333.81	1.76	296.17
7	-0.06	10.68	-0.20	380.31	-2.22	179.61	1.65	137.95
8	0.60	10.91	0.29	95.95	-3.22	167.00	0.20	21.36
9	6.37	234.47	0.08	199.07	0.42	192.66	1.66	1.77

4.6 General Regression Neural Network

This section presents the General Regression Neural Network (GRNN) model. The diagram of this model is shown in Figure 2-4. Like the RBFNN model, the GRNN model was trained using 149 different data points (60 small, 48 medium and 41 large). The most important parameter of the GRNN model is the spread value. Eleven experiments (Table 4-7) were conducted to learn the optimal value of the spread by taking different values of spread. If the spread value is very small, the training error will be small but the validation error will be high and this leads to overfitting. When the spread value increases, the training error increases where the validation error decreases to a point where both the training and validation errors start to increase until the training and the validation errors become equal. The k-fold (k=10) cross validation technique was used in the training stage. The optimal value of the spread value was selected based on the values of R^2 and RMS in each of the training and validation stages. The best results were obtained when the spread value was 0.81.

Figure (4-48) shows the relationship between the actual and predicted target values.

Table 4-7 GRNN spread value

	Training		Validation	
Spread	R^2	RMS	R^2	RMS
0.5	0.99	1530	0.53	14479
0.6	0.99	1761	0.57	13950
0.7	0.99	2041	0.57	13840
0.81	**0.99**	**2431**	**0.70**	**11705**
0.85	0.99	2578	0.67	12044
0.88	0.98	2661	0.68	12098
0.93	0.98	2850	0.64	12795
1.44	0.96	4377	0.68	12146
2.81	0.88	7446	0.75	10577
3.10	0.86	8096	0.51	13787
6.05	0.56	14180	0.56	14159

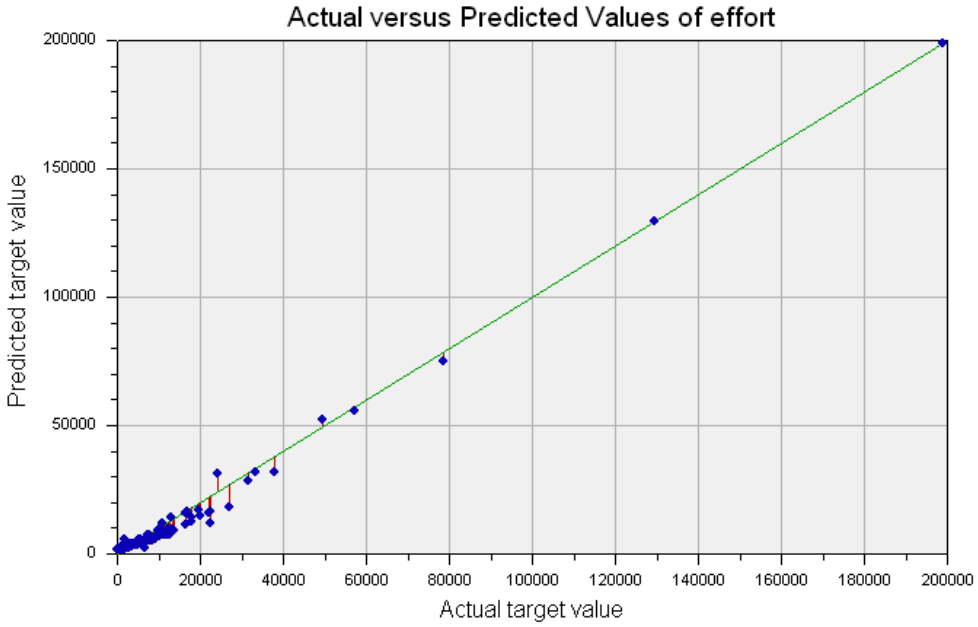


Figure 4-48 Actual versus predicted target (GRNN)

4.7 Software Estimation

This section presents the prediction of software estimation based on the above four models (non-linear regression, linear regression, RBFNN and GRNN).

4.7.1 Estimation using non-linear regression

Our novel model for software effort estimation is different from the one proposed in Equation (4.2), as our model incorporates the non-linear relationship between software size and effort and the requirements uncertainty, in addition to project complexity and team productivity. The general equation of our model is shown as follows:

$$Effort = \frac{C * R}{P} f(size). \tag{4.7}$$

Where "Effort" is measured in person-hours, "C" is the project complexity as introduced in Section 4.2.2, "R" is the degree of the requirements stability as introduced in Section 4.2.4 and "P" is the productivity as depicted in Table (4-3) in Section 4.2.3. Equation (4.7) shows that effort is proportional to project complexity and requirements instability and inversely proportional to productivity. For instance, Equation (4.7) answers the second, third and fourth research questions. The complexity "C" can increase the effort by 30%. The requirements uncertainty "R" can increase the effort by 40%. The productivity "P" can increase the effort by 42%. By taking the influence of each of the complexity, productivity and requirements uncertainty factors, we deduced that the three factors combined (non-functional requirements) can increase software effort by 160% and this answers the fifth research question proposed in Section 1.2.

f(size) is different for each range (Small, Medium and Large) and calculated as follows:

$$f(size\_small) = a * size^2 + b * size + c. \quad (4.8)$$

Where "size" is software size of values less than 100 UCP calculated based on the rules listed in Section 2.1. The constants "a", "b" and "c" have values of 0.08, 12 and -20, respectively. Similarly,

$$f(size\_medium) = \exp(a + b * size). \quad (4.9)$$

Where "size" is software size of values between 100 and 300 UCP calculated based on the rules listed in Section 4.2.1. The constants "a" and "b" have values of 6.9 and 0.0072 respectively. Similarly,

$$f(size\_large) = a * \exp(b * size) + c * \exp(d * size). \quad (4.10)$$

Where "size" is software size of values greater than 300 UCP calculated based on the rules listed in Section 4.2.1. The constants "a", "b", "c" and "d" have values of 25,780, 0.00067, -29,570 and -0.00083 respectively. Please note that there is a limitation for the maximum size that can be used. This is discussed in the Threats to Validity Section (Section 4.10).

4.7.2 Estimation using linear regression

Equation (4.7) will also be used to estimate software effort using the linear regression model. Similarly, *f(size)* is different for each range (Small, Medium and Large) and calculated as follows:

$$f(size) = A * (size)^B. \quad (4.11)$$

Where A=5.431 and B=1.25 for the small range (size less than 100 UCP), A=4.602 and B=1.286 for the medium range (size between 100 and 300 UCP), A=6.431 and B=1.26 for the large range (size greater than 300 UCP). Please note that there is a limitation for the maximum size that can be used. This is discussed in the Threats to Validity Section (Section 4.10).

4.7.3 Estimation using RBFNN and GRNN

Each of the neural network models is trained and designed to take 4 input vectors which are software size, project complexity, team productivity and requirements stability. The output is the predicted target (software effort) measured in person-hours.

4.8 Models verification

In this section, we will verify the regression and the neural network models by injecting random data points to the input and measuring the output. For regression models, 10 data points are chosen such that 5 data points are of size ranging between 90 and 110 UCP. However, the other 5 data points are of size between 280 and 320 UCP. The reason behind choosing these data points is that size of the first 5 points is critical and can be used as input to the model of the small range (less than 100 UCP) or with the model of the medium range (between 100 and 300 UCP). Similarly, the second 5 data points fall between the medium and the large ranges. Regarding the neural network models, 16 data points of size ranging between 50 and 800 UCP (incremented by 50) are used to verify the RBFNN and the GRNN models. The other three inputs (product complexity, team productivity and requirements stability) are considered normal (value = 1). The main goal

of verifying the neural network models is to check the output of the models when the input slightly increases. Please note that the verification part is different from the evaluation part (Section 4.9) where the regression and the neural models are tested using the same industrial data points.

4.8.1 Non-Linear Model Verification

Equations (4.8), (4.9) and (4.10) represent three non-linear regressions used for small, medium, and large software size, respectively. However, before we can generalize these equations, we have to make sure that there is no abrupt change in results when Equations (4.8) and (4.9) are used at the same time with software size around 100 UCP. The same assumption is valid when applying Equations (4.9) and (4.10) on projects of size around 300 UCP. In other words, Equation (4.8) should be used on projects of sizes that belong to the interval [1,100] and Equation (4.9) should be used on the interval [100,300]. But since the interval [90,110] falls between the intervals [1,100] and [100,300], the results obtained from applying Equations (4.8) and (4.9) when software sizes belonging to the interval [90,110] should be close. This assumption should be correct if we try to use Equations (4.9) and (4.10) on projects of software size that fall in the interval [280,320]. To verify the above hypothesis, Equations (4.8) and (4.9) are applied on five software size values which are 90, 95, 100, 105 and 110. Similarly, Equations (4.9) and (4.10) are applied on five values which are 280, 290, 300, 310 and 320. Table 4-8 shows the results with the mean error and the 95% confidence interval. These results show that the mean error and the 95% confidence interval are relatively small and thus, we conclude that

there is a smooth transition from Equation (4.8) to Equation (4.9) and from Equation (4.9) to Equation (4.10).

Table 4-8 Non-linear regression verification

Software Size (UCP)	Equation (4.8)	Equation (4.9)		Equation (4.10)
90	1708	1916	N/A	N/A
95	1842	1987	N/A	N/A
100	1980	2060	N/A	N/A
105	2122	2136	N/A	N/A
110	2268	2215	N/A	N/A
280	N/A	N/A	7583	7668
290	N/A	N/A	8152	8070
300	N/A	N/A	8764	8473
310	N/A	N/A	9422	8876
320	N/A	N/A	10129	9278
Mean Error	-78.8		337	
Margin Error (95%	128		460	
Confidence Interval	-78.8 ± 128		337 ± 460	

4.8.2 Linear Model Verification

The same process used to verify the non-linear model is used to verify the linear one. Table 4-9 shows the results with the mean error and the 95% confidence interval. Results show that the transition from the Small to the Medium ranges is very smooth and is better than the transition in the non-linear regression. However, the transition between the Medium and Large ranges is smoother in the non-linear regression. Most importantly, models verification does not show the accuracy of models. The accuracy (evaluation) is presented in Section 4.9.

Table 4-9 Linear regression verification

Software Size (UCP)	Equation (4.11)small	Equation (4.11) medium		Equation (4.11)large
90	1505	1500	N/A	N/A
95	1610	1608	N/A	N/A
100	1717	1717	N/A	N/A
105	1825	1828	N/A	N/A
110	1934	1941	N/A	N/A
280	N/A	N/A	6456	7792
290	N/A	N/A	6754	8145
300	N/A	N/A	7055	8500
310	N/A	N/A	7359	8859
320	N/A	N/A	7666	9220
Mean Error	-0.6		-1445	
Margin Error (95% CI)	5.7		106.7	
Confidence Interval	-0.6 ± 5.7		-1445 ± 106.7	

4.8.3 Neural Network models verification

To verify the RBFNN and GRNN models, 16 data points are used that have average values in complexity, productivity and requirements stability (values = 1). The size of these data points varies between 50 and 800 UCP incremented by 50. The main goal of this verification is to measure the output (predicted effort) when software size varies from small project size to large project size. There are two main hypotheses in the verification of the neural network models. First, software effort is proportional to software size. An increase in software size should lead to an increase to software effort. The second hypothesis is that a slight change in software size should not lead to a big change in software effort. Table 4-10 shows the results of RBFNN and GRNN models.

Table 4-10 Neural network models verification

	RBFNN		GRNN	
Size	Effort	Ratio	Effort	Ratio
50	383.95	7.68	1053.54	21.07
100	1521.26	15.21	2400.63	24
150	2933.26	19.56	2926.72	19.51
200	4611.92	23.06	3652.51	18.26
250	6516.15	26.06	4623.15	18.49
300	8574.92	28.58	5857.78	19.52
350	10711.75	30.61	7332.30	20.94
400	12875.90	32.19	8982.60	22.45
450	15027.60	33.39	10729.94	23.84
500	17058.79	34.12	12508.65	25.01
550	18773.60	34.13	14285.46	25.97
600	20055.82	33.43	16084.77	26.80
650	21050.16	32.38	18025.33	27.73
700	22073.24	31.53	20316.17	29.02
750	23354.45	31.14	23096.08	30.79
800	24928.72	31.16	26120.43	32.65

The effort column corresponds to the output of the model; however, the ratio column corresponds to the division of the effort by the size. As a comparison between the RBFNN and GRNN models in the verification stage, in the first data point, we notice that the RBFNN models underestimates small projects, however, the GRNN model overestimates small projects. With respect to the other 15 data points, both models perform well with an advantage to the GRNN model as it seems to be more robust. The accuracy of these models is presented in Section 4.9.

4.9 Models Evaluation and Comparison

This section presents the evaluation of the regression and neural network models. The models were evaluated using 65 industrial projects using different evaluation criteria such as MMRE, MMER, PRED and the Mean Error with 95% Confidence Interval (CI). Our model is compared with other models that conduct software estimation from the use case diagrams such as the original UCP and Schneider's et al. models. Furthermore, a discussion is provided in regards to the assessment of models.

4.9.1 Project Dataset

This research is based on software effort prediction from use case diagrams. We have encountered many difficulties in acquiring industrial projects because revealing UML diagrams of projects is considered confidential to many companies. For this reason, we have prepared a questionnaire that could help us obtain industrial data without actually having UML diagrams. In this questionnaire, we asked for example, the quantity of use cases in each project, the number of transactions in the Main Success Scenario and in the Extensions part, actual software size and effort as well as some non-functional requirements such as the project complexity, uncertainty in requirements, and factors contributing to productivity. Two hundred and fourteen industrial projects were collected from three main sources using the questionnaire presented in Appendix B. These include (See Appendix E and Appendix F):

- Source 1: One hundred and fifty six projects of software efforts vary between 120 person-hours and 60,826 person-hours were used as part of our whole dataset. The main architecture of these projects is web architecture. Application types include customer billing software, network management, insurance software, as well as human resource. Programming languages include C++, Powerbuilder, Java and .Net.
- Source 2: Thirteen projects were prepared that met our requirements. The range of the projects effort falls between 4,648 and 129,353 person-hours. Information about project types were kept confidential as required from the company.
- Source 3: This is a medium-sized company that employs 14 people to develop several projects such as information systems for chains of hotels, multi-branch universities and multi-warehouses book stores. The architectures used to develop these projects are 2-tier desktop application and 3-tier web architecture. The CASE tool used is Sybase PowerDesigner 12.5 and 15. Forty five projects of effort between 570 and 224,890 person-hours were collected.

4.9.2 Models Evaluation

To fairly compare the four models, same data points (65 projects) were used for evaluation. These data points contain 25 small projects (size less than 100 UCP), 21 medium-sized projects (size between 100 and 300 UCP) and 19 large projects (size larger than 300 UCP). These data points were not included in the training stage of the models. To thoroughly compare these models, we have conducted four experiments. In the first

experiment, the whole data points (65 projects) were used. In this experiment, the three non-linear regression models and the three linear regression models were used based on the value of software size (models that were developed based on the Small range are used to evaluate data points of size less than 100 UCP. The same is correct for other models). Then, we divided the whole dataset into three ranges; the Small range (25 projects), the Medium range (21 projects) and the Large range (19 projects). In each of the four experiments, our model (two regression and two neural network models) was evaluated against other models that predict software estimation from the use case diagrams such as the UCP and Schneider's model. The evaluation criteria used for testing are MMRE, MMER, and Mean Error with CI at 95%, as well as PRED (25), PRED (50), PRED (75) and PRED (100). The PRED values were calculated based on both the MMER and the MMRE criteria. Moreover, in each of the four different experiments, the interval plots at 95% CI of MMRE, MMER and Mean Error were constructed (Figures 4-49 to 4-60). The labels "non-ln", "UCP", "Sch", "ln", "RB" and "GR" correspond to "non-linear regression model", "UCP model", "Schneider's model", "linear regression model", "RBFNN model" and "GRNN model" respectively. The main goal of conducting four experiments is to see how each of the models performs for different software size ranges. Tables 4-11, 4-12, 4-13 and 4-14 show the evaluation results of the models based on the All, Small, Medium and Large ranges.

Table 4-11 Models evaluation- all data points

Criteria	Non-linear	UCP	Schneider	Linear	RBFNN	GRNN
MMRE	0.29	0.53	0.50	0.28	0.57	0.86
MMER	0.40	1.56	1.23	0.43	0.55	0.53
PRED_25_MMRE	49.23	12.3	10.7	49.23	32.30	35.38
PRED_50_MMRE	84.6	41.5	43	89.23	61.53	63.07
PRED_75_MMRE	98.46	86.15	96.9	98.46	73.84	80
PRED_100_MMRE	98.46	98.46	98.46	98.46	86.15	84
PRED_25_MMER	50.76	10.76	10.76	41.53	30.76	30.76
PRED_50_MMER	73.84	20	27.69	67.69	56.92	63.07
PRED_75_MMER	84.61	30.76	38.46	81.53	78.46	80
PRED_100_MMER	90.76	43.07	44.61	90.76	90.76	90.76
Mean Error CI(95%)	1366+/- 1503	9261+/- 6316	8674+/- 6234	4350+/- 3528	72004+/- 3134	154335+/- 2208

Table 4-12 Models evaluation- small range

Criteria	Non-linear	UCP	Schneider	Linear	RBFNN	GRNN
MMRE	0.29	0.55	0.53	0.25	0.75	1.66
MMER	0.29	1.31	1.14	0.31	0.43	0.43
PRED_25_MMRE	64	8	4	76	24	28
PRED_50_MMRE	80	44	48	88	48	44
PRED_75_MMRE	96	88	96	96	76	60
PRED_100_MMRE	96	96	96	96	88	64
PRED_25_MMER	76	4	4	68	32	32
PRED_50_MMER	84	12	24	84	68	56
PRED_75_MMER	92	32	44	88	72	80
PRED_100_MMER	92	48	52	92	88	100
Mean Error CI(95%)	-152+/- 249	816+/- 294	752+/- 294	-1.34+/- 235	-158+/- 390	885+/- 354

Table 4-13 Models evaluation- medium range

Criteria	Non-linear	UCP	Schneider	Linear	RBFNN	GRNN
MMRE	0.41	0.55	0.50	0.41	0.57	0.48
MMER	0.67	1.87	1.33	0.70	0.86	0.76
PRED_25_MMRE	9.5	14.28	14.28	4.7	19.04	23.80
PRED_50_MMRE	80.95	33.33	33.33	85.71	52.38	57.14
PRED_75_MMRE	100	76.19	100	100	76.19	85.71
PRED_100_MMRE	100	100	100	100	85.71	95.23
PRED_25_MMER	9.5	19.04	19.04	4.76	19.04	9.52
PRED_50_MMER	47.61	23.80	28.57	42.58	38.09	61.90
PRED_75_MMER	66.66	23.80	28.57	61.90	57.14	71.42
PRED_100_MMER	85.71	33.33	33.33	85.71	76.19	76.19
Mean Error CI(95%)	2301+/- 1300	4096+/- 1612	3639+/- 1433	2424+/- 1260	1760+/- 1628	2223+/- 1674

Table 4-14 Models evaluation- large range

Criteria	Non-linear	UCP	Schneider	Linear	RBFNN	GRNN
MMRE	0.16	0.49	0.45	0.19	0.34	0.24
MMER	0.25	1.55	1.23	0.31	0.37	0.40
PRED_25_MMRE	73.68	15.78	15.78	63.15	52.63	57.89
PRED_50_MMRE	94.73	47.36	47.36	94.73	73.68	94.73
PRED_75_MMRE	100	94.73	94.73	100	84.21	100
PRED_100_MMRE	100	100	100	100	100	100
PRED_25_MMER	63.15	10.52	10.52	47.36	47.36	52.63
PRED_50_MMER	89.47	26.31	31.57	73.68	78.94	73.68
PRED_75_MMER	94.73	36.84	42.10	94.73	94.73	89.47
PRED_100_MMER	94.73	47.36	47.36	94.73	94.73	94.73
Mean Error CI(95%)	2330+/- 4920	26083+/- 19792	24661+/- 19714	12204+/- 11366	725+/- 10750	3713+/- 7287

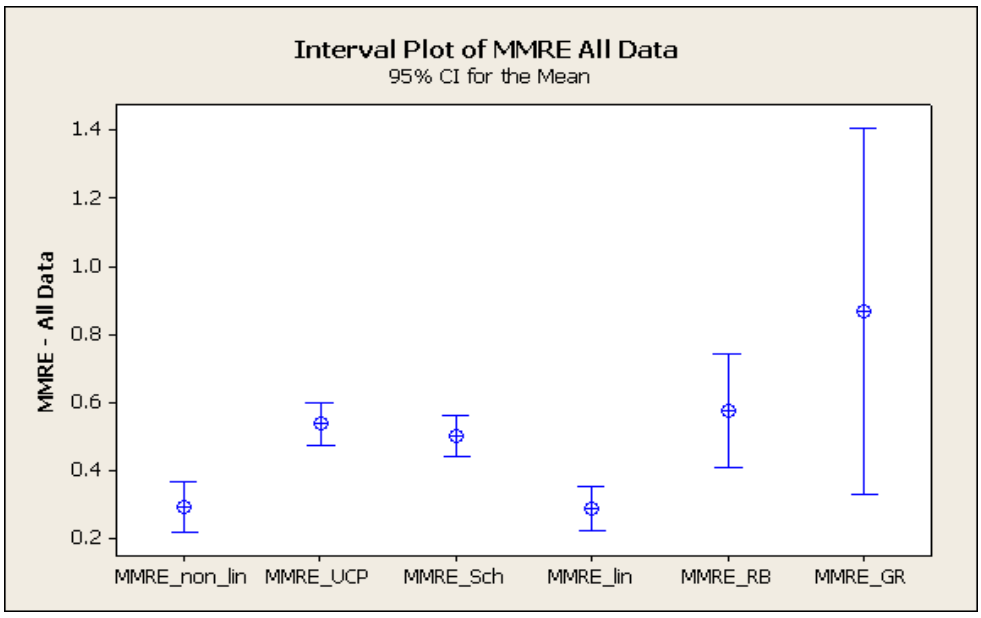


Figure 4-49 MMRE, all data

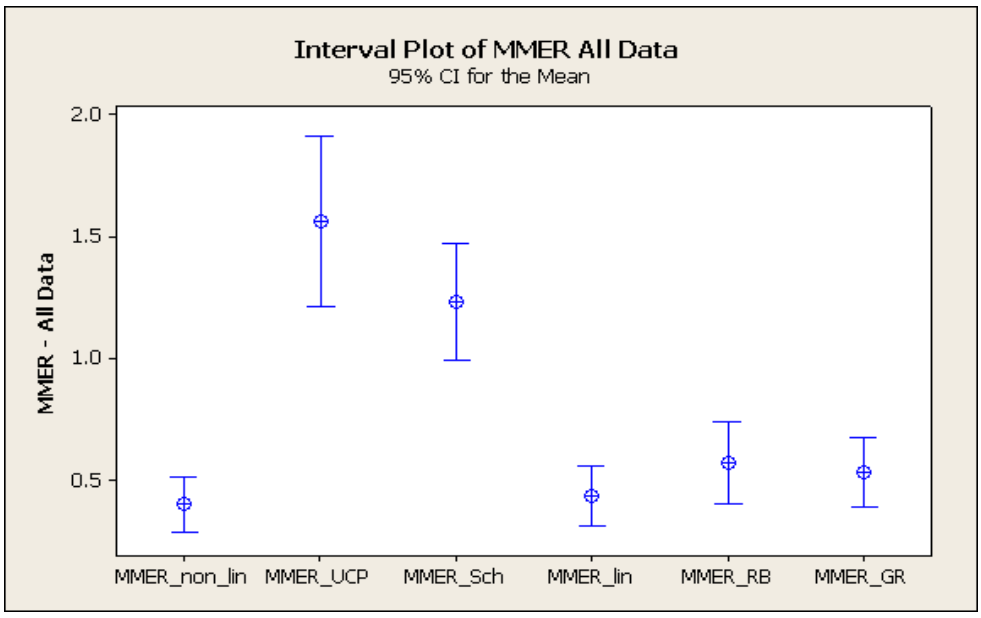


Figure 4-50 MMER, all data

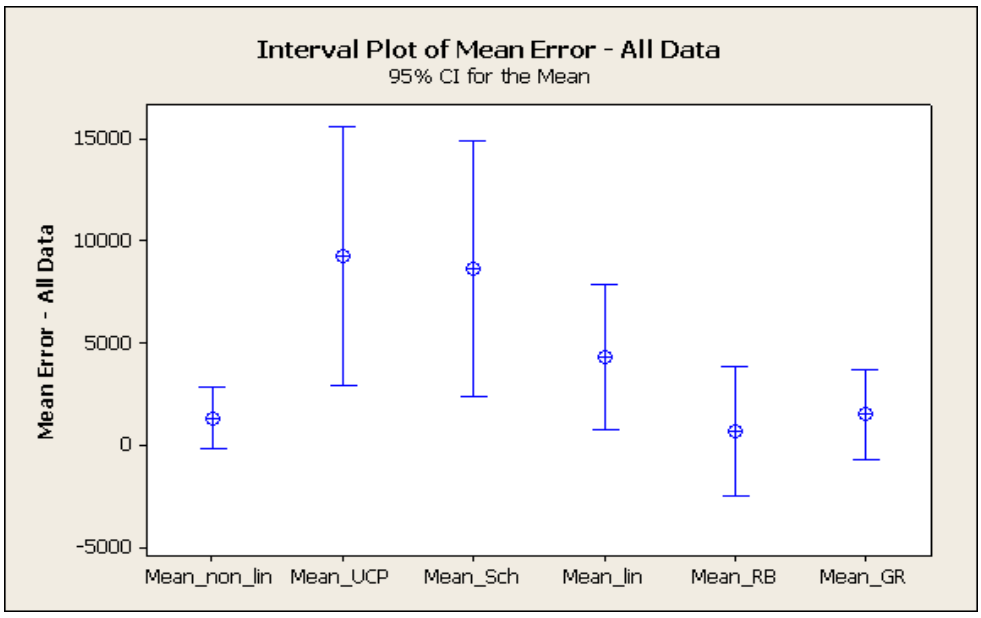


Figure 4-51 Mean error, all data

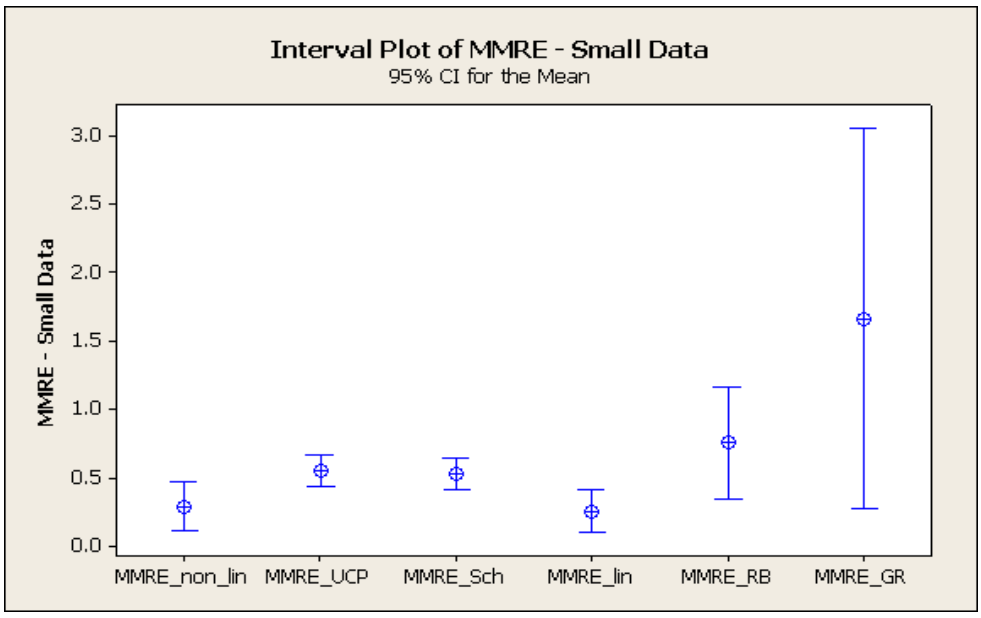


Figure 4-52 MMRE, small data

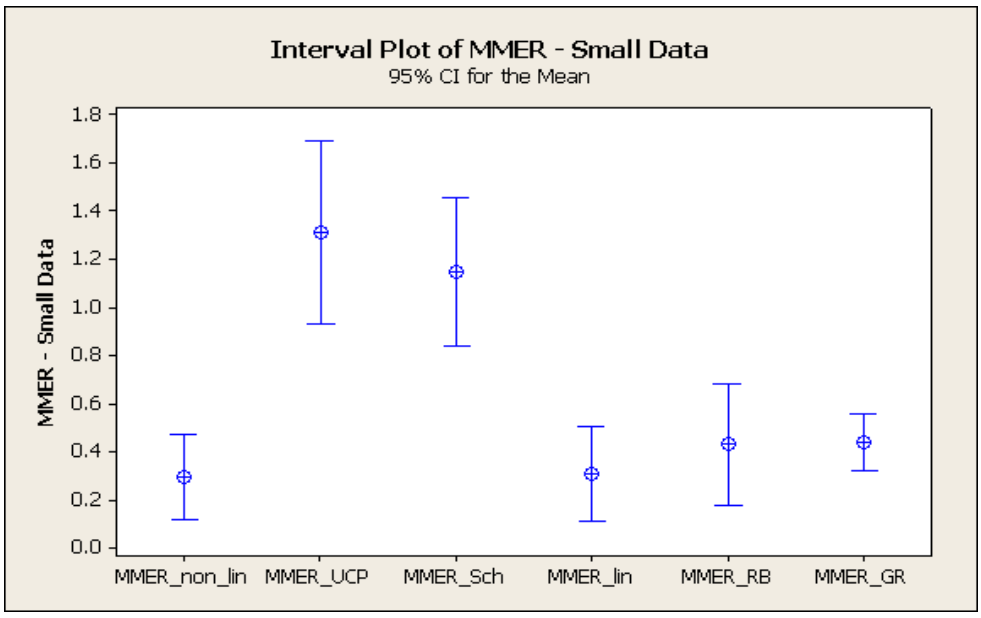


Figure 4-53 MMER, small data

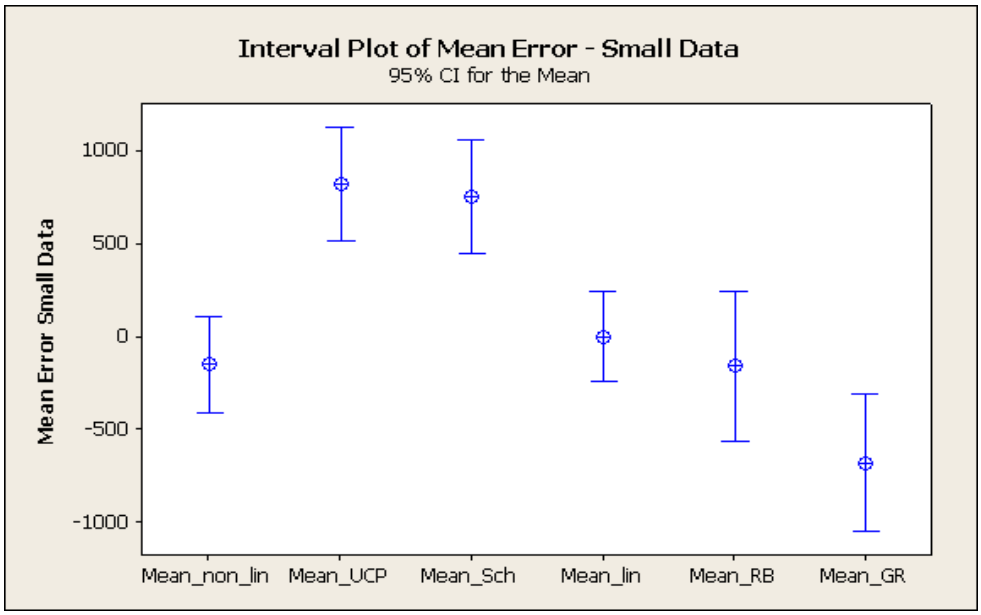


Figure 4-54 Mean error, small data

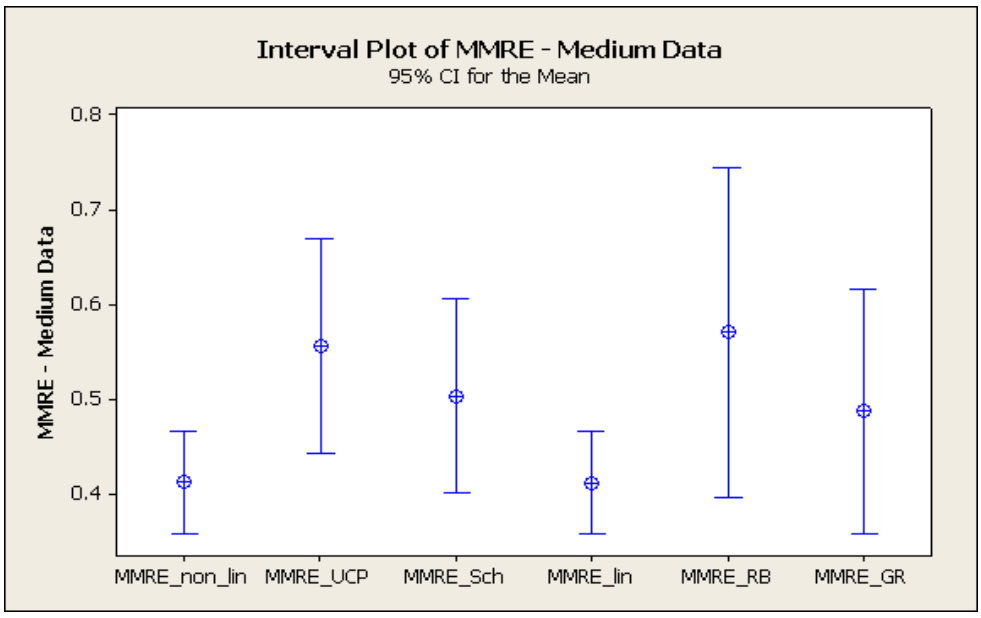


Figure 4-55 MMRE, medium data

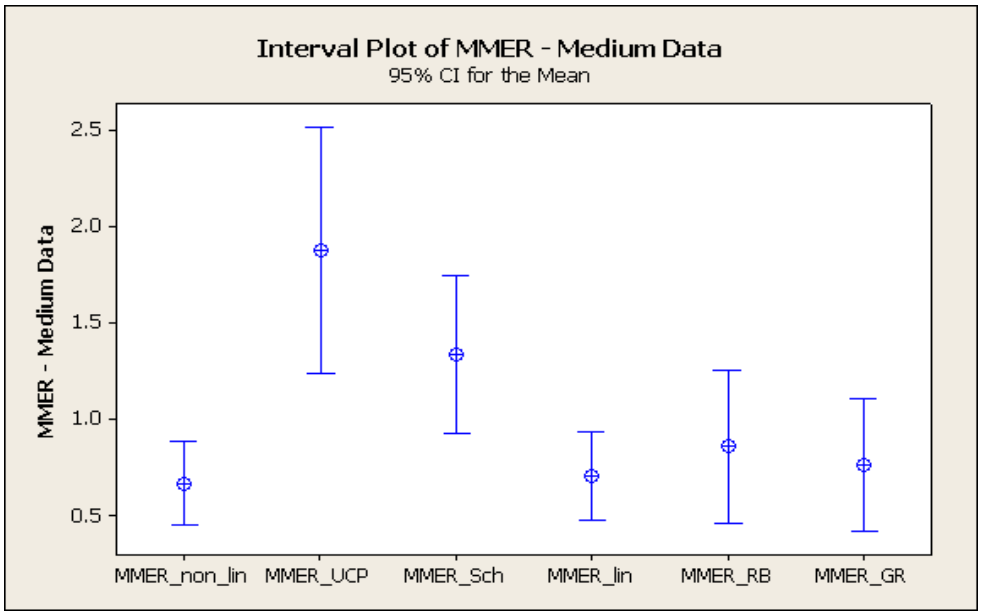


Figure 4-56 MMER, medium data

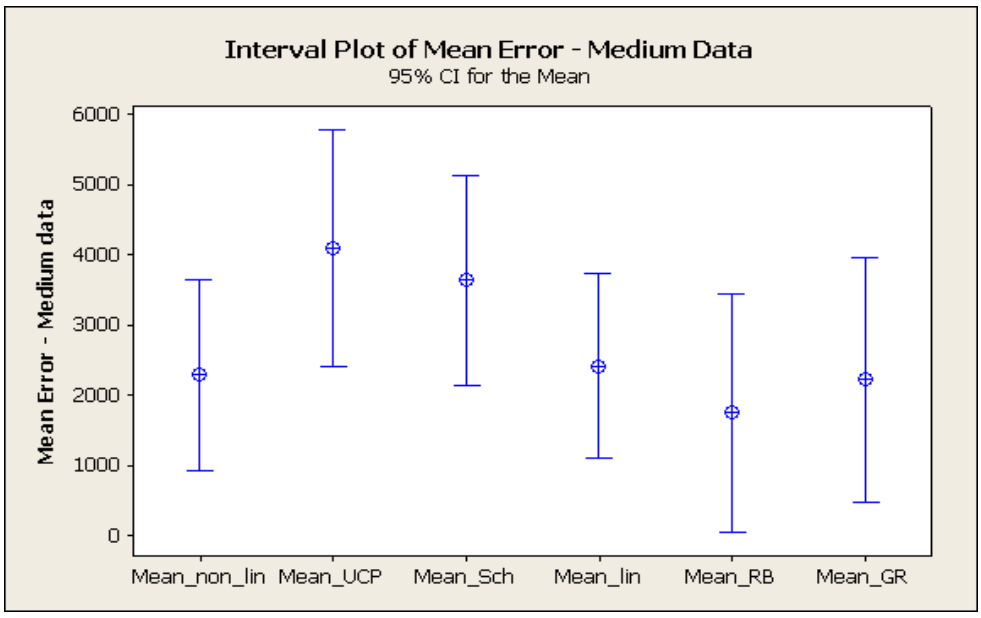


Figure 4-57 Mean error, medium data

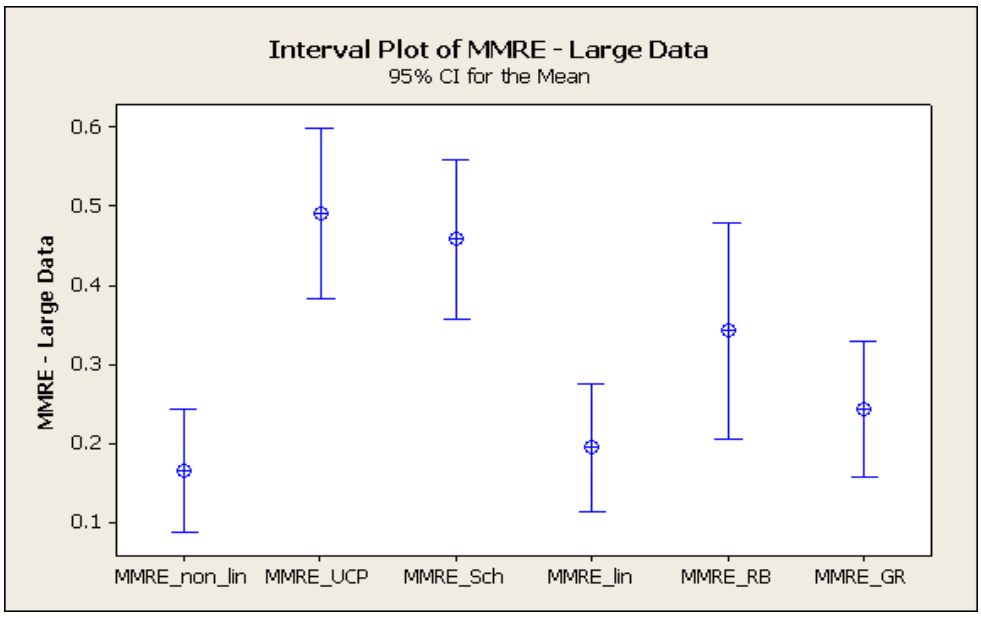


Figure 4-58 MMRE, large data

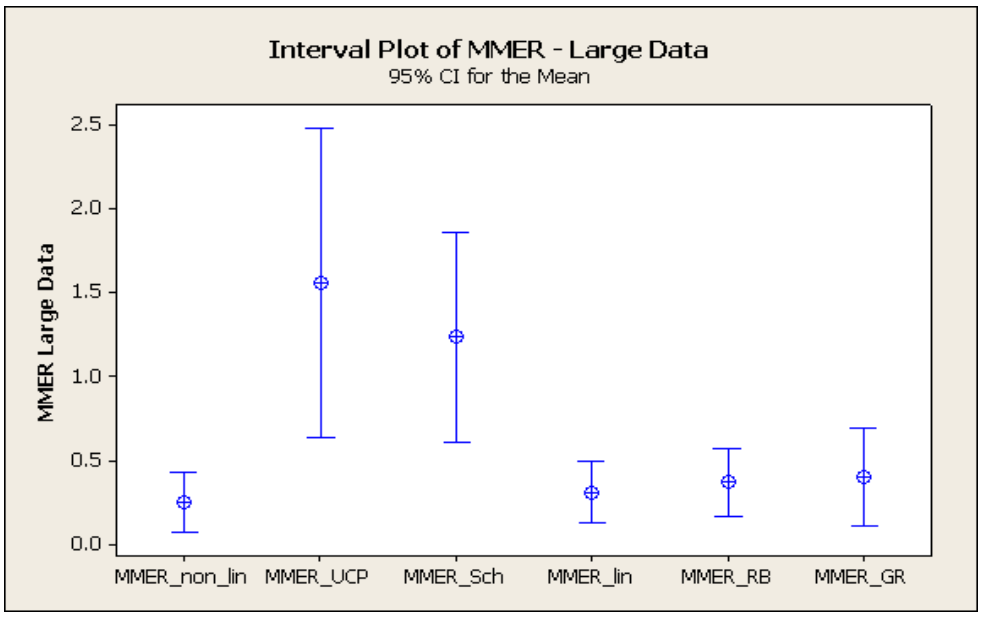


Figure 4-59 MMER, large data

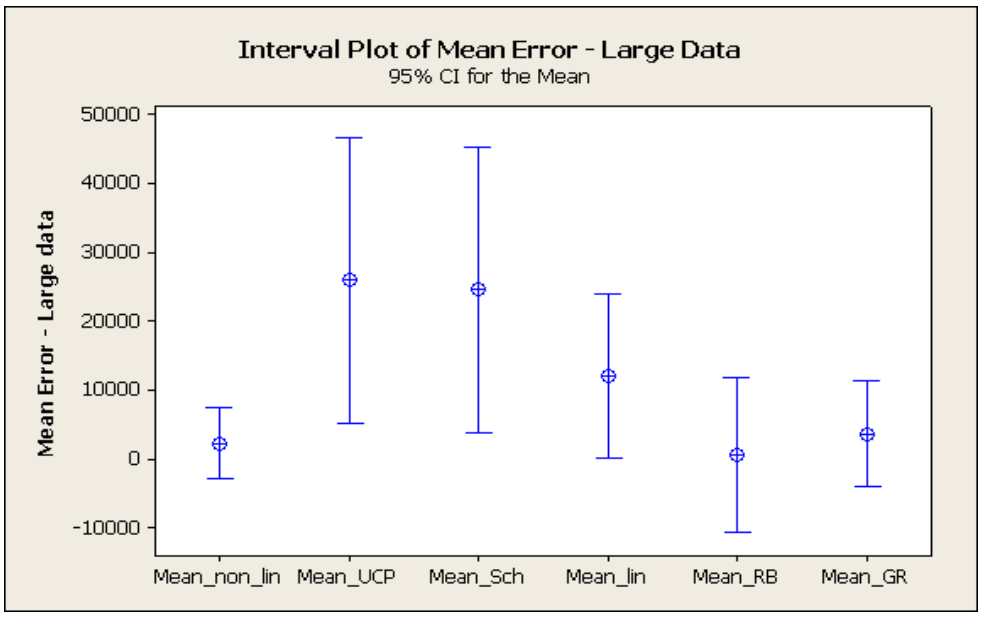


Figure 4-60 Mean Error, large data

4.9.3 Comparison Between Models

In this section, we will compare the proposed four models as well as the UCP and Schneider's models based on the above four experiments. In the first part of the comparison, all testing data points (65 projects) were used. On the second comparison, small testing data points (25 projects) were used. Medium (21 projects) and large (19 projects) data points were used in the third and fourth comparisons respectively.

4.9.3.1 Comparison With All Data Points

Table (4-11) and Figures 4-49 to 4-51 show the evaluation of the proposed four models as well as the UCP and Schneider's. models when all testing data points were used. Table (4-11) shows that the proposed four models outperform the UCP and Schneider's models. We noticed that the UCP and Schneider's models deteriorate when the MMER criterion is used. This means that these two models underestimate the value of the predicted effort. For instance, the non-linear regression model surpasses the UCP and Schneider's models by 116% and 83% respectively when the MMER criterion is used. Moreover, the non-linear model slightly surpasses the linear one especially in MMER and PRED which is based on MMER. Regarding the neural network models, we notice that the RBFNN model competes with the GRNN model when the MMRE criterion is used. On the other hand, the GRNN model gives better results when other criteria are used, so we can deduce that the GRNN model is better than the RBFNN model in this case. Figures 4-49 to 4-51 show the interval level of all models based on three criteria. Figure 4-49 shows

that the GRNN model has the largest variation based on the MMRE criterion which is not good.

4.9.3.2 Comparison With Small Data Points

During the training process of the regression models, we noticed that the linear regression (Figure 4-42) was better than the non-linear model (Figure 4-9). The R^2 and RMS values of the linear model are 0.89 and 141.5 respectively; however, these values are 0.84 and 167 with the non-linear model. The same observation was noticed in the testing stage even with data points that were not used in the training. Table 4-12 shows that the MMER value of the non-linear model was slightly better than the linear model. On the contrary, the linear model surpasses the non-linear based on all other criteria. The MMRE values of the UCP and Schneider's models are acceptable; however, these models are still suffering from underestimating software effort as shown in the MMER values. Regarding the neural network models, the RBFNN surpasses the GRNN model when small testing data points are used. Figure 4-52 shows that the GRNN model is the worst, while Figures 4-53 and 4-54 show that the UCP and Schneider's models are the worst.

4.9.3.3 Comparison With Medium-Sized Data Points

In the training stage, the non-linear model (Figure 4-16) outperforms the linear model (Figure 4-43). This remains true in the evaluation process where the non-linear model surpasses the linear model and other models in all criteria. The UCP and Schneider's models are the worst in this category. Furthermore, the GRNN model surpasses the RBFNN in this category.

4.9.3.4 Comparison With Large Data Points

When large data points were used for testing, we noticed that the non-linear model is the best model where the linear model comes second. This conclusion was also correct in the training stage. The GRNN model also outperforms the RBFNN in this category. The results of the UCP and Schneider's models are very far from the actual results based on all criteria.

We can conclude that in general the non-linear regression model has the best results among all the models, where the linear model comes second. We also noticed that the GRNN model is better than the RBFNN model. However, we observed that the UCP and Schneider's models worsen dramatically when the size of the data points becomes larger as shown in Figures 4-58 to 4-60. Typically, these models become inappropriate to use for projects of effort greater than 10,000 person-hours. The reason that the UCP and Schneider's models do not perform well with large projects is because these models define a use case as "complex" when the number of transactions of this use case is more than 7. Based on our dataset, we found that many use cases have more than 20 transactions. Another reason that contributes to the problem of the UCP and Schneider's models when used with large projects is that these models assume that the relationship between software effort and size is linear.

Based on this comparison, we notice that linear and non-linear regression models as well as RBFNN and GRNN models can be used for software effort prediction. This answers the last research question proposed in Section 1.2.

4.10 Threats to Validity

Threats to validity can be summarized as follows:

- Four proposed models are used to predict software effort for different ranges of software size. Nonetheless, our model has a limitation and cannot be used for projects of more than 4,000 UCP (around 150,000 person-hours). The non-linear regression model is more sensitive than the linear regression one with large projects because the equation used in the large range of projects is exponential. This means that a slight increase in software size over the size limit might cause a dramatic increase in software effort.
- Based on the data points that we have, the actual effort of the very small projects (size less than 25 UCP) is much larger than the predicted effort. This might be because in some companies, there is a base cost in project development no matter how small the size is. For instance, the predicted effort of a project that has two small use cases was 55 person-hours; however the actual effort is 378 person-hours. Nevertheless, our model was not highly affected by this change because only 4% of the data points are considered as very small.
- Regarding size estimation (Table 4-1), the largest use case is defined when the number of transactions is more than 20. Although this is much better than the definition of the largest use case of the UCP model (greater than 7 transactions), we have noticed that the number of transactions of some use cases in large

projects is about 40. This means that the size of these huge use cases is underestimated.

- It was difficult to elicit the factors that contribute to Productivity (Section 4.2.3) from the team that is developing software projects. For instance, developers might be optimistic when answering questions about their experiences and motivations. Moreover, the motivation of a developer/programmer might differ when placed in a different team, even in the same project. Furthermore, there is no straightforward rule to calculate the productivity of the team based on the productivity of each team member. In this work, the average of all team members was performed to calculate the productivity of the team. Furthermore, the productivity factors were obtained from the project manager of each project and not from the actual people who were involved in developing the projects.
- We were not able to obtain copies of the use case diagrams of the projects because they are considered confidential and proprietary. We therefore simply relied on the information provided by those who were involved in preparing the data used. For instance, an error in counting the number of transactions of a use case in either the Main Success Scenario or the Extensions part might lead to a flaw in the design of our model.

4.11 Conclusions

This chapter introduces a novel model based on four sub-models to predict software estimation from use case diagrams. These models include non-linear regression, linear

regression with a logarithmic transformation, Radial Basis Function Neural Network (RBFNN) and General Regression Neural Network (GRNN). The first step of our model was to estimate the size of a project by counting the number of use cases in the use case diagram as well as the number of transactions in both the Main Success Scenario and the Extensions part. Moreover, the project complexity, team productivity and the degree of requirements uncertainty are factors in the effort estimation. We have also proved that the relationship between software effort and size is non-linear because when software size increases, the number of team members required to develop this software increases. When the team becomes larger, communication overhead will incur and this requires additional effort. This concludes that when software size increases, software effort will increase exponentially. Furthermore, when building regression models, we found that one regression equation cannot fit all project datasets of different size ranges. For this purpose, we proposed three non-linear equations as well as three linear equations for software effort estimation that can be used with three different ranges (Small, Medium and Large) of software size. In the non-linear regression, a second degree polynomial equation was proposed for the Small range and two different exponential equations were proposed for the Medium and Large projects respectively. We have noticed that the non-linear relationship is not significant in the Small projects and thus, the linear regression model performed better in this range. However, the non-linear relationship stands out in the Medium and Large projects.

Two neural network models were also proposed to predict software effort. The GRNN model was slightly better than the RBFNN. We also showed that the RBFNN and GRNN

models can be used for software effort prediction as alternatives method to linear and non-linear regression.

We have collected 214 industrial projects from three different sources. Sixty five projects were used to train the regression models; however, 149 projects were used to train the neural network models. All models were evaluated (tested) using 65 projects based on four experiments which include evaluation using all data points (65 projects), evaluation using small data points (25 projects), evaluation using medium-sized data points (21 projects) and evaluation using large data points (19 projects). Our model was also evaluated against two other models (UCP and Schneider) that predict software effort from use cases. We used four different evaluation criteria; MMRE, MMER, PRED and the Mean Error with 95% Confidence Interval (CI). The proposed model gave promising results in comparison with the other two models and especially with the MMER criterion. Our model is limited to projects of maximum effort around 150,000 person-hours. Despite this limitation, we believe that our model can widely be applied since 150,000 person-hours projects are classified as large in the eye of the industry.

References

[1] G. Karner, "Resource Estimation for Objectory Projects," *Objective Systems,* 1993.

[2] M. Ochodek, J. Nawrocki and K. Kwarciak, "Simplifying effort estimation based on Use Case Points," *Information and Software Technology,* vol. 53, pp. 200-213, 2011.

[3] D. Baccarini, "The concept of project complexity—a review," *Int. J. Project Manage.,* vol. 14, pp. 201-204, 8, 1996.

[4] L. Ireland, "Project complexity: A brief exposure to difficult situations," Asapm, Tech. Rep. PrezSez 10-2007, 2007.

[5] D. D. Galorath and M. W. Evans, *Software Sizing, Estimation, and Risk Management.* Boston, MA, USA: Auerbach Publications, 2006.

[6] S. McConnell, *Software Estimation: Demystifying the Black Art.* Redmond, Washington: Microsoft, 2006.

[7] B. Boehm, C. Abts, W. Brown and S. Chulani, *Software Cost Estimation with COCOMO II.* Upper Saddle River, New Jersey: Addison Wesley, 2000.

[8] E. H. Mamdani, "Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis," *IEEE Transactions on Computers,* vol. C-26, pp. 1182-1191, 1977.

[9] Z. Xu and T. M. Khoshgoftaar, "Identification of fuzzy models of software cost estimation," *Fuzzy Sets and Systems*, vol. 145, pp. 141-163, 2004.

[10] P. D. Allison, *Event History Analysis: Regression for Longitudinal Event Data.* Sage Publications, 1984.

[11] A. Legendre, *Nouvelles Méthodes Pour La Détermination Des Orbites Des Comètes. "Sur La Méthode Des Moindres Quarrés".* 1805.

[12] C. F. Gauss, *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientum.* 1809.

[13] A. B. Nassif, D. Ho and L. F. Capretz, "Regression model for software effort estimation based on the use case point method," in *2011 International Conference on Computer and Software Modeling*, Singapore, 2011, pp. 117-121.

[14] A. B. Nassif, L. F. Capretz and D. Ho, "Estimating software effort based on use case point model using sugeno fuzzy inference system," in *23rd IEEE International Conference on Tools with Artificial Intelligence*, Florida, USA, 2011, pp. 393-398.

[15] W. Humphrey, *A Discipline for Software Engineering.* Addison Wesley, 1995.

[16] A. C. Cameron and P. K. Trivedi, *Regression Analysis of Count Data.* Cambridge, UK: Cambridge University Press, 1998.

[17] J. M. Zhu, P. Du and T. T. Fu, "Research for RBF Neural Networks Modeling Accuracy of Determining the Basis Function Center Based on Clustering Methods," *Advanced Materials Research,* vol. 317-319, pp. 1529-1536, 2011.

[18] S. Chen, X. Hong and C. J. Harris, "Orthogonal forward selection for constructing the radial basis function network with tunable nodes," in *IEEE International Conference on Intelligent Computing*, 2005, pp. 777-786.

Chapter 5

5. A Treeboost Model for Software Effort Estimation[4]

5.1 Introduction

This chapter presents a Treeboost model to predict software effort from use case diagrams based on three independent variables (predictors). These predictors include software size, productivity and complexity. The Treeboost model was trained using 168 data points and evaluated using 69 projects. To measure the accuracy of the proposed model, a multiple linear regression model was developed based on the same 168 projects. The Treeboost model was then evaluated against the multiple linear regression model developed as well as the use case point model. The evaluation criteria used in this chapter are MMRE, MMER, PRED(x) and MAE.

The Treeboost algorithm was introduced by J. Friedman [1] [2]. This algorithm was put forward to improve the accuracy of decision trees models. The Treeboost algorithm has been applied in many fields such as ecology [9], fresh water studies [10], earth and

---

[4] This chapter has been submitted to the International Conference of Predictive Models in Software Engineering (PROMISE 2012)

1- **Ali Bou Nassif**, Danny Ho and Luiz Fernando Capretz, "A Treeboost Model for Software Effort Estimation Based on Three Independent Variables", *Predictive Models in Software Engineering*, 2012 (Under review).

environmental science [11] and agronomy [12]. Section 5.2 defines the decision tree model, followed by the introduction of the model's inputs in Section 5.3. The Treeboost and the multiple linear regression models are discussed in Sections 5.4 and 5.5, respectively. The evaluation of models and a discussion on the results are presented in Sections 5.6 and 5.7 respectively. Section 5.8 lists threats to validity and Section 5.9 concludes the chapter.

5.2 Decision Tree Model

A decision tree is a logical model that is mainly used in operations research, specifically in decision analysis. A decision tree is composed of nodes of which the topmost node is called the root. Each node is split into two nodes (children) until a decision is satisfied. A node with no children is called a terminal node or a leaf. A node is split based on the condition of a predictor after an analysis of the input data (data points used to train the decision tree model). Examples of this analysis include the study of the influence of each predictor on the dependent variable. For instance, based on the analysis of the variables of the proposed model, when calculating the dependent variable "effort", the predictor "size" is more important that the predictors "productivity" and "complexity". The type of the dependent variable (target) can be continuous (e.g. 100, 200, 500, etc.) or categorical (e.g. "male", "female"). If the target variable is continuous, the name of the decision tree is called Regression. However, if the target variable is categorical, the name of the decision tree is called Classification. There are many available tools to generate decision trees such as Automatic Interaction Detection (AID) [3], CHAID [4], THAID [5] and

DTREG [6]. Figure (5-1) shows an example of a decision tree to build a model to predict software effort from three independent variables (size, productivity and complexity). The model is trained using 168 data points. For instance, the proposed Treeboost model (Section 5.4) uses the same training data points and variables used to train the decision tree model (Figure 5-1). In this figure, the root node is fed with 168 data points for training purposes. The average (mean) effort of these data points is 7,189 person-hours. The root node (Node 1) is split into two nodes (Node 2 and Node 3) based on a value of the size (size=570). Similarly, Nodes 2, 3, 4 and 12 are split based on the variable "size". Nodes 5, 6 and 7 are split based on the variable "productivity". However, Node 13 is split based on the variable "complexity".

The main advantage of the decision tree model is that it displays the problem and its solution at a level that be comprehended by technical and non-technical people.

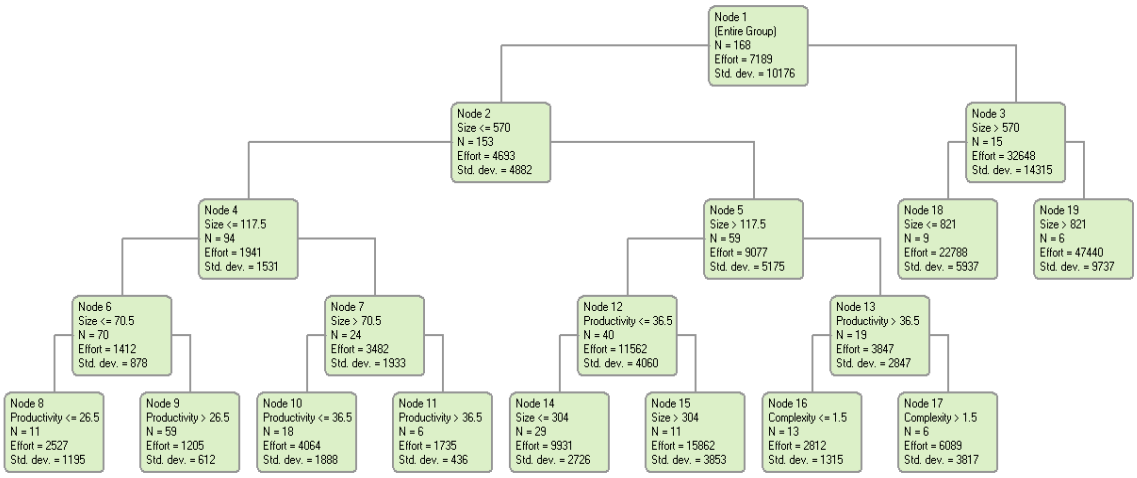


Figure 5-1 Decision tree model

The main limitation of the decision tree model is that it lacks accuracy if the number of training rows is insufficient. To enhance the accuracy of the decision tree model, the Treeboost model is brought into play.

5.3 Model's Inputs

This chapter focuses on predicting software effort from use case diagrams based on three independent variables. These include software size, productivity and complexity. Software size was computed based on the use case point method (Equation 2.22). The reason that requirements stability was not included as an independent variable (as was the case in Chapter 4), is because in this experiment, a multiple linear regression model was created to compare it against the Treeboost model. Generating multiple linear regression models based on four independent variables (if requirements stability was an independent

factor) becomes inappropriate if the number of training rows is small. Requirements stability factor was one of the factors that contributed to the productivity factor (Table 2-6). To compensate the importance of the requirements stability factor, several experiments were conducted to adjust its weight by assigning values from 2 to 5, incremented by 0.5. The multiple linear regression model (Equation 5.3) was generated based on each trial (from 2 to 5 incremented by 5). The highest R^2 value was achieved when the requirements stability weight was "4". This indicates that the requirements stability weight was modified from 2 (initial weight proposed by the UCP model) to 4. The productivity factor was calculated according to this equation:

$$\text{Pr}\,oductivity = \sum_{i=1}^{8} E_i \times W_i. \qquad (5.1)$$

Where E_i and W_i are the Environmental factors and their corresponding weights as depicted in Table (2-6) with one exception for the weight of the requirements stability factor (weight is 4 instead of 2).

Regarding project complexity, we introduce five levels of complexity based on these rules:

- Level-1: The complexity of a project is classified as Level-1 if the project team is familiar with this type of project and the team has developed similar projects in the past. The number and type of interfaces are simple. The project will be installed in normal conditions where high security or safety factors are not required. Moreover, Level-1 projects are those for which around 20% of their design or implementation

parts are reusable (came from old similar projects). The weight of the Level-1 complexity is 1.

- Level-2: This is similar to Level-1 category except that only about 10% of these projects are reusable. The weight of the Level-2 complexity is 2.
- Level-3: This is the normal complexity level where projects are not said to be simple, nor complex. In this level, the technology, interface, and installation conditions are normal. Furthermore, no parts of the projects had been previously designed or implemented. The weight of the Level-3 complexity is 3.
- Level-4: In this level, the project is required to be installed using a complicated topology/architecture such as distributed systems. Furthermore, in this level, the number of variables and interface is large. The weight of the Level-4 complexity is 4.
- Level-5: This is similar to Level-4 but with additional constraints such as a special type of security or high safety factors. The weight of the Level-5 complexity is 5.

Please note that software effort is inversely proportional to productivity and proportional to complexity.

5.4 The Treeboost Model

The Treeboost model is also called Stochastic Gradient Boosting (SGB) [2]. Boosting is a method to increase the accuracy of a predictive function by applying the function frequently in a series and combining the output of each function. In other words, as Kearns once asked [7], “can a set of weak learners create a single strong learner?”. The

main difference between the Treeboost model and a single decision tree is that the Treeboost model consists of a series of trees. The main limitation of the Treeboost is that it acts like a black box (similar to some neural network models) and cannot represent a big picture of the problem as a single decision tree does. The Treeboost model has the following characteristics:

- The Treeboost uses Huber-M loss function [8] for regression. This function is a hybrid of ordinary least squares (OLS) and Least Absolute Deviation (LAD). For residuals which are less than a cutoff point (Huber's Quantile Cutoff), the square of the residuals is used. Otherwise, absolute values are used. This method is used to alleviate the influence of outliers. For outliers, where residuals have high values, squaring the residuals will lead to huge values, so outliers will be treated with the "absolute values" method instead. The Huber's Quantile Cutoff value is recommended to be between 0.9 and 0.95. If it is 0.9, the residuals will first be sorted from small to high. Then, the smallest 90% of the residuals will be squared (OLS) and the other residuals (largest 10%) will be treated with the LAD method.
- In the Stochastic Gradient Boosting algorithm, "Stochastic" means that instead of using all data for training, a random percentage of training data points (50% is recommended) will be used for each iteration instead. This has yielded an improvement in the results.
- The Stochastic Gradient Boosting (SGB) algorithm has a factor called *Shrinkage* factor. Experiments show that multiplying each tree in the series by this factor

(between 0 and 1) will delay the learning process and consequently, the length of the series will be longer to compensate for the shrinkage. This also leads to better prediction values.

- To improve the optimization of the process, an Influence Trimming Factor is applied. In the Treeboost model, the residual errors of a tree are used as inputs to the next consecutive iteration. The Influence Trimming Factor allows the rows with small residuals to be excluded. If this factor is 0.10, rows with residuals that represent less than 10% of the total residual weight will be ignored.

The Treeboost algorithm is described in Equation (5.2):

$$F(x) = F_0 + A1 * T1(x) + A2 * T2(x) + ... + AM * TM(x). \qquad (5.2)$$

Where F(x) is the predicted target, F_0 is the starting value, x is a vector which represents the pseudo-residuals, T1(x) is the first tree of the series that fits the pseudo-residuals (as defined below) and A1, A2, etc. are coefficients of the tree nodes. The Treeboost algorithm is applied based on the following rules:

1- Find the coefficient of F_0. This is the mean of the target variable.
2- Select the rows that will feed the next tree. If the stochastic factor is set to 0.5, 50% of the rows will be randomly chosen.
3- Sort the residuals of the rows being used and transform the residuals using Huber's Quantile Cutoff factor. The transformed residual values are called pseudo-residuals.

4- Fit the first tree (T1) to pseudo-residuals.

5- Calculate the mean of the pseudo-residuals in each of the terminal nodes. This mean becomes the predicted variable of the node.

6- Calculate the residuals between the predicted variable and the pseudo-residuals that fed the tree, and apply Huber's Quantile Cutoff factor again. Then, compute the mean of these residuals.

7- Calculate the boost coefficient (A1) of the node which is the difference between the mean residual value and the mean of the predicted values of the tree.

8- Multiply the boost coefficient by the shrink value to retard the learning process.

Regarding the use of the Treeboost algorithm in software estimation, one modest work has been published by M. Elish [13] that compares a Stochastic Gradient Boosting model with other neural and regression models. The main limitations of Elish's work include:

- The Stochastic factor was set to 1. This means that all data points were used for training. However, the main goal of the SGB algorithm (the stochastic part) is that a random portion of the training data should be used for training as opposed to using all data. By setting the Stochastic factor to "1", the Stochastic Boosting Algorithm will no longer be "stochastic".
- Some important parameters such as the number of trees and shrinkage factor are missing.
- The model and other neural and regression models were only trained using 18 projects. This is insufficient.

- The comparison conducted between the SGB and other models was based on training and generalization processes only. In other words, the models should have been tested with new data that were not included in the training process.

The Treeboost model proposed in our research work was trained using 168 data points based on the parameters listed in Table (5-1). Figure (5-2) shows the plot of the training data points used in the experiment and the training curve. Figure (5-3) shows the actual-predicted effort diagram. The model was developed based on a series of 1,000 trees. To avoid overfitting during the training process, 20% of the training rows were used for validation. As shown in Figure (5-4), best validation results (the blue line represents the training process and the red line represents the validation process) were obtained when the number of trees was 359. Appendix H shows the validation error for each tree.

The analysis of variance (ANOVA) shows that the coefficient of determination (R^2) and the Root Mean Squared Error (RMS) are 0.97 and 1,556, respectively in the training process. However, R^2 and RMS values in the validation process are 0.86 and 4,385, respectively.

Table 5-1 Model's Parameters

# of trees	Huber Quantile Cutoff	Shrinkage Factor	Stochastic Factor	Influence Trimming Factor
359	0.9	0.1	0.5	0.1

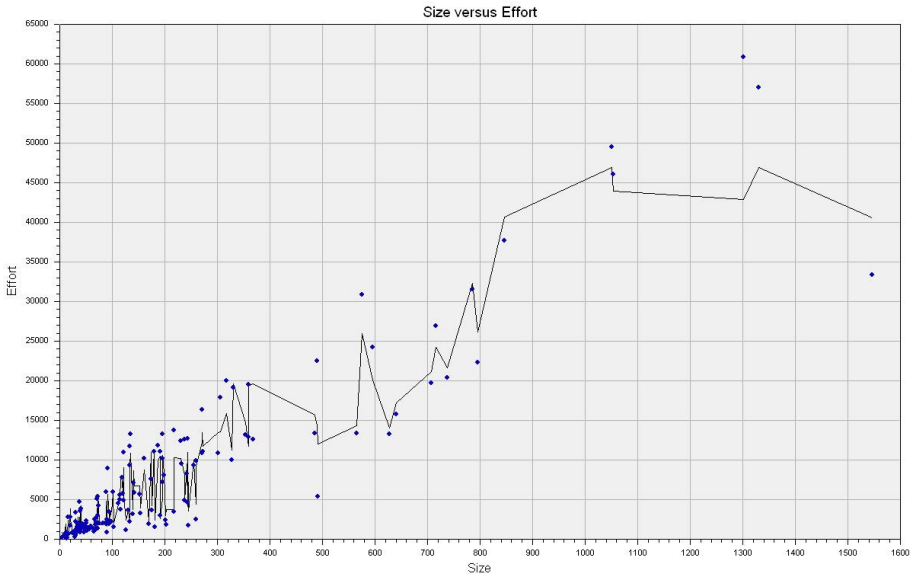


Figure 5-2 Data points used in training and the learning curve

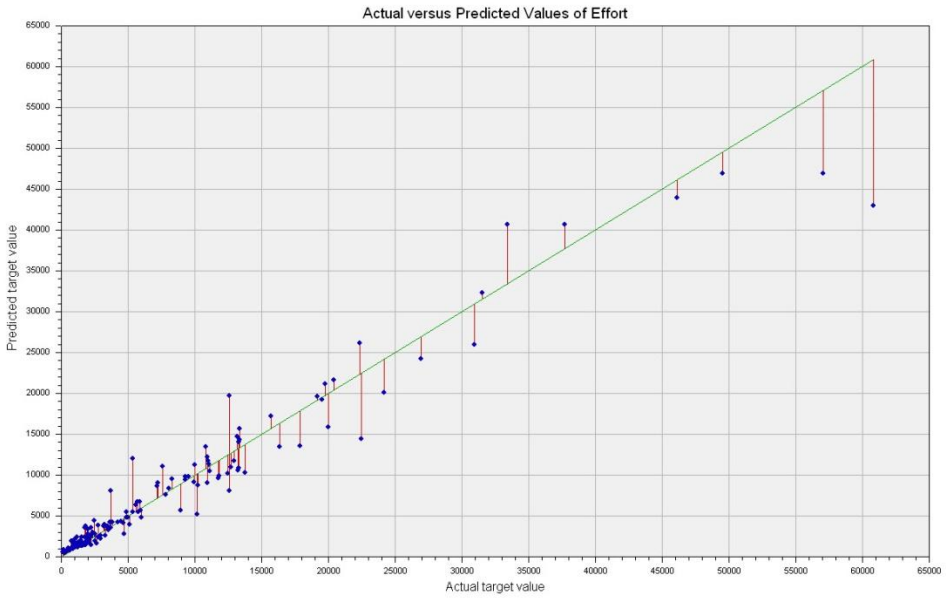


Figure 5-3 Actual versus predicted effort

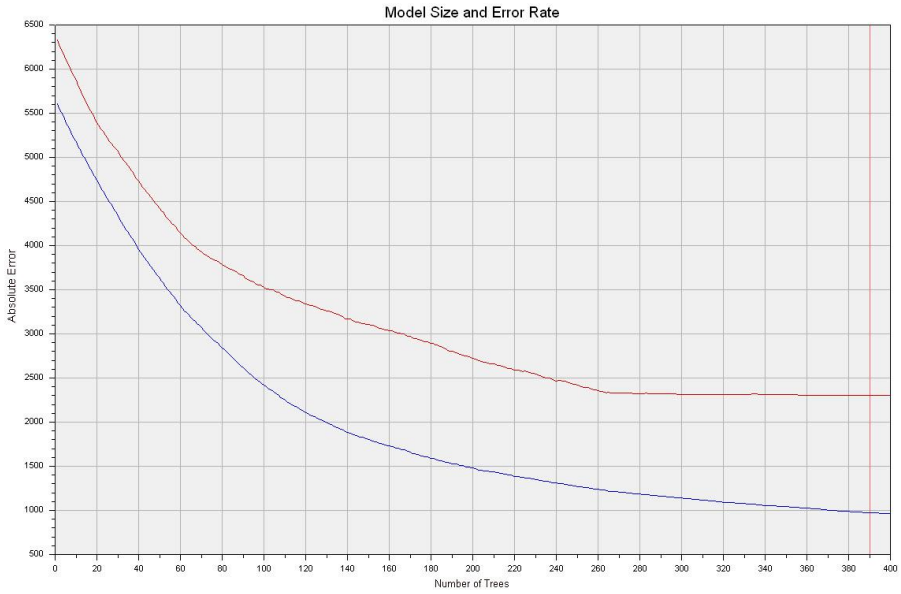


Figure 5-4 Number of trees, training and validation curves

5.5 Multiple Linear Regression Model

The multiple linear regression model was constructed using the same 168 data points that were used to train the Treeboost model. Minitab version 16 was used for this purpose. The equation of the regression model is:

$$Effort = 3661 + (32.7 \times Size) - (183 \times \Pr oductivity) + (1080 \times Complexity). \quad (5.3)$$

Where Effort is measured in person-hours, Size in UCP, Productivity is measured based on Equation (5.1) and Complexity is measured as proposed in Section (5.3). Equation (5.3) shows that Effort is proportional to Size and Complexity but inversely proportional to Productivity. This indicates that if the size or the complexity of a project increases,

software effort will increase. However, for the same software size and complexity, less effort is required to develop the project if a highly productive team is used.

To measure the accuracy of the regression model, we measured the value of the coefficient of determination R^2 which is 0.882. This indicates that approximately 88 % of the variation in Effort can be explained by the independent variables Size, Complexity and Productivity. Moreover, we measured the ANOVA and the model parameters. The "p" value of the model is 0.000 which indicates that there is a significant relationship among the variables at the 99% confidence level. The "p" values of the independent variables are 0.000 and 0.0083 for the constant. Since the highest "p" value of the model's parameters is less than 0.005, this indicates that all independent variables are significant at the 95% confidence level, and consequently the model is verified.

5.6 Model Evaluation

A total of 237 projects (211 industrial and 26 educational) were used in training and testing the model. The reason that only 211 industrial projects were used here as opposed to the 214 industrial projects used in section (4.9.1), is because the largest three projects were eliminated as they were larger than the largest project used in training the model. This is one of the limitations of the Treeboost model; the predicted effort of projects of size above a certain limit is the same. This is discussed in detail in Section 5.8 (Threats to Validity). Out of the 237 projects, 168 projects (70%) were used for training and 69 projects were used for evaluation. Four different criteria were used for evaluation. These include MMRE, MMER, PRED and MAE. The Treeboost model was evaluated against

the UCP model as well as the multiple linear regression model. Table (5-2) shows the evaluation results. Figures (5-5), (5-6) and (5-7) show the interval plots of MMRE, MMER and MAE at 95% confidence level, respectively.

Table 5-2 Evaluation results

Criteria	Treeboost	UCP	Multiple Regression
MMRE	0.44	0.40	0.93
MMER	0.35	1.06	0.51
PRED(25)	42	33	31
PRED(50)	75	49	63
PRED(75)	91	52	84
PRED(100)	94	62	88
MAE	2900	3890	3231

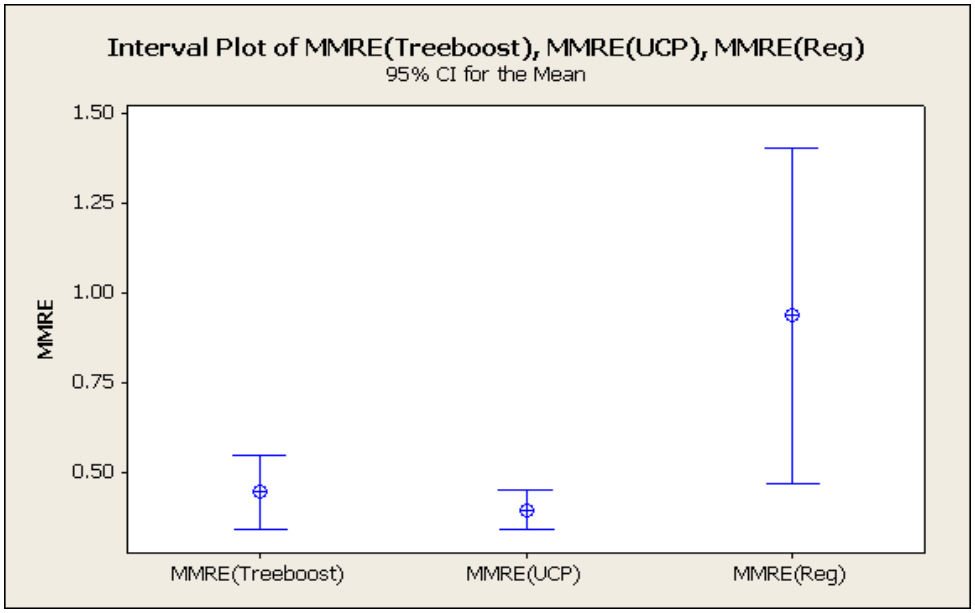


Figure 5-5 MMRE interval plot

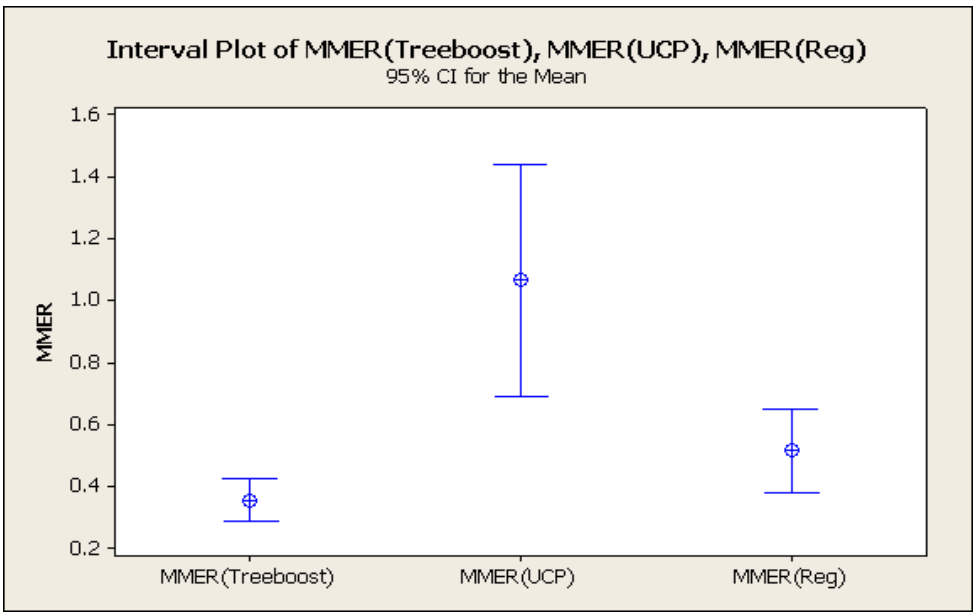


Figure 5-6 MMER interval plot

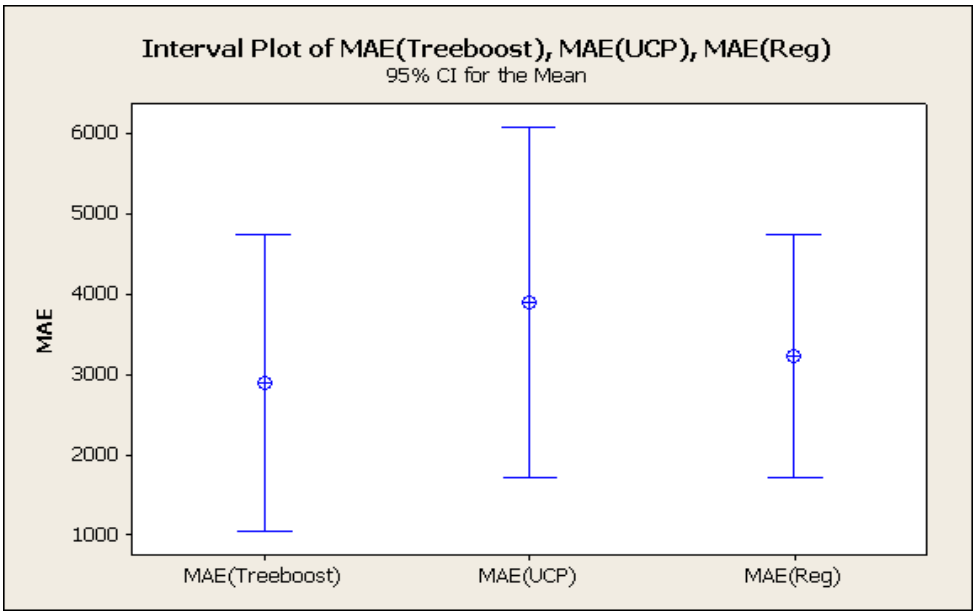


Figure 5-7 MAE interval plot

5.7 Discussion

Table (5-2) shows that the Treeboost model outperforms the UCP and the regression models when MMER, PRED and MAE criteria are used (lower MMER, MAE values and higher PRED values). The UCP model was improved by 71% based on the MMER criterion. The UCP model slightly surpasses the Treeboost model when MMRE was used. By comparing the MMRE and MMER of the UCP model, we notice that the average estimated effort of the UCP model is much less than the actual effort. As a comparison between the UCP and the multiple regression models, the multiple regression model outperforms the UCP model in all criteria except MMRE.

Figure (5-5) depicts that the multiple linear regression model is the most inferior model based on the MMRE criteria; not only in the mean value (0.93), but also in the variation of error (the multiple linear regression model has the longest interval). Figures (5-6) and (5-7) show how the UCP model deteriorates when MMER and MAE criteria are used.

Bases on the above results, we conclude that the Treeboost model can be used to predict software effort and can be competitive to other regression models. The Huber's loss function makes the model less sensitive to outliers. This indicates that this model is recommended to estimate projects if the project manager believes that the values of one or more independent variables might fall beyond the expected ranges.

5.8 Threats To Validity

1- The Treeboost model is a series of many small trees. The proposed model consists of 359 trees. The model was trained using 168 projects with efforts ranging between 120 and 60,862 person-hours. The mean value is 7,188 person-hours and the standard deviation is 10,206. This shows that there is a significant difference in size between the smallest and the largest data point. Despite the good results obtained from the evaluation of the Treeboost model, this model would perform better if more training data points would have been used.

2- The neural network and linear/non-linear regression models have the capability to extrapolate the relationship between input and output vectors during the training process and thus, can map outputs to inputs even if these inputs are beyond (to a certain degree) the inputs of the training data points. However, this is not true with Treeboost models. Based on the decision tree model (Figure 5-1), the node with the largest number (Node 19) handles the last decision. For example, the condition in Node 19 is that the predicted effort of projects of size larger than 821 UCP is 47,440 person-hours. This shows that the size limitation of testing data points is around 821 UCP. The Treeboost model works in a similar way, but it is more complicated than the single decision tree. Nonetheless, the Treeboost model also has limitations determined by the values of the three independent variables (size, productivity, complexity). To demonstrate this limitation, the Treeboost model was tested using 16 data points with sizes ranging between 800 and 1,600 UCP incremented by 50. Since software size is the most important predictor in the model, productivity and complexity values were set as 20 and 4, respectively for

all projects. Figure (5-8) shows the Scatterplot graph between software size and predicted effort. The graph shows that the predicted effort of any project with a size greater than 950 UCP (productivity = 20 and complexity =4) is 41,693 person-hours. Although the size limitation varies based on the values of other predictors (productivity and complexity), it is not recommended to use the proposed Treeboost model to test projects of size more than 1,000 UCP.

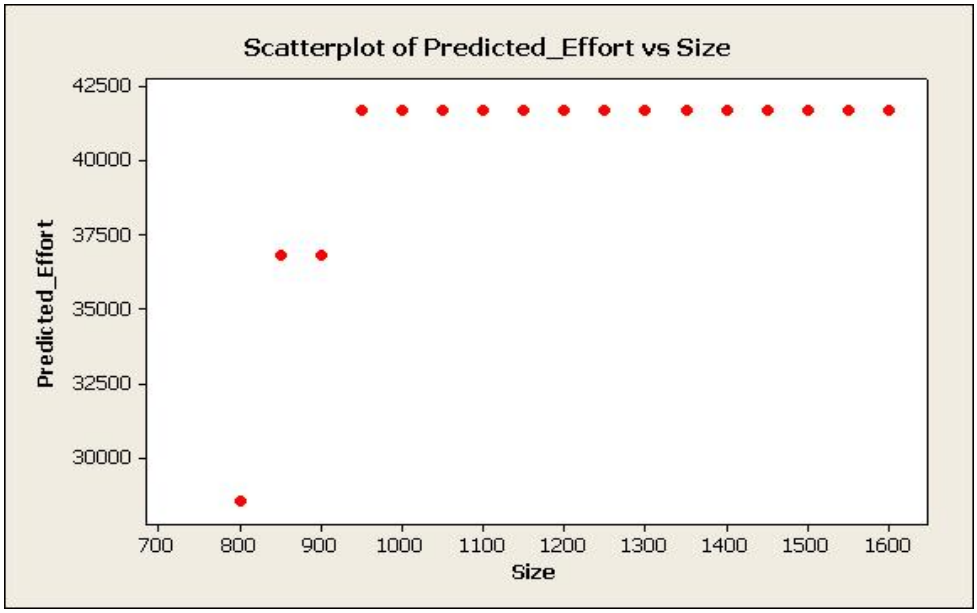


Figure 5-8 Scatterplot of size/predicted_effort

5.9 Conclusions

This chapter proposed a Treeboost model to predict software effort based on three independent variables which include software size, productivity and complexity. The Treeboost model was developed through a series of 359 trees and was trained using 168

data points. The model was evaluated using 69 data points against the UCP, as well as a multiple linear regression model. The evaluation criteria used were MMRE, MMER, PRED and MAE. The proposed model is limited to projects of size around 1,000 UCP (around 40,000 person-hours). Results showed that the Treeboost model outperformed the multiple linear regression model in all evaluation criteria and surpassed the UCP model when MMER, PRED and MAE were used. Based on these results, we conclude that the Treeboost model can be used for software effort estimation and can compete with other regression models.

References

[1] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Annals of Mathematical Statistics,* vol. 29, pp. 1189-1232, 2001.

[2] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics & Data Analysis,* vol. 38, pp. 367-378, 2002.

[3] J. N. Morgan and J. A. Sonquist, "Problems in the Analysis of Survey Data, and a Proposal," *Journal of the American Statistical Association,* vol. 58, pp. pp. 415-434, 1963.

[4] G. V. Kass, "An Exploratory Technique for Investigating Large Quantities of Categorical Data," *Journal of the Royal Statistical Society,* vol. 29, pp. 119-127, 1980.

[5] J. N. Morgan and R. C. Messenger, *THAID, a Sequential Analysis Program for the Analysis of Nominal Scale Dependent Variables.* Ann Arbor: Survey Research Center, Institute for Social Research, University of Michigan, 1973.

[6] P. Sherrod, "DTREG," *Software for Predictive Modeling and Forecasting,* 2011.

[7] M. Kearns, "Thoughts on Hypothesis Boosting," *Machine Learning Class Project,* 1988.

[8] P. J. Huber, "Robust Estimation of a Location Parameter," *Annals of Mathematical Statistics,* vol. 35, pp. 73-101, 1964.

[9] R. Lawrence, A. Bunn, S. Powell and M. Zambon, "Classification of remotely sensed imagery using stochastic gradient boosting as a refinement of classification tree analysis," *Remote Sensing of Environment,* vol. 90, pp. 331-336, 2004.

[10] M. Cappo, G. Deâ€™ath, S. Boyle, J. Aumend, R. Olbrich, F. Hoedt, C. Perna and G. Brunskill, "Development of a robust classifier of freshwater residence in barramundi (Lates calcarifer) life histories using elemental ratios in scales and boosted regression trees," *Marine and Freshwater Research,* vol. 56, pp. 713-723, 07/25, 2005.

[11] J. M. MatÃas, A. Vaamonde, J. Taboada and W. GonzÃ¡lez-Manteiga, "Support vector machines and gradient boosting for graphical estimation of a slate deposit," *Stochastic Environmental Research and Risk Assessment,* vol. 18, pp. 309-323, 2004.

[12] K. D. Shepherd, C. A. Palm, C. N. Gachengo and B. Vanlauwe, "Rapid Characterization of Organic Resource Quality for Soil and Livestock Management in Tropical Agroecosystems Using Near-Infrared Spectroscopy," *Agronomy Journal,* vol. 95, pp. 1314-1322, 2003.

[13] M. O. Elish, "Improved estimation of software project effort using multiple additive regression trees," *Expert Systems with Applications,* vol. 36, pp. 10774-10778, 9, 2009.

Chapter 6

6. Summary and Future Work

Each chapter has its own conclusions. This chapter summarizes the entire thesis and presents research avenues for the future work. In this thesis, we proposed an innovative model to predict software size and effort from use case diagrams. The main model is composed of six independent sub-models. These sub-models include linear regression with a logarithmic transformation, non-linear regression, Multilayer Perceptron (MLP) neural network model, Radial Basis Function Neural Network (RBFNN), General Regression Neural Network (GRNN) and Treeboost. There are four main inputs to our model. These include software size, productivity, complexity and requirements uncertainty.

In Chapter 1, we introduced the motivation of our work and put forward several research questions. The main motivation of our work was to develop a model to predict software effort that can be used in the early stages of the software life cycle with a good level of accuracy. The main research questions were concerned with how a project can be estimated using cases diagrams and what the influence of non-functional requirements is on software estimation. Additionally, we inquired about the type of models that can be used to estimate software effort.

The second chapter provides the definition of the most commonly used terms in the thesis. These include fuzzy logic, neural networks, regression analysis, evaluation criteria used in this work (MMRE, MMER, PRED, CI and MAE). As well, a literature review and related work were also presented.

Chapter 3 proposed a linear regression with logarithmic transformation, as well as an MLP model. The inputs of the MLP were software size and team productivity. Team productivity factor was calculated based on eight factors as shown in Table (2-6). These factors include "familiar with objectory" (IBM Rational Unified Process), "object oriented experience", "analyst capability", "stable requirements", "application experience", "motivation", "part-time workers" and "difficult programming language". We also demonstrated that the relationship between software effort and size is not linear based on 125 projects that have similar productivity values. We compared our model against two other models, namely, the Use Case Point (UCP) and Schneider's model. We chose to compare our model with these two models because these models predict software effort from use case diagrams. The project dataset was divided into two main parts; Small that contains projects of sizes less than 100 UCP and Large that contains projects of sizes greater than 100 UCP. We conducted three experiments to evaluate our model. In the first experiment, the whole dataset was used. In the second and third experiments, the Small and Large parts were used, respectively. Results indicated that our models outperform the UCP and Schneider's models in all experiments using the MMER and PRED criteria. As a comparison between the regression and the MLP model, the

MLP model gave better results when the Small part of the dataset was used. However, the MLP model deteriorated when the Large part was used. In this chapter, the second, sixth and seventh research questions were addressed.

In Chapter 4, we proposed linear and non-linear regression models, as well as RBFNN and GRNN. New methods to calculate software size, productivity, complexity and requirements uncertainty level were also introduced. The main difference between Chapter 4 and Chapter 3 is that in Chapter 4 we introduced "requirements stability" as an independent factor that affects software effort estimation. In Chapter 3, "requirements stability" was one of eight factors that contributed to productivity. Another main difference between these two chapters is that in chapter 3, the size of the projects used is relatively small. In Chapter 4, we used industrial projects of efforts ranging between 120 person-hours and 224,890 person-hours. These projects were not available at the time when the experiments of chapter 3 were conducted. In Chapter 4, we also carried out a thorough comparison among the models. We evaluated the models based on four different experiments. In the first experiment, the entire project dataset was used for evaluation. Then, we divided our dataset into three main parts. These include Small, Medium and Large sized projects. Results show that our model surpasses alternative models based on the four experiments. We used four different criteria for evaluation. These include MMRE, MMER, PRED and CI. As a comparison among our four sub-models, the non-linear regression outperformed all models; however, the GRNN model

surpassed the RBFNN model. Chapter 4 tackles all the research questions raised in Section 1.2.

The fifth chapter presents a Treeboost model to predict software effort based on three independent variables. These include software size, productivity and complexity. The model was trained using 168 projects. The Treeboost model was evaluated against a multiple linear regression model as well as the UCP model based on four different criteria which include MMRE, MMER, PRED and MAE. Experiments showed that the Treeboost model surpasses the other two models and can be used to predict software effort. The main advantage of the Treeboost model is that it is not sensitive to outliers in the training process as other neural network and regression models are. The main disadvantage of this model is that all testing data points should fall between the smallest and the largest data points used in training the model. In this chapter, the second, fourth and seventh research questions were tackled.

Each of the six sub-models has its own characteristics. Although the six sub-models can be used for effort estimation, the performance of each model varies based on the size and quality of training and testing data points. Table (6-1) lists the features and the applicability of each sub-model.

Table 6-1 Model features and applicability

Model type	Features and applicability
Linear regression with logarithmic transformation	• Good results with projects whose efforts are less than 3,000 person-hours (PH) • Acceptable results with projects whose efforts are between 3,000 and 150,000 PH • Regression analysis is based on the ordinary least squares method. This means the model is sensitive if training data contain outliers • Testing data points that are slightly beyond the training data points can be used as model's inputs
Non-linear regression	• Acceptable results (when the polynomial equation is used) with projects whose efforts are less than 3,000 PH • Good results (when the exponential models are used) with projects whose efforts are between 3,000 and 150,000 PH • Testing data points that are slightly beyond the training data points can be used as model inputs only if the model type is polynomial • Not recommended to use testing data points that are slightly beyond the training data points when the model is of the exponential type
MLP	• Very good results with projects whose efforts are less than 3,000 PH • Not recommended to estimate projects whose efforts are greater than 3,000 PH • Sensitive to outliers
RBFNN	• Good results with projects whose efforts are less than 3,000 PH • Acceptable results with projects whose efforts are between 3,000 and 150,000 PH • Recommended to re-train the model if the number of the training data points is more than 500 data points
GRNN	• Acceptable results with projects whose efforts are less than 3,000 PH • Good results with projects whose efforts are between 3,000 and 150,000 PH • Not recommended to re-train the model if the number of the training data points is more than 500 data points • More robust than MLP and RBFNN • Less sensitive to outliers
Treeboost	• Good results with projects whose efforts are less than 40,000 PH • Highly recommended if the training data points might contain outliers • Not recommended to re-train the model with new training data points if the range between two consecutive data points is big, or if the standard deviation of the new data points is high • Absolutely inappropriate to predict the effort of projects that are beyond the training data points

6.1 Future Work

One of the limitations of our work is the scarcity of the projects available to train and test the model. Published datasets such as NASA[1], PROMISE [2], COCOMO [3], CeBASE [4], Experience [5], Desharnais [6] and Maxwell [7] do not include information about use case diagrams. Future work will focus on:

1- Contacting more companies to collect data based on the questionnaire proposed in Appendix B.
2- Re-train all models when new data are available. The weight of the new data during model recalibration will be determined based on their source and importance.
3- Developing hybrid models between neural networks and evolutionary algorithms, such as genetic algorithms and particle swarm. It is believed that genetic algorithms can be used to train neural networks, and would thus increase the accuracy of the model.

References

[1] NASA datasets. [Online]. Available: http://data.giss.nasa.gov/.

[2] PROMISE datasets. [Online]. Available: http://www.promisedata.org.

[3] COCOMO datasets. [Online]. Available: http://promisedata.org/?p=6.

[4] CeBASE datasets. [Online]. Available: http://www.cebase.com.

[5] Experience datasets. [Online]. Available: http://www.fisma.fi.

[6] Desharnais datasets. [Online]. Available: http://www.promisedata.org/?p=9.

[7] Maxwell datasets. Available: http://www.promisedata.org/?p=108.

Appendix A

Questionnaire I

1- What is the name of the project?
2- What is the number of people involved in this project?
3- What is the actual effort this project? (if you can break down the work per each stage of the software life cycle, this would be preferable.)
4- Based on the use case diagram, what is the number of "simple use cases", "average use cases" and "complex use cases" based on the definition below (including the "extend", "include", and "generalized" use cases).
   - A use case is rated as "Simple" if the number of transactions in the use case scenario (Including both the Success and Extensions scenario) is less than or equal 4. (check the example below to see how transactions are counted).
   - A use case is rated as "Average" if the number of transactions in the use case scenario (Including both the Success and Extensions scenario) is between 4 and 7.
   - A use case is rated as "Complex" if the number of transactions in the use case scenario (Including both the Success and Extensions scenario) is more than 7.
5- What is the programming language used in the project?
6- Please rate these factors from "1" which represents "very low" to "5" which represents "very high". Factors with average classifications are rated as "3".

Factor	**Rate**
Familiar with Objectory	
Object oriented experience	
Analyst capability	
Stable requirements	
Application experience	
Motivation	
Part-time workers	
Difficult programming language	

7- Please rate these factors from "1" which represents "very low" to "5" which represents "very high". Factors with average classifications are rated as "3".

Factor	Rate
Easy installation	
Portability	
End user efficiency	
Reusability	
Complex internal processing	
Special security features	
Usability	
Application performance	
Special user training facilities	
Concurrency	
Distributed systems	
Provide direct access for third	
Changeability	

Example of a use case scenario (description):

The following example introduces the scenario of the use case "Student Enrolls in a Course" in a University Online Registration System.

Use Case Title: Student Enrolls in a Course
Actors: Student, Admin
Precondition: The student is not enrolled in a course
Main Success Scenario (Main Flow):

1. The student chooses the course he or she wishes to enroll in
2. The student enrolls in the course

Extensions (Alternative)

2a: The student does not have permission (e.g. the student has not paid the tuition)
2a1: Notify the student to contact the administrator
2b: The deadline has passed
2b1: An Error message will be displayed
2c: The prerequisite of the course is not fulfilled
2c1: The student is advised to contact the professor to obtain permission
2d: Two courses have the same schedule
2d1: The student is advised to choose either one
2e: The number of the enrolled courses has been exceeded

2e1: An error message will be displayed

2f: The course is full

2f1: An error message will be displayed

Post condition: The student has enrolled in a course

Appendix B

Questionnaire II

1- What is the name of the project?
2- What is the number of students involved in this project?
3- How many hours did each student work to finish this project? (if you can break down the work per each stage of the software life cycle, this would be preferable. If you cannot, just put the total number of hours).
4- Based on the use case diagram, what is the number of the use cases (including the "extend", "include", and "generalized" use cases).
5- What is the number of transactions in the success scenario of **each use case**? (check the example below).
6- What is the number of transactions in the Extension (exception) part of the scenario of **each use case**?
   (for example, if your use case diagram contains 20 use cases, you can name them as U1, U2, U3, … U20. For each use case, write the number of transactions in the success scenario as well as in the Extension part).
7- What is the programming language used in the project?
8- What is the complexity level of the project based on this definition:
   - Level-1: The complexity of a project is classified as Level-1 if the project team is familiar with this type of project and the team has developed similar projects in the past. The number and type of interfaces are simple. The project will be installed in normal conditions where high security or safety factors are not required. Moreover, Level-1 projects are those of which around 20% of their design or implementation parts are reusable (came from old similar projects).
   - Level-2: This is similar to level-1 category except that only about 10% of these projects are reusable.
   - Level-3: This is the normal complexity level where projects are not said to be simple, nor complex. In this level, the technology, interface, installation conditions are normal. Furthermore, no parts of the projects had been previously designed or implemented.
   - Level-4: In this level, the project is required to be installed on a complicated topology/architecture such as distributed systems. Moreover, in this level, the number of variables and interface is large.

- Level-5: This is similar to Level-4 but with additional constraints, such as a special type of security or high safety factors.

9- Please rate these factors from "1" which represents "very low" to "5" which represents "very high". Factors with average classifications are rated as "3".
   - Team experience regarding the problem domain.
   - Team motivation.
   - Programming language experience.
   - Object oriented experience (UML).
   - Analytical skills.

10- Please rate the Requirements Stability degree of your project from Level-1 (unstable requirements) to Level-5 (stable requirements).
   - Level-1: This indicates that there is an increase of 40% of the requirements during the project life cycle. This incorporates new requirements and changes in existing requirements.
   - Level-2: This indicates that there is an increase of 30% of the requirements during the project life cycle.
   - Level-3: This indicates that there is an increase of 20% of the requirements during the project life cycle.
   - Level-4: This indicates that there is an increase of 10% of the requirements during the project life cycle.
   - Level-5: This indicates that the requirements are stable during the project life cycle.

Example of a use case scenario (description):

Use Case Title: Student Enrolls in a Course
Actors: Student, Admin
Precondition: The student is not enrolled in a course
Main Success Scenario (Main Flow):
1. The student chooses the course he or she wishes to enroll in
2. The student enrolls in the course

Extensions (Alternative)
2a: The student does not have permission (e.g. the student has not paid the tuition)
2a1: Notify the student to contact the administrator

2b: The deadline has passed

2b1: An Error message will be displayed

2c: The prerequisite of the course is not fulfilled

2c1: The student is advised to contact the professor to obtain permission

2d: Two courses have the same schedule

2d1: The student is advised to choose either one

2e: The number of the enrolled courses has been exceeded

2e1: An error message will be displayed

2f: The course is full

2f1: An error message will be displayed

Post condition: The student has enrolled in a course

Appendix C

project #	size (ucp)	Effort (person-hour)	project #	size (ucp)	Effort (person-hour)
1	28	420	64	47	658
2	28	414.4	65	47	846
3	29	420.5	66	47	869.5
4	29	432.1	67	47	817.8
5	30	450	68	48	844.8
6	30	465	69	48	816
7	31	461.9	70	48	844.8
8	31	465	71	48	854.4
9	32	480.32	72	48	768
10	32	496	73	48	792
11	32	544	74	48	777.6
12	32	448	75	48	792
13	32	464	76	48	787.2
14	32	496	77	49	784
15	32	486.4	78	51	785.4
16	33	495	79	51	775.2
17	33	478.5	80	54	810
18	33	462	81	54	972
19	33	504.9	82	55	814
20	33	511.5	83	55	803
21	34	530.4	84	55	770
22	34	540.6	85	56	812
23	35	556.5	86	57	832.2
24	36	576	87	58	812
25	36	586.8	88	58	841
26	37	592	89	58	858.4
27	38	615.6	90	61	915
28	38	623.2	91	61	1342
29	39	647.4	92	62	868
30	39	631.8	93	63	894.6
31	40	660	94	66	957
32	41	688.8	95	66	924
33	41	697	96	66	976.8
34	41	656	97	69	966
35	41	664.2	98	71	958.5

36	41	672.4	99	71	979.8
37	41	697	100	74	1036
38	41	779	101	74	1110
39	41	615	102	74	1184
40	41	574	103	74	1332
41	42	631.26	104	80	1441.6
42	42	634.2	105	82	1492.4
43	42	642.6	106	84	1545.6
44	42	621.6	107	84	1520.4
45	42	625.8	108	85	1572.5
46	42	630	109	92	1720.4
47	42	617.4	110	92	1564
48	43	636.4	111	92	1582.4
49	43	638.55	112	94	1635.6
50	43	640.7	113	98	1862
51	43	645	114	98	1911
52	44	666.6	115	101	1616
53	44	671	116	105	1890
54	44	660	117	111	2109
55	45	697.5	118	118	1888
56	45	720	119	128	2432
57	45	742.5	120	145	3190
58	45	686.25	121	155	3875
59	46	736	122	212	4452
60	46	745.2	123	240	5760
61	46	782	124	280	7280
62	46	759	125	340	8160
63	47	893			

Appendix D

project #	Actual Effort (person-hour)	Size (UCP)	F1 (Familiar with Objectory) w=1.5	F2 (Stable requirements) w=2	F3 (Analyst capability) w=0.5	F4 (Application experience) w=0.5	F5 (Object oriented experience) w=1	F6 (Motivation) w=1	F7 (Difficult programming language) w=-1	F8 (Part-time workers) w=-1
1	2124	118	2	2	2	2	2	3	4	0
2	1430	130	4	3	3	4	4	3	3	0
3	1445	85	3	3	3	3	3	4	3	0
4	4895	275	4	4	4	5	4	4	2	0
5	2420	110	3	2	2	3	3	3	4	0
6	2080	65	2	2	2	2	2	3	3	4
7	1265	55	3	3	2	2	3	2	4	3
8	1240	40	2	2	2	3	2	4	4	4
9	1950	78	3	4	3	3	3	3	3	4
10	967	52	3	2	2	3	3	2	4	3
11	1664	128	4	4	4	4	4	2	3	0
12	3630	110	3	2	3	2	3	3	4	0
13	3915	145	4	4	4	4	4	4	2	0
14	1553	135	2	1	3	1	2	3	3	0
15	1440	90	3	3	3	3	3	3	3	0
16	1334	58	1	3	1	1	1	4	4	5
17	1617	98	4	4	4	4	4	4	3	0
18	2875	125	1	2	3	3	1	4	4	0
19	1984	64	1	1	2	2	1	2	4	5
20	1050	75	4	4	4	4	4	4	3	4

21	1050	75	4	4	3	3	4	2	2	0
22	1218	84	3	3	5	4	3	5	2	0
23	2465	145	3	2	3	4	3	5	2	0
24	3875	155	2	3	3	3	2	2	4	0
25	1116	62	3	4	2	2	3	2	3	4
26	7952	284	4	4	4	3	4	4	3	0
27	2697	87	2	1	2	2	2	2	4	5
28	696	58	2	2	2	2	2	3	3	5
29	3248	112	3	2	2	2	3	3	4	4
30	4338	241	5	4	4	4	5	4	2	0
31	5040	210	5	3	3	3	5	3	3	1
32	6292	286	3	3	3	4	3	3	3	0
33	2871	87	2	3	4	3	2	3	3	5
34	2754	102	4	3	4	3	4	3	2	5
35	2736	114	3	3	5	4	3	4	4	5
36	2212	79	3	2	3	3	3	4	4	5
37	1512	84	2	4	2	3	2	4	4	5
38	2064	86	2	1	3	3	2	4	4	5
39	2772	154	3	3	2	4	3	3	3	4
40	3828	174	3	2	3	4	3	3	3	0
41	3213	189	3	4	2	5	3	3	4	0
42	3666	141	3	3	2	2	3	3	2	3
43	2904	132	4	3	4	2	4	4	2	3
44	2880	120	4	2	4	2	4	4	4	4
45	2058	98	5	2	3	3	5	4	3	4
46	3096	129	3	2	4	4	3	4	2	1
47	2384	149	3	4	3	3	3	3	2	0
48	3528	196	4	5	3	3	4	3	3	0
49	4992	208	4	4	4	2	4	3	3	0
50	4165	245	4	3	3	3	4	3	2	0
51	2646	147	4	3	4	4	4	4	4	1
52	4450	178	4	3	4	3	4	4	3	1
53	1392	58	2	3	3	3	2	2	2	5
54	1776	74	2	2	3	3	2	2	1	5
55	2156	98	3	4	4	4	3	3	4	5
56	1976	104	4	4	4	4	4	3	3	5
57	1496	68	4	3	4	4	4	3	3	5
58	2162	94	4	3	4	2	4	3	4	5
59	2832	118	3	3	4	2	3	3	2	5
60	2850	114	3	3	3	2	3	4	2	5

61	1794	69	3	2	3	1	3	3	2	5
62	2688	84	2	2	3	1	2	3	2	5
63	4032	168	2	2	3	3	2	3	1	0
64	4536	189	2	3	4	4	2	3	1	0
65	3915	174	3	4	4	3	3	3	4	0
66	4708	214	4	3	4	2	4	3	3	0
67	6993	259	5	3	2	4	5	3	2	0
68	3864	168	2	3	2	3	2	3	2	2
69	4848	202	3	4	3	1	3	3	4	1
70	3654	174	4	4	3	3	4	3	2	1
71	4368	168	1	3	3	4	1	3	2	1
72	3128	184	4	3	4	4	4	3	1	1
73	3485	198	5	2	4	3	5	3	1	0
74	6604	254	3	3	4	4	3	3	2	0
75	5568	232	3	4	4	2	3	3	1	0
76	1044	58	2	2	3	2	2	3	3	5
77	2340	78	2	2	4	3	2	3	3	5
78	2444	94	1	3	4	3	1	4	3	5
79	1482	78	3	3	3	3	3	4	4	5
80	1443	74	3	3	3	4	3	4	4	5
81	1365	65	3	3	3	1	3	3	3	5
82	2156	98	4	3	4	2	4	3	3	1
83	2162	94	4	4	4	2	4	3	3	0
84	1258	74	4	4	4	3	4	4	3	0
85	1173	69	5	4	4	3	5	4	4	0
86	1098	61	3	4	5	4	3	2	3	0
87	1428	84	3	3	3	4	3	2	3	1
88	1584	88	3	3	3	3	3	3	3	1
89	1584	88	3	3	3	3	3	3	2	4
90	1656	72	3	2	2	3	3	3	1	4
91	2832	118	4	2	2	3	4	3	4	5
92	2256	94	4	3	3	2	4	3	4	5
93	2716	97	2	1	2	2	2	3	3	5
94	1768	68	2	1	3	3	2	4	3	5
95	1998	74	2	2	2	1	2	4	3	2
96	1985	81	3	3	3	2	3	3	4	2
97	1955	85	3	3	3	4	3	3	3	0
98	1840	80	3	2	4	3	3	3	3	0
99	3784	172	4	3	2	3	4	4	4	1
100	1932	84	2	3	4	3	2	3	4	5

101	1632	96	3	3	3	4	3	4	4	0
102	1666	98	3	3	3	4	3	4	4	0
103	1653	87	4	4	3	4	4	4	4	0
104	1472	64	2	4	4	5	2	4	2	0
105	1276	58	4	4	4	3	4	3	2	5
106	2162	94	4	3	3	3	4	3	3	5
107	2064	86	4	3	2	3	4	3	3	5
108	1454	57	3	3	4	2	3	5	3	5
109	1911	91	3	2	3	2	3	5	3	1
110	1110	74	3	2	3	4	3	4	3	1
111	3615	241	3	4	3	4	3	4	4	0
112	2632	188	2	3	3	3	2	4	4	0
113	3472	124	2	3	4	3	2	4	3	0
114	1734	102	5	2	2	4	5	3	3	5
115	2668	116	3	2	3	3	3	3	2	5
116	2832	118	3	3	3	3	3	3	4	5
117	4680	156	1	3	3	2	1	4	4	5
118	3060	170	4	2	3	3	4	4	1	5
119	6072	264	4	2	3	3	4	2	4	0
120	4081	154	3	1	3	3	3	4	3	2

Appendix E

Project_number	Compexity value	Productivity value	Requirements Stability value	Size	Effort
1	1	1.3	1	13.5	122
2	1	1.15	1	18	296
3	0.7	1	1.2	20.5	360
4	0.7	1.15	1	28	170
5	0.85	1.15	1	39	507
6	1	1.15	1	41.5	634
7	1	1	1	47	752
8	1	1	1	47.5	751
9	0.7	1.15	1	51	244
10	1	1	1	52	843
11	0.85	1	1.1	53	948
12	0.85	1	1	53.5	809
13	1	1	1	58	870
14	1	1.15	1	61	902
15	1	1	1.1	63	1022
16	0.85	1	1	64	1024
17	0.85	1	1	65.5	1049
18	1	1	1	68	1212
19	1	1.15	1	70	1228
20	1	1.15	1	72	1209.6
21	1	1	1	75.5	1400
22	1	1.15	1	78	1216
23	1	1	1	80	1440
24	1	1	1.1	88	1613
25	1	1.3	1	90	1313
26	1	1.15	1	93	1550

27	1	1.3	1	104.5	1280
28	1	1	1	106.5	1983
29	1	1	1	111	2121
30	1	1.3	1	112	1702
31	1	1	1	115	2530
32	1	1	1	117	2640
33	1	1	1	123.5	2535
34	1	1.3	1	124.5	2020
35	0.85	1.3	1	131.5	1635
36	0.7	1.15	1	145.5	1926
37	1	0.7	1	150	4648
38	1	0.7	1	173	4498
39	1	1	1	192	3840
40	1.3	1	1	192	4992
41	1	1.15	1	197.5	3698
42	0.85	0.85	1	216.5	4198
43	1	1	1	226.5	7823
44	1	1	1.1	275	11580
45	0.7	1.3	1	286.5	1821
46	1	1	1.1	290	7224
47	1.3	1	1	293	8497
48	1	1	1	302	8298
49	1	1	1	313	8413
50	1	1	1	341	9507
51	1	1	1	357	10167
52	1	1	1	388.5	12606
53	1	1	1	407	13789
54	1	1	1	409	12449
55	1	1	1	472	16350
56	1	1	1	498	17848
57	1	1	1	552	17906
58	1	1	1	612	22491
59	1	1	1	619	19529
60	1	1	1	840	31542
61	1	1	1	967	33409
62	1.15	1.15	1	986.5	37723
63	1.15	1	1	1412	57044
64	1	0.7	1	1780	78693
65	1.3	1	1.1	2455	129353

Appendix F

66	1.15	0.7	1.4	5.5	167
67	1	0.7	1.4	10	278
68	0.85	1.15	1.1	17	374
69	1.15	1	1.2	18	368
70	0.85	1.15	1.1	25.5	664
71	0.85	1.15	1.1	31	626
72	1.3	0.7	1.4	31.5	1224
73	1.15	1	1.3	33.5	1280
74	1.3	1	1.3	36.5	1124
75	1.3	0.7	1.4	37	988
76	0.85	1.15	1.1	40	817
77	0.85	1.15	1	41.5	887
78	0.7	1.15	1.2	47.5	1078
79	1	1.15	1.2	47.5	1449
80	1.15	1	1.3	50.5	1586
81	1	1.15	1.3	53.5	1824
82	0.85	1.15	1.1	54	972
83	0.85	1.3	1.3	56	890
84	1.3	1	1.3	56.5	1608
85	1	1.15	1.1	58	1328
86	1.15	1	1.4	58.5	2158
87	1.15	1	1.4	59.5	2248
88	0.85	1.15	1.1	61	1278
89	0.85	1.15	1.1	63	1733
90	1	1.15	1.2	64	1860
91	1	1.15	1.1	68	1074.4
92	1.3	0.7	1.4	71	2244
93	0.85	1.15	1.2	71.5	1821
94	1	1.15	1.1	76	2964
95	0.85	1.15	1.2	77	2009
96	1.15	1	1.3	82	2965
97	1	1.3	1.2	92	1840
98	0.85	1.15	1.2	96	2264
99	1.15	1.15	1.1	96.5	2380

100	1.3	1	1.3	102.5	3240
101	0.85	1.3	1	115	1824
102	1.15	1	1.3	117	3890
103	0.85	1	1.1	123.5	3480
104	1.15	0.7	1.4	126.5	6645
105	0.7	1.3	1.1	127.5	4190
106	0.85	1	1.3	129.5	3480
107	0.85	1.15	1.1	134	2933
108	1	1.15	1.3	135	3430
109	1.3	1	1.3	147	5480
110	0.85	1.15	1.2	156.5	3147
111	1.15	1	1.4	163	6480
112	1	1.15	1.2	172	5963
113	0.7	1.3	1	180.5	3480
114	0.7	1.3	1.3	192.5	4800
115	1.15	1	1.4	196.5	5445
116	0.7	1.3	1	197	3660
117	1	1.15	1.2	198.5	5882
118	1.15	1	1.3	205.5	6810
119	1.3	1	1.3	210.5	7050
120	1	1.3	1.1	215	5760
121	1.3	1	1.2	245.5	7845
122	0.85	1.3	1	248	8340
123	1.3	1.3	1.2	260	8960
124	0.7	1.3	1	270.5	5210
125	0.85	1.15	1.1	282.5	5709
126	0.7	1.3	1.1	286.5	1904
127	1	1.15	1.3	311	11818
128	0.85	1.15	1.2	320	10240
129	1	1.15	1.2	322	11270
130	0.7	1.3	1.1	329.5	7880
131	1.3	1.15	1.1	335	12730
132	1	1.15	1.1	341	10912
133	0.85	1.3	1	350	7872
134	0.7	1.3	1	354	7234
135	1.15	0.7	1	390	13260
136	0.85	1.3	1.1	407	9930.8
137	0.7	1.3	1.1	433	12410
138	0.85	1.15	1	441.5	10004
139	1.3	1	1.3	455	20020

140	1.3	1	1.3	496.5	22110
141	1	1.3	1.1	508	22352
142	0.85	1.15	1.1	525	11022
143	1.3	1	1.3	554	26940
144	1	1.3	1.2	585	12916
145	1.15	1.15	1	660.5	16600
146	0.7	1.3	1	707	16845
147	0.85	1	1	1060	24192
148	1.15	1.3	1.1	1830	49536
149	1	1.15	1.3	4010	198840
150	1.3	0.7	1.4	6	378
151	1.3	0.7	1.4	13	397
152	0.85	1.3	1	17	120
153	1.3	0.7	1.3	18	400
154	1.3	0.7	1.4	23.5	838
155	1.3	0.7	1.4	25.5	760
156	0.85	1.15	1.1	32	724
157	1.3	0.7	1.4	39	1153
158	1.3	0.85	1.1	45	957
159	1.3	0.7	1.4	48.5	3323
160	1.3	0.7	1.3	53	2002
161	1.15	0.7	1.1	54.5	1090
162	1.3	0.7	1.3	56	2134
163	1.3	0.7	1.3	58	2175
164	1.3	0.85	1.2	59.5	1877
165	1.3	0.7	1	60	1400
166	1.3	0.7	1.1	63.5	1536
167	1.3	0.7	1.4	68	1820
168	1	0.85	1.2	68.5	1583
169	1.3	0.7	1.4	71.5	1880
170	1	0.7	1.2	73	1972
171	1	0.7	1.1	76.5	1882
172	1.3	0.7	1.4	77.5	1052
173	1.3	0.7	1.4	86	4108
174	1.15	1.15	1.3	94	2080
175	1.3	1	1.2	101	7602
176	1.15	1	1.2	104	4209
177	1.3	1	1.3	117.5	5374
178	1	1.15	1.2	124.5	4551
179	1	1.15	1.2	127	4651

180	1	1.15	1.2	130	4184
181	1.15	1	1.4	137	6910
182	0.85	1.15	1.2	167.5	4879
183	1.3	0.7	1.3	168.5	11680
184	0.85	1.3	1	180	1705
185	1.3	1	1.3	187	13288
186	1.15	1	1.2	196	15729
187	1	1.15	1.2	198	6051
188	1.15	1	1.2	227.5	9301
189	0.85	1.15	1.2	234	6552
190	1	1	1.3	253	11749
191	0.85	1.3	1.1	256	3664
192	0.85	1.3	1.1	264	3244
193	0.85	1.3	1	268	2978
194	0.85	1.3	1	274	3153
195	0.85	1.15	1.2	293	8790
196	0.7	1.3	1	310.5	6220
197	1	1	1.2	317	11095
198	0.7	1.3	1	324.5	5280
199	0.7	1.3	1.1	338.5	8100
200	0.85	1.15	1.1	349.5	8060
201	0.7	1.15	1.1	388	9312
202	0.7	1.3	1	412.5	7820
203	1	1.15	1.1	426	17892
204	1.15	1	1.3	436.5	20389
205	0.7	1.3	1	449	8180
206	1.15	1	1.3	509	60826
207	0.7	1.3	1	576.5	16532
208	0.7	1.3	1	737.5	19820
209	1.3	1.15	1	760	30912
210	0.85	1.3	1.1	878.5	27800
211	0.7	1.15	1.1	910	32800
212	0.7	1.3	1	3070	89030
213	0.7	1.3	1	3860	188340
214	0.7	1.3	1	3980	224890

Appendix G

RBFNN Parameters

Neuron	Bias	Weight	Center_size	Width_size	Center_prod	Width_prod	Center_complex	Width_complex	Center_Req	Width_Req
1	9700.5	205163.3	14.09376	57.96999	-3.46125	268.984	0.236544	379.5427	0.655411	193.2344
2	9700.5	40.8244	-0.30085	9.86213	-3.67838	149.6633	-0.06608	0.107198	1.91701	390.2638
3	9700.5	1296.638	1.717095	0.26561	0.45686	344.4288	-0.24646	141.6568	0.257214	31.21814
4	9700.5	15834.48	0.196781	329.2464	-0.85419	300.5105	-0.0693	204.6251	0.581591	0.060439
5	9700.5	-74059.4	14.56015	144.2844	-1.60751	151.2063	-0.05865	67.63445	0.532074	0.154067
6	9700.5	3705.166	1.224678	1.011923	0.945957	333.8084	0.118394	355.9286	1.761941	296.1721
7	9700.5	-35516.6	-0.05855	10.67504	-2.22027	179.6136	-0.20381	380.3062	1.648088	137.9463
8	9700.5	23263.67	0.599317	10.90981	-3.21756	167.0027	0.294767	95.94758	0.20181	21.36284
9	9700.5	3509.583	6.371061	234.4714	0.41795	192.6596	0.079526	199.0717	1.661398	1.767917

Appendix H

Trees	Validation Absolute Error	Trees	Validation Absolute Error	Trees	Validation Absolute Error	Trees	Validation Absolute Error
10	5869.987	260	2308.749	500	2293.532	750	2309.085
20	5386.938	270	2287.525	510	2298.875	760	2305.311
30	5064.88	280	2280.722	520	2294.796	770	2303.961
40	4721.407	290	2277.771	530	2298.728	780	2303.618
50	4410.863	300	2269.83	540	2300.85	790	2301.491
60	4127.959	310	2272.116	550	2298.998	800	2300.617
70	3902.936	320	2271.898	560	2301.035	810	2300.517
80	3767.422	330	2273.275	570	2301.466	820	2301.233
90	3642.507	340	2267.082	580	2305.604	830	2299.708
100	3509.104	350	2265.949	590	2308.908	840	2300.222
110	3411.708	359	2260.936	600	2311.757	850	2303.442
120	3315.563	360	2261.781	610	2310.508	860	2304.086
130	3230.62	370	2264.029	620	2310.197	870	2303.419
140	3141.166	380	2266.435	630	2309.814	880	2303.328
150	3076.801	390	2267.845	640	2311.502	890	2301.861
160	3008.4	400	2272.518	650	2309.714	900	2302.601
170	2947.416	410	2273.739	660	2314.439	910	2301.998
180	2867.299	420	2275.863	670	2315.995	920	2302.819
190	2787.113	430	2274.62	680	2317.473	930	2303.39
200	2713.735	440	2273.57	690	2315.097	940	2302.268
210	2647.587	450	2274.487	700	2315.732	950	2300.764
220	2551.555	460	2280.963	710	2314.046	960	2300.61
230	2494.742	470	2279.811	720	2312.364	970	2299.892
240	2420.565	480	2282.187	730	2311.937	980	2301.007
250	2370.713	490	2285.307	740	2310.119	990	2301.796
						1000	2300.96

Curriculum Vitae and Thesis-Relevant Publications

Name:	Ali Bou Nassif
Post-Secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada Electrical and Computer Engineering 2009-2012 Ph.D (GPA 96.25%) The University of Western Ontario London, Ontario, Canada Computer Science 2007-2009 M.Sc (GPA 91%) Beirut Arab University Beirut, Lebanon Electrical Engineering 1992-1997 B.Eng
Honours and Awards:	Ontario Graduate Scholarship (OGS) The University of Western Ontario May 2011 – April 2012 Outstanding Presentation in Graduate Symposium (2011) The University of Western Ontario
Related Work Experience:	Teaching and Research Assistant The University of Western Ontario 2007-2012
Thesis-Related Publications:	**Journal Papers:**

1- **Ali Bou Nassif**, Luiz Fernando Capretz and Danny Ho, "Regression and Neural Network Models for Software Effort Estimation from Use Case Diagrams", *Empirical Software Engineering*, 2012 (Under review).

2- **Ali Bou Nassif**, Luiz Fernando Capretz, Danny Ho and Daniel Varona, "Software Effort Estimation from Use Case Diagrams Using Non-Linear Regression Analysis", *Journal of Systems and Software, Elsevier*, 2012 (Under review).

3- Wei Lin Du, Luiz Fernando Capretz, Danny Ho and **Ali Bou Nassif:** "A Hybrid Neuro-fuzzy SEER-SEM Model for Better Software Effort Estimation", *IEEE Computational Intelligence*, 2012 (Under review).

4- **Ali Bou Nassif,** Danny Ho and Luiz Fernando Capretz: Towards an Early Software Estimation Using Log-Linear Regression and A Multilayer Perceptron Model, *Journal of Systems and Software, Elsevier*, 2012 (Under Review).

5- **Ali Bou Nassif,** Luiz Fernando Capretz and Danny Ho: Enhancing Use Case Points Estimation Method using Soft Computing Techniques, *Journal of Global Research in Computer Science*, Volume 1, No. 4, November 2010, PP. 12-21 (Published).

Conference Papers:

6- **Ali Bou Nassif**, Luiz Fernando Capretz and Danny Ho, "Software Effort Prediction from Use Case Diagrams Using Multilayer Perceptron", *6th IEEE/ACM International Symposium on Empirical Software Engineering and Measurement* , 2012 (Under review)

7- **Ali Bou Nassif**, Luiz Fernando Capretz and Danny Ho, "Software Effort Estimation in the Early Stages of the Software Life Cycle Using a Cascade Correlation Neural Network Model", *13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2012 (Accepted)

8- **Ali Bou Nassif**, Danny Ho and Luiz Fernando Capretz, "A Treeboost Model for Software Effort Estimation Based on Three Independent Variables", *Predictive Models in Software Engineering*, 2012 (Under review).

9- **Ali Bou Nassif,** Luiz Fernando Capretz and Danny Ho: A Regression Model with Mamdani Fuzzy Inference System for Early Software Effort Estimation Based on Use Case Diagrams, *2011 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS 2011)*, November 2011, Guangzhou, China (Published)

10- **Ali Bou Nassif,** Luiz Fernando Capretz and Danny Ho: Estimating UML Size/Cost, *CASCON*, November 2011, Toronto, Ontario, Canada (Published).

11- **Ali Bou Nassif,** Luiz Fernando Capretz and Danny Ho: Estimating Software Effort Based on Use Case Point Model Using Sugeno Fuzzy Inference System, *23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2011),* November 2011, Boca Raton, Florida, USA (Published)

12- **Ali Bou Nassif,** Danny Ho and Luiz Fernando Capretz: Regression Model for Software Effort Estimation Based on the Use Case Point Model, *2011 International Conference on Computer and Software Modeling (ICCSM 2011),* September 2011, Singapore (Published)

13- **Ali Bou Nassif,** Luiz Fernando Capretz and Danny Ho: Software Estimation in the Early Stages of the Software Life Cycle, *International Conference on Emerging Trends in Computer Science, Communications and Information Technology (CSCIT 2010),* January 2010, Nanded, India (Published)