1-2013

Towards an Early Software Estimation Using Log-Linear Regression and a Multilayer Perceptron Model

Ali Bou Nassif
University of Western Ontario, abounas@uwo.ca

NFA-Estimation

Luiz Fernando Capretz
University of Western Ontario, lcapretz@uwo.ca

The Journal of Systems and Software 86 (2013) 144–160

Contents lists available at SciVerse ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Towards an early software estimation using log-linear regression and a multilayer perceptron model

Ali Bou Nassif[a,*], Danny Ho[b], Luiz Fernando Capretz[a]

[a] Department of ECE, Western University, London, Ontario, Canada
[b] NFA Estimation Inc., Richmond Hill, Ontario, Canada

ARTICLE INFO

Article history:
Received 26 October 2011
Received in revised form 30 June 2012
Accepted 14 July 2012
Available online 1 August 2012

Keywords:
Use case points
Log-linear regression model
Software effort estimation
Multilayer perceptron

ABSTRACT

Software estimation is a tedious and daunting task in project management and software development. Software estimators are notorious in predicting software effort and they have been struggling in the past decades to provide new models to enhance software estimation. The most critical and crucial part of software estimation is when estimation is required in the early stages of the software life cycle where the problem to be solved has not yet been completely revealed. This paper presents a novel log-linear regression model based on the use case point model (UCP) to calculate the software effort based on use case diagrams. A fuzzy logic approach is used to calibrate the productivity factor in the regression model. Moreover, a multilayer perceptron (MLP) neural network model was developed to predict software effort based on the software size and team productivity. Experiments show that the proposed approach outperforms the original UCP model. Furthermore, a comparison between the MLP and log-linear regression models was conducted based on the size of the projects. Results demonstrate that the MLP model can surpass the regression model when small projects are used, but the log-linear regression model gives better results when estimating larger projects.

1. Introduction

Software project failure is one of the main challenges in the software industry. During the last five decades, it was reported that the percentage of project failures and incomplete projects surpassed 30% (Eck et al., 2009; Lynch, 2009). According to the International Society of Parametric Analysis (ISPA) (Eck et al., 2009) and the Standish Group International (Lynch, 2009), the main reasons behind project failures are:

- Lack of estimation of the staff's skills and levels.
- Lack of understanding the requirements.
- Improper software size estimation.
- Uncertainty of system and software requirements.
- Optimism in software estimation.

In a nutshell, many software projects fail because of the inaccuracy of software estimation and misunderstanding or incompleteness of the requirements. This motivated researchers to investigate software estimation to yield better software size and effort assessment.

As software estimation became crucial to prevent or reduce project failures, estimation in the early stages of the software life cycle became imperative. The earlier the estimation is, the better project management will be. The importance of the early estimation reveals when it is required to bid on the project or commit to a contract between the customer and the developer. The early software estimation is conducted at a point when the details of the problem are not yet divulged. This is called the size estimation paradox (Demirors and Gencel, 2004). The software size should first be estimated in the early stages of the software life cycle, which is mainly the requirements stage. Several cost estimation techniques exist and they can be classified under three main categories (Mendes et al., 2002). These categories are:

1. *Expert judgment*: In this category, a project estimator tends to use his or her expertise which is based on historical data and similar projects to estimate software. 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 . . .*" (Jørgensen, 2007).
2. *Algorithmic models*: This is still the most popular category in literature (Briand and Wieczorek, 2002). These models include COCOMO (Boehm, 1981), SLIM (Putnam, 1978) and SEER-SEM (Galorath and Evans, 2006). The main cost driver of these mod-

* Corresponding author.
E-mail addresses: abounas@uwo.ca (A.B. Nassif), danny@nfa-estimation.com (D. Ho), lcapretz@uwo.ca (L.F. Capretz).

0164-1212/$ – see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jss.2012.07.050

els is the software size, usually the Source Lines of Code (SLOC). Algorithmic models either use a linear regression equation, like the one used by Kok et al. (1990) or non-linear regression equations, those which are used by Boehm (1981).

3. *Machine learning*: Recently, machine learning techniques are being used in conjunction or as alternatives to algorithmic models. These techniques include neural networks, fuzzy logic, neuro-fuzzy, Genetic Algorithm and regression trees. Machine learning models can incorporate historical data and can be trained to better predict software effort.

None of the above techniques are perfect and can fit all situations (Boehm et al., 2000a). In this paper, machine learning techniques (fuzzy logic and neural networks) are used with an algorithmic model (use case point model) for better software estimation results.

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 represents the functional requirements of a system and it is usually included in the Software Requirements Specification (SRS) documents.

The main purpose of this research is to propose a model to predict software effort from use case diagrams. Our model can be used in the early stages of the software life cycle. This is important for project managers who wish to conduct early cost estimation so that they can bid on projects. The accuracy of the proposed approach can surpass the original UCP model. In this work, a linear regression model with a logarithmic transformation (aka log-linear) is created to calculate software effort from use case diagrams. In this model, software effort is a function of software size and team productivity. The proposed model takes into consideration the non-linear relationship between software size and effort. The non-linear relationship is described in detail in Section 2. As shown in Eq. (17), software effort is directly proportional to software size and inversely proportional to team productivity (Galorath and Evans, 2006). A multiple linear regression equation was generated to predict the values of the productivity factor. Moreover, a Mamdani fuzzy logic approach (Mamdani, 1977) has been used to adjust the productivity factor. Furthermore, a MLP model was developed using the k-fold cross validation technique. A comparison was performed between the proposed log-linear regression model against the proposed MLP model as well as other models that conduct software effort estimation from use case diagrams. Experiments indicate that, among the existing data points, when small projects (<3000 person-hours) are used in the evaluation process, the MLP model exceeds other models. On the contrary, the regression model gives better results when projects of effort greater than 3000 person-hours are being used.

The remainder of the paper is organized as follows: Sections 1.1, 1.2, 1.3 and 1.4 present an overview of the use case point (UCP) model, evaluation criteria used in this paper, fuzzy logic and neural networks, respectively. Section 1.5 lists some related work. Sections 2 and 3 propose the novel regression and MLP models. Section 4 demonstrates an evaluation of the proposed models and provides some comparison among the models. Section 5 lists some threats to validity. Finally, Section 6 concludes the paper and proposes future work.

1.1. Use case points

The use case point model was first described by Karner (1993). This model is used for software cost estimation based on the use case diagrams. 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 1 and 2, respectively.

Table 1
Complexity weights of use cases (Karner, 1993).

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
Complexity weights of actors (Karner, 1993).

Actor complexity	Description	Weight
Simple	Through an API	1
Average	Through a text-based user interface	2
Complex	Through a graphical user interface	3

As shown in Table 1, 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 Eq. (1).

$$UUCW = \sum_{i=1}^{3} n_i \times W_i. \tag{1}$$

where n_i is the number of items of variety i of the use cases and W_i is the complexity weight of the corresponding use case. Similarly, UAW is represented as follows:

$$UAW = \sum_{j=1}^{3} m_j \times C_j. \tag{2}$$

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:

$$UUCP = UUCW + UAW. \tag{3}$$

After calculating the UUCP, the Adjusted Use Case Points (UCP) is calculated. UCP is achieved by multiplying UUCP by the technical factors (TFs) and the environmental factors (EFs). TF contributes to the complexity of the project while EF contributes to the team efficiency and productivity. The technical and environmental factors

Table 3
Technical factors (Karner, 1993).

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 objectives	1
T_9	Special user training facilities	1
T_{10}	Concurrency	1
T_{11}	Distributed systems	2
T_{12}	Provide direct access for third parties	1
T_{13}	Changeability	1

Table 4
Environmental factors (Karner, 1993).

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

are depicted in Tables 3 and 4, respectively. The technical factor is detailed as follows:

$$TF = 0.6 + 0.01\sum_{i=1}^{13} T_i \times W_i. \quad (4)$$

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). W_i represents the weight of technical factors (Table 3).

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

$$EF = 1.4 - 0.03\sum_{i=1}^{8} E_i \times W_i. \quad (5)$$

where E_i is the environmental factor (which is similar to T_i in Eq. (4), taking values between 0 and 5. Finally, the Adjusted Use Case Points (UCP) can be defined as follows:

$$UCP = UUCP \times TF \times EF. \quad (6)$$

By incorporating TF and EF, the value of UCP will be more or less than the value of UUCP by 30%. For effort estimation, the UCP model proposed 20 person-hours to develop each UCP. This is expressed in Eq. (7):

$$Effort = Size \times 20. \quad (7)$$

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

Schneider and Winters (2001) mentioned that when calculating software effort, instead of multiplying the size by 20, 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.

In this research, the Unadjusted Actor Weight (UAW) is neglected as suggested by Ochodek et al. (2011) since the estimation accuracy will not be affected.

1.2. Evaluation criteria

Several methods exist to evaluate cost estimation models. In this research, the evaluation criteria used include the Mean of Magnitude of Error Relative to the estimate (MMER), the Root Mean Squared Error (RMSE), the Mean Absolute Error (MAE), the standard deviation (SD) of the mean error and prediction level (PRED).

- MMER: MMER is one of the criteria used for cost estimation models evaluation (Kitchenham et al., 2001). Foss et al. (2003) argued that MMER can sometimes be more accurate than the Mean of the Magnitude of Relative Error (MMRE). MMER is the mean of MER as shown in Eqs. (8) and (9).

$$MER_i = \frac{|Actual\ \ Effort_i - Predicted\ Effort_i|}{Predicted\ \ Effort_i}. \quad (8)$$

$$MMER = \frac{1}{N}\sum_{1}^{N} MER_i. \quad (9)$$

- $PRED(x)$: PRED can be described as the average of the MRE's (or MER's) off by no more than x as defined by Jorgensen (1995):

$$PRED(x) = \frac{1}{N}\sum_{i=1}^{N} \begin{cases} 1 \text{ if } MER_i \le x \\ 0 \quad \text{otherwise} \end{cases}. \quad (10)$$

The estimation accuracy is directly proportional to $PRED(x)$ and inversely proportional to MMER.

- MAE: The Mean Absolute Error (MAE) is the average of the absolute errors between the actual and the predicted effort as shown in Eq. (11).

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |Actual\ Effort_i - Predicted\ Effort_i|. \quad (11)$$

- Standard deviation: The equation of the standard deviation can be seen as:

$$SD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}. \quad (12)$$

where x_i is the error of the observation "i" such that: $x_i = (Actual\ Effort_i - Predicted\ Effort_i)$ and $\bar{x}$ is the mean error for N observations.

- RMSE: The Root Mean Squared Error (RMSE) is the square root of the mean of the square of the differences between the actual and the predicted efforts as shown in Eq. (13).

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(E_{a_i} - E_{Pi})^2}{N}}. \quad (13)$$

where E_a and E_p are the actual and predicted efforts respectively, N is the number of observations.

1.3. Fuzzy logic

Fuzzy logic is derived from the fuzzy set theory that was proposed by Zadeh (1965). 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 ranging between 0 and 1. This means that in the binary logic, a member completely belongs to or does not belong 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:

$$F_z[x \in A] = \mu_A(x) : \mathbb{R} \to [0, 1]. \quad (14)$$

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 base 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 paper, fuzzy logic is used to calibrate the productivity factor of the regression model.

1.4. Neural network

An artificial neural network (ANN) is a network composed of artificial neurons or nodes which emulate the biological neurons (Lippman, 1987). 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:

$$y(t) = f\left[\sum_{i=1}^{n} w_i x_i - w_0\right]. \tag{15}$$

where x_i are neuron inputs, w_i are the weights and $f[.]$ is the activation function.

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. If the relationship between the input and output is not linear, one or more hidden layers should 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). A MLP contains at least one hidden layer and each input vector is represented by a neuron. 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. In this paper, an MLP model is used to predict software effort from use case diagrams.

1.5. Related work

Some issues related to the UCP model have been addressed in previous work. Authors in (Diev, 2006) and (Anda et al., 2001) worked on adjustment factors, while others in (Anda et al., 2001) and (Arnold and Pedross, 1998) highlighted the discrepancies in designing use case models. Researchers in (Robiolo and Orosco, 2008), (Robiolo et al., 2009) and (Ochodek and Nawrocki, 2008) proposed different size metrics such as transactions, TTPoints and paths, while others (Periyasamy and Ghode, 2009; Wang et al., 2009; Schneider and Winters, 2001; Braz and Vergilio, 2006; Nassif et al., 2011a,b; Mohagheghi et al., 2005; Ochodek et al., 2011) went further to extend the UCP model by providing new complexity weights or by modifying the method used to predict effort.

Regarding software effort prediction models based on machine leaning techniques, Azzeh et al. (2010) and Azzeh et al. (2011) 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. (2005) 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.

Papatheocharous et al. (2010) used feature selection technique on the Desharnais and ISBSG datasets to reduce the number of cost drivers that are used as inputs to cost prediction models without deteriorating the performance of the model. The experiments show that reducing the number of cost drivers leads to reducing the complexity, learning time and eliminating needless calculations.

Andreou and Papatheocharous (2008) used fuzzy decision trees to predict software effort. The authors used the decision trees algorithms CHAID and CART to evaluate their model using the ISBSG dataset. The evaluation criteria used include the Mean Relative Error (MRE), Normalized Root Mean Squared Error (NRMSE), and the correlation coefficient (CC). The experiments showed that the proposed approach can be used for software effort estimation at highly accurate levels.

Papatheocharous and Andreou (2007) used artificial neural networks (ANN) with Input Sensitivity Analysis (ISN) to develop software cost prediction model. Several ANN topologies were created and trained using the Desharnais and ISBSG datasets. The evaluation criteria used include NRMSE, CC, MSE, RMAE, MAE and PRED. The results show that the number of inputs of a cost prediction model can be reduced to a number between 3 and 5 to reduce the complexity of using many input parameters.

Huang and Chiu (2006) used Genetic Algorithm to determine the appropriate weighted similarity measures of effort drivers in analogy-based software effort estimation models. The authors used three weighted analogy methods. These include the unequally weighted, the linearly weighted and the nonlinearly weighted methods. The experiments were conducted using the ISBSG dataset and proved that the proposed approach can enhance the accuracy of software effort estimation.

Kumar et al. (2008) proposed a Wavelet Neural Network (WNN) for software cost prediction using Morlet and Gaussian functions as transfer functions. The proposed approach was evaluated using the Canadian Financial and IBM data processing services datasets using the MMRE criteria. The WNN model outperformed other models such as MLP, RBFNN, multiple linear regression, dynamic evolving neuro-fuzzy inference system and support vector machine.

de Barcelos Tronto et al. (2008) investigated a stepwise regression model and a neural network model for software cost prediction. The proposed models were evaluated using COCOMO'81 dataset using the MMRE and R^2 criteria. The results showed that the

proposed models can compete with other models such as COCOMO and SLIM.

Idri and Abran (2000) applied fuzzy logic on the COCOMO'81 model. Fuzzy sets with Trapezoidal membership function were defined for each cost driver. The evaluation was performed using the COCOMO'81 dataset using the RE and PRED. The results show that COCOMO'81 used with fuzzy logic tackles the imprecision caused by the crisp inputs (cost drivers) and generates more graduate output.

Ahmed et al. (2005) proposed an adaptive fuzzy logic framework for software effort prediction. This framework incorporates expert knowledge to improve the accuracy of software effort estimation. The framework was evaluated using an artificial datasets as well as the COCOMO database.

Jiang et al. (2007) and Xia et al. (2008) 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 of 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 and Baek (2008) 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 MRE criterion.

Huang et al. (2007) used a neuro-fuzzy approach to calibrate the parameters of the COCOMO model. The proposed model has some characteristics such as learning ability and good interpretability, while maintaining the merits of the COCOMO model. The model deals effectively with imprecise and uncertain input and enhances the reliability of software cost estimates. In addition, it allows input to have continuous rating values and linguistic values, thus avoiding the problem of similar projects having large different estimated costs. The results showed that PRED(20%) and PRED(30%) were improved by more than 15% and 11%, respectively in comparison with that of COCOMO 81.

Attarzadeh and Ow (2011) proposed a neural network model that incorporates COCOMO for software development cost and time forecasting. The COCOMO and NASA datasets were used for evaluation based on the MMRE and PRED criteria.

Idri et al. (2010) proposed two Radial Based Function Neural Network (RBFNN) model for software effort estimation. One model uses the C algorithm where the other model uses the Apc-III algorithm. 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. The results show that the accuracy of the estimation generated by the RBFNN model is affected by the type of the width formula used in the model.

Idri et al. (2008) investigated 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. The results show that the C-means algorithm performs better than the APC-III algorithm.

Reddy et al. (2008) 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 and Goel (2000) 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 (2002) 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. (2009) 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 and Murphy (2011) used machine learning techniques to create recommenders to triage bug reports that can be useful to streamline the development process.

Lopez-Martín (2011a,b) and Lopez-Martín et al. (2008, 2011) created regression models from short scale programs and from ISBSG repository. The authors 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 MMRE and MMER.

Li et al. (2010) proposed a holistic problem-solving approach which uses a ridge regression technique and multi-objective optimization. The experiments showed that adaptive regression models outperform machine learning models when multi-collinear datasets are used. In this research, Albrecht and Desharnais datasets were used and the evaluation was based on the MMRE, MdMRE and PRED(0.25).

References in the first paragraph of Section 1.5 focused on enhancing the UCP model; however, they did not tackle the main problems that exist in software estimation such as the non-linear relationship between software size and effort. Additionally, none of the previous work used neural network models to predict software effort from use case diagrams. On the other hand, References (Pendharkar et al., 2005; Papatheocharous and Andreou, 2007; Kumar et al., 2008; de Barcelos Tronto et al., 2008; Park and Baek, 2008; Attarzadeh and Ow, 2011; Idri et al., 2008, 2010; Reddy et al., 2008; Shin and Goel, 2000) used neural network models such as MLP and RBFNN to predict software estimation. References (Azzeh et al., 2010, 2011; Huang and Chiu, 2006) used soft computing techniques with analogy based estimation, whereas References (Idri and Abran, 2000; Huang et al., 2007) used soft computing with algorithmic models. References (Ahmed et al., 2005; Papatheocharous et al., 2010) used fuzzy logic and fuzzy decision tree, respectively for software effort estimation. Other works such as (Heiat, 2002; Lopez-Martín, 2011a,b; Lopez-Martín et al., 2008, 2011) developed neural network models and compared their works with regression models. Regression models such as linear, non-linear, stepwise and ridge have been used to predict software effort as shown in (Jiang et al., 2007; Xia et al., 2008; Tan et al., 2009; Li et al., 2010). Other work such as (Papatheocharous et al., 2010) was to make software estimation easier by reducing the number of the model's inputs without deteriorating the performance of the model.

The main distinguishing aspect of this work from the existing ones is that we propose a novel log-linear regression model to estimate software effort from use case diagrams that takes into consideration the non-linearity in the software size-effort relationship. Furthermore, none of the existing work proposed a MLP model to predict software effort from use case diagrams. In this paper, not only new regression and MLP models are proposed for software effort prediction, but also a thorough comparison between the proposed log-linear regression and MLP model was

conducted based on how large the software size is being used as an input to the models.

2. Regression model

This section presents the proposed regression model, the inputs to the model as well as the calibration of the productivity factor using fuzzy logic approach. This work is an extension to the work proposed in (Nassif et al., 2011a).

In statistics, regression analysis focuses on generating a relationship between a dependent variable (aka response) and one or more independent variables (aka predictors) (Allison, 1984). 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 (1805) and Gauss (1809) were among the first people who worked with regression models 200 years ago. There are many types of regression analysis. These include simple linear regression, multiple 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 as shown in (Heiat, 2002).

Eq. (7) shows how software effort is calculated from software size based on the UCP model. As shown in the equation, the relationship between software effort and size is linear and this assumption does not reflect the actual situation in the software industry as explained by McConnell (2006). McConnell 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". Secondly, Longstreet (2008) reported that when estimation is based on the Function Points method, the effort required to develop one Function Point is between 0.5 and 5 h for small projects (less than 100 function points) and between 20 and 60 h for large projects (greater than 7000 function points). Thirdly, the equation used by Boehm et al. (2000b) for software effort estimation is $Effort = a(SLOC)^b$. SLOC is the size in Source Lines of Code. Boehm's equation shows that the relationship between software size and effort in non-linear. Fourthly, Pendharkar and Rodger (2009) mentioned that the larger the project is, the larger the team is required. When the number of the team members increases, the number of the communication paths among this team will dramatically increase as shown in Eq. (16), and consequently, this requires more effort for the team communication and project management.

$$Communication\_Paths = \frac{N(N-1)}{2}. \tag{16}$$

where "N" is the number of people. Based on the above references, we conclude that when software size increases, software effort would increase but with a non-linear relationship. In this research investigation, a novel regression analysis is applied to generate a new equation to calculate software effort. The proposed regression model 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.

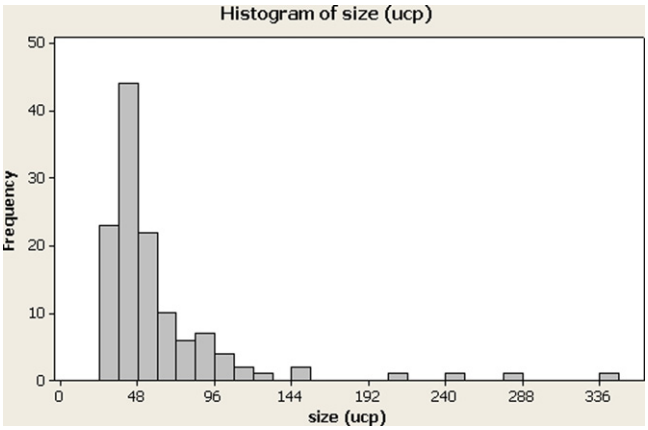

Fig. 1. Histogram of size.

The general equation of software effort can be represented as (Galorath and Evans, 2006):

$$Effort = \frac{Complexity}{Productivity} \times Size. \tag{17}$$

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 projects used in previous work (Nassif et al., 2011a) that have similar complexity and team productivity. Thus, at this point, complexity and productivity factors are ignored and software effort is a function of software size only. To obtain accurate results in regression analysis, data should be normally distributed (Cameron and Trivedi, 1998). If data were normally distributed, the regression equation would be:

$$Effort = a \times Size + b. \tag{18}$$

where a and b are constants.

The histograms of software size (Fig. 1) and software effort (Fig. 2) show that data are not normally distributed. Generating regression models from data based on Figs. 1 and 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 more normally distributed (Figs. 3 and 4). In this case, the linear regression is applied on ln(Size) and ln(Effort) instead as shown in Eq. (19). This is also called a log-linear regression.

$$\ln(Effort) = c \times \ln(Size) + d. \tag{19}$$

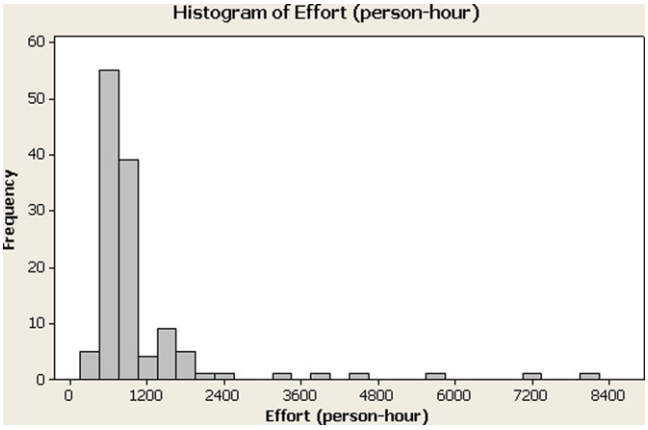

Fig. 2. Histogram of effort.

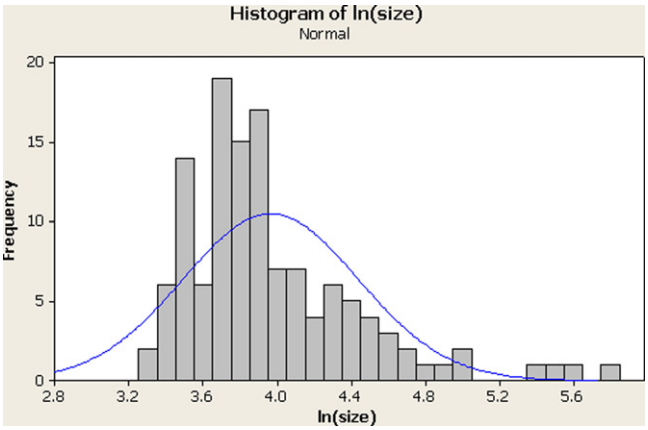

Fig. 3. Histogram of ln(size).

where c and d are constants. Eq. (19) can be rewritten as:

$$Effort = A \times Size^{B}. \quad (20)$$

Using Minitab, the values of A and B are 8.16 and 1.17 respectively. The values of A and B were determined based on the dataset used for training the regression model. When new datasets become available, the model can be calibrated. The method used for calibrations varies based on the source and the importance of the new dataset. For instance, more weight can be given for the new datasets. Furthermore, older projects (e.g. more than 5 years) can be deleted or given less weight. The Effort-Size relationship is represented as follows:

$$Effort = 8.16 \times Size^{1.17}. \quad (21)$$

where *Size* is the software size in UCP and *Effort* is the software effort in person-hours. For instance, Eq. (21) shows the non-linear relationship between Effort and Size and ignores the Complexity and Productivity factors. The main equation of the proposed regression model is expressed in Eq. (23).

Fig. 5 shows the relationship between software size and effort based on the UCP model as expressed in Eq. (7) (solid line) and the proposed log-linear regression model as expressed in Eq. (21) (marked by circles). 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 1000 UCP, software effort based on the log-linear regression model

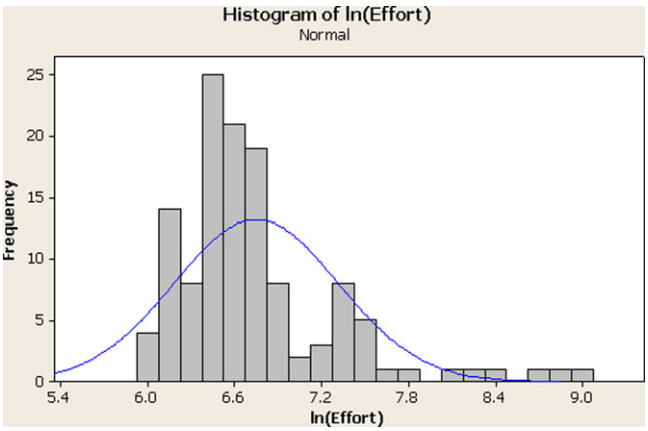

Fig. 4. Histogram of ln(effort).

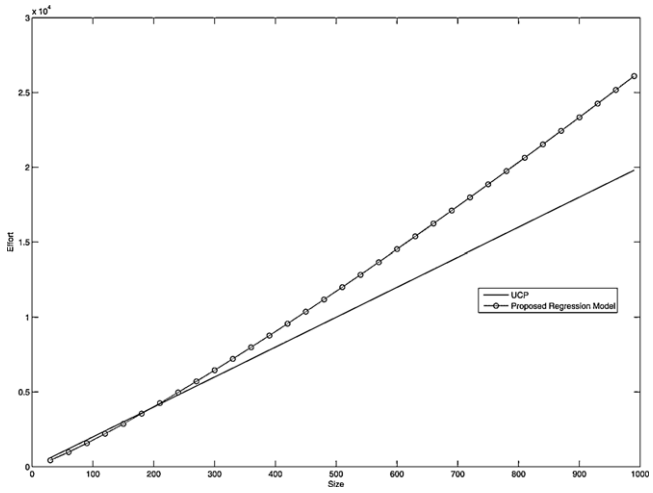

Fig. 5. Comparison between software size and software effort.

is larger than the software effort based on the original UCP model by 30%.

It is very important to test and validate the proposed regression equation (Eq. (21)) because this equation will be the core of the regression model. To thoroughly validate this equation, several techniques were used. These include the coefficient of determination R^2, Spearman and Pearson coefficients and the Analysis of Variance (ANOVA). R^2 is the percentage of variation in Effort explained by the variable Size. An acceptable value of $R^2 \geq 0.5$ (Humphrey, 1995).The value R^2 reported for Eq. (21) is 0.972. Approximately 97% of the variation in Effort can be explained by the variable Size. This shows a strong relationship between Size and Effort. To thoroughly test the regression model, Spearman (Lehmann, 1998) and Pearson (Edwards, 1976) 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. The Analysis of Variance (ANOVA) of Eq. (21) shows that the "*P*" value of the model as well as the predictors is 0.000. The *P* value is a probability with a value ranging between 0 and 1. In statistics, a *P* value of less than 0.05 indicates that the results are statistically significant at 95% confidence level. Since the p values of the model and the predictors are less than 0.05, we deduce that all independent variables (predictors) 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 Eq. (21) and Eq. (17), the main equation for software effort in the proposed model can be expressed as follows:

$$Effort = 8.16 \times \frac{Complexity}{Productivity} \times (Size)^{1.17}. \quad (22)$$

The second step of the proposed model is to calculate the values of project Complexity and Productivity. Table 3 presents some technical factors that represent the complexity of a project. We will assume that the UCP model's technical factor TF can represent the project complexity factor during the estimation of UCP and consequently, the Complexity factor in Eq. (22) can be

ignored. The main effort equation of the proposed regression model becomes:

$$Effort = \frac{8.16}{Productivity} \times (Size)^{1.17}. \quad (23)$$

With respect to productivity, Table 4 lists some productivity attributes represented by the environmental factors. In the original UCP model, productivity factor is only included when estimating the UCP size. Schneider and Winters (2001) 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 4, the highest productivity factor is achieved when the value of the factors E_1–E_6 is 5 and the value of the factors E_7 and E_8 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 *F*1–*F*6 is set to 0 and the value of *F*7 and *F*8 is set to 5. This implies that the value of *prod_sum* is −10. In the proposed model, the productivity factor in Eq. (23) 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 with two predictors (*Size* and *prod_sum*) as shown in Eq. (24).

$$Effort = 409 + (24.9 \times Size) - (52.8 \times prod\_sum). \quad (24)$$

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 is compatible with Eq. (23).

The value of the coefficient of determination R^2 of Eq. (24) is 0.861. This indicates that 86% of the variation in Effort can be explained by the independent variables *size* and *prod_sum*. The ANOVA of Eq. (24) shows that the "*P*" value of the model is 0.000 and the "*P*" value of each of the predictors is 0.000. This indicates that all independent variables are significant at the 95% confidence level.

From the above results, we deduce that the proposed multiple linear regression equation is valid and it will be used to determine the productivity factor in Eq. (23) 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 Eq. (23) based on each value of *prod_sum*. For this reason, the *productivity* variable will be depicted based on four main ranges of *prod_sum*. Since the *prod_sum* variable falls between [−10, 32.5], the main four regions of this variable will be selected as between [−10,0], between [1,10], between [11,20] and between [21, 32.5]. To find the influence of *prod_sum* on the dependent variable *Effort* in Eq. (24), four values of *prod_sum* will be 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 will be 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 Eq. (24), 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]).

Table 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

If we substitute the values of *size* and *Effort* of the aforementioned four rules in Eq. (23), the value of the *productivity* variable will be 0.4, 0.7, 1 and 1.3, respectively as shown in Table 5.

2.1. Productivity factor calibration

Usually in any project, the productivity factor is determined prior to the development of the project. For example, in COCOMO cost drivers, a productivity factor can be described as Very Low, Low, Nominal and High. Schneider and Winters (2001) also assigned values to productivity such as 1 for normal productivity (Effort = size × 20), 1.4 for low productivity (Effort = size × 28) and 1.8 for very low productivity (Effort = size × 36). In this research we followed the same approach as COCOMO and Schneider and we propose 4 levels of productivity as shown in Table 5. This makes the use of the productivity factor simpler. However, the main drawback of this approach is the abrupt change in productivity levels where previous work such as (Idri and Abran, 2000) and (Huang et al., 2007) tackled this issue in COCOMO. In this work, to resolve the drawback of the abrupt change in productivity levels and provide more accurate results, 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, the fuzzy system type is Mamdani (1977), the input membership of the fuzzy logic system used is Trapezoidal because the input is represented by a range (e.g. between 1 and 10) and Trapezoidal memberships can handle that by representing a range through the upper base of the Trapezoid. On the other hand, the output membership is Triangular because the output is a number and not a range and it can be represented as the triangle's vertex. The method used in the defuzzification stage is the centroid because this is the default and most used method. Matlab version 2010b was used to conduct the experiments of the fuzzy logic approach. Figs. 6 and 7 show the input and the output memberships, respectively. In Fig. 6, there are four membership functions which include mf1, mf2, mf3 and mf4. Each function represents the "If" part of the "If–Then" rule (aka the antecedent or premise). For instance, input membership mf2 represents the "If" part of the second rule which is "If *prod_sum* is between 0 and 10". On the other hand, the four output membership functions (mf1, mf2, mf3 and mf4) represent the "Then" part of the "If–Then" rule (aka the consequent or conclusion). For instance, the output membership mf2 represents "then productivity factor = 0.7) which corresponds to the vertex of the second triangle mf2.

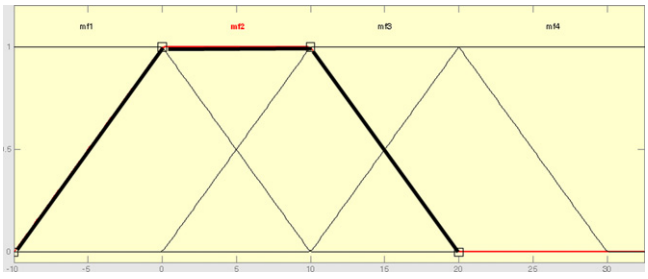

Fig. 6. Input membership.

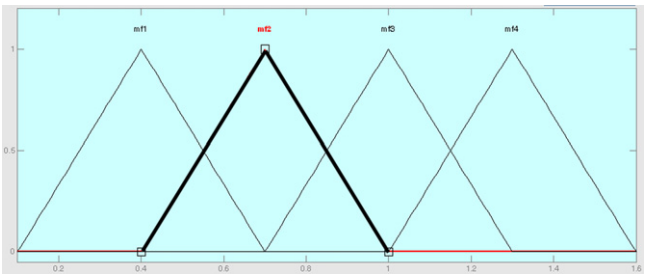

Fig. 7. Output membership.

There are two main approaches to elicit fuzzy rules (Xu and Khoshgoftaar, 2004). These include:

1. The If–Then rules are known. 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 paper, 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 fuzzy inference system shown in Fig. 8 can take any input within the input boundary (from −10 to 32.5). Based on the input value, two or more input membership function will be interpolated. The output value is determined based on the center of gravity of the output surface. For instance, Fig. 8 shows that when the input (productivity) is 9, the output is 0.817. After applying the fuzzy logic approach, the productivity factor has a specific value for each value of *prod_sum*. Table 6 shows some samples of the new values of the productivity factors. The labels IN, PO and PN correspond to *prod_sum*, old productivity factor and new productivity factor, respectively.

As seen in Table 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 more accurate estimation values. For instance, a complete list of the productivity factor values can be obtained using the proposed fuzzy logic inference system.

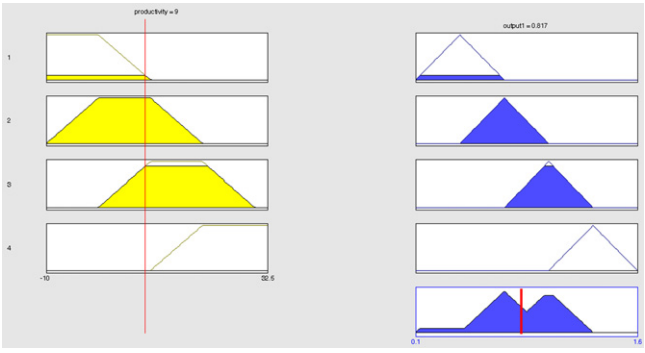

Fig. 8. Fuzzy inference system.

Table 6
New productivy factor.

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

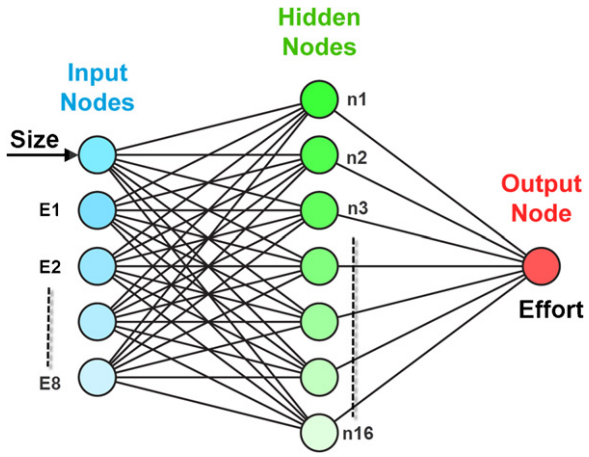

Fig. 9. MLP model.

3. Multilayer perceptron (MLP) model

This section presents the MLP neural network model. The main inputs to the proposed MLP model are software size and team productivity represented by the eight environmental factors (E_1–E_8 as shown in Table 4). The output of the model is software effort. The structure of the proposed neural network is depicted in Fig. 9. The MLP training parameters are listed in Table 7. The network will stop training when the number of epochs reaches 250 or when the Mean Squared Error (MSE) becomes zero or when the mu value exceeds $1e+10$. The time was set to "infinity" which indicates that the training time does not have a control on when the training should stop. The algorithm used to train the MLP model was Levenberg–Marquardt backpropagation. Ninety data points were used in developing the MLP model (data sets are explained in Section 4.1). Among the 90 data points, 60% were randomly used for training, 20% were used for validation and 20% were used for testing. The training data points were used in the training process and the model was adjusted according to their error. The validation data points were used to measure the network generalization and this cause the training to stop when generalization stops improving to prevent overfitting. Overfitting occurs when the model gives good results in training but bad results in the validation process. The testing data points have no effect on the training process and they

Table 7
MLP training parameters.

Parameter	Default value	Description
Epochs	250	Maximum number of epochs to train
Goal	0	Performance goal based on MSE
Min_grad	$1e-10$	Minimum performance gradient
mu	0.001	Initial learning rate (mu)
mu_dec	0.1	mu decrease factor
mu_inc	10	mu increase factor
mu_max	$1e+10$	Maximum mu
time	Inf	Maximum time to train in seconds

were used to measure the model performance during the training. Please note that the term "testing" used here is different from the model's evaluation conducted in Section 4. Testing in this context is used during the development (training) of the model. When the process of developing the model has finished, the model will be evaluated using 70 data points that were not included in the training process.

One of the important steps in developing the MLP model 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, we may get low training error but still have high generalization error due to overfitting. Blum (1992) and Linoff and Berry (2011) 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 our case, the number of hidden nodes falls between 10 and 18 since the number of inputs is 9. Within the data points used to develop the model, the size, environmental factors (E_1–E_8) and the actual effort of each project are known. In the training process, the k-fold ($k = 10$) cross-validation technique is used. This means that the 90 data points will be divided into 10 equal sets. The process will be repeated 10 times. In each time, 9 sets will be used for training and validation, and 1 set for testing. After the complete process has finished, all the sets will have been used in the training, validation and testing processes. The round with minimal testing error will be selected. After the MLP has been developed, it will be evaluated on 70 data points that were not included in the training stage. To demystify the process of training the neural network model, the following algorithm is used:

1- Assign the data (90 projects) to be used in developing the MLP model.
2- The remaining 70 projects will be used to evaluate the MLP model (Section 4).
3- Randomly divide the 90 data points 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 validation and 1 set for testing (for each value of "i", 9 different sets are used for training/validation and the remaining set for testing)
7- Record the testing error Vi-nh ("i" represents the number of the round, and "nh" the number of nodes in the hidden layer. For instance, the first testing error will be 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 process and exit.
11- Go to step "6"

Ten rounds of training/validation and testing 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. Experiments showed that the minimal value of the testing error occurred when the number of the hidden nodes is 16.

Fig. 10 shows the performance graph of the training process. The training curve is represented in blue (lower curve) and it shows that the training error (based on MSE) decreases when the number of epochs increases. Nonetheless, the best performance is not set when the number of epochs is 250 because the validation error (green curve) starts to increase after epoch 0. This means that

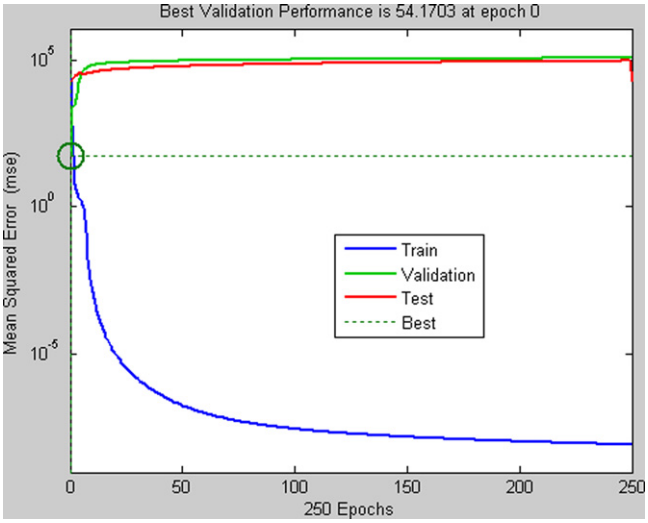

Fig. 10. Performance graph.

generalization stopped improving after epoch 0 and the training process was stopped at this stage to prevent overfitting. The testing curve (red color) also indicates that the error increases after epoch 0 and thus stopping the training at epoch 0 is a valid approach. Fig. 11 shows that regression graph of training, validation and testing. The best results are achieved when each of the training, validation and testing points represents a straight line. This means that the value of the correlation coefficient R is 1. The correlation coefficient measures the strength and the direction between two variables which are in this case the actual effort (x axis) and the estimated effort from the model (y axis). The values of R in training, validation and testing are 0.99, 0.82 and 0.76, respectively. These values show that there is a positive and strong relationship between the actual and estimated efforts in each of the training, validation and testing.

3.1. Testing the robustness of the MLP model

After the model has been developed, it was tested to see how stable and robust the model is. This is to study how the output of the MLP model varies when a change in the input occurs. The output of the model is a function of software size and team productivity. If we ignore the team productivity at this moment, the output (software effort) will be a function of software size. For the model to be stable and robust, we have two constraints. First, if the software size increases, software effort should increase. Second, the average output error of the model should fall within an acceptable error range. Twelve projects of size between 30 and 40 UCP (incremented by 10) were used to test the robustness. We found that output of the MLP model increases when the size increases and the error based on the log-linear regression model is acceptable.

4. Models' evaluation and comparison among models

This section presents the evaluation of the proposed MLP and log-linear regression models. A comparison was conducted between the proposed models and two other models that predict software effort from use cases which are the UCP and Schneider's models. Ninety data points were used in training the models and 70 data points were used for evaluation. The evaluation process was based on the MMER, PRED, RMSE, MAE and the standard deviation (SD) of the mean error (difference between actual and estimated effort).

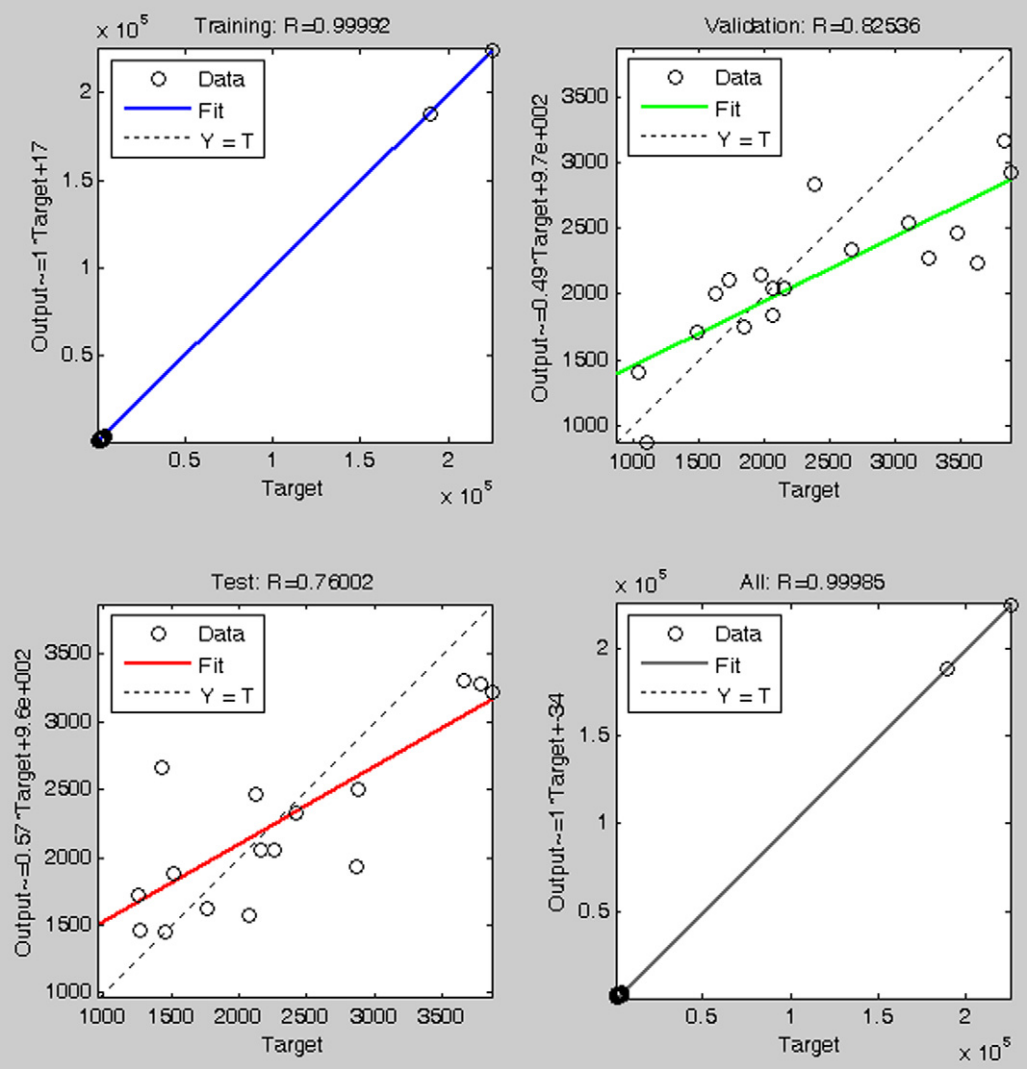

Fig. 11. Regression graph.

4.1. Project datasets

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. Public published datasets such as Desharnais, COCOMO, Albrecht and NASA cannot be used in our work because the unit of size in these datasets is either SLOC or function points. 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 Extension Scenario, actual software size and effort as well as some non-functional requirements such as factors contributing to productivity and complexity. One hundred and sixty industrial and educational projects were collected from three main sources. A statistical profile of these datasets is depicted in Table 8. The three datasets include:

- ISBSG: The default ISBSG repository does not contain projects that have required information about UML diagrams. We have requested special projects from ISBSG that contain information about the use case diagrams. As for our request, we have received 223 projects prepared specifically for us. These projects were filtered since many of them do not contain the information required in our research. Out of the 223 projects, 50 projects were selected to be used in the evaluation process that satisfy our requirements. FP size was converted to UCP using the rules proposed by Koirala (2009).

Table 8
Statistical profile of datasets.

Dataset	Mean	StDev	Minimum	Median	Maximum	Skewness	Kurtosis
Western	1672.4	414.3	696	1653	2444	−0.05	−0.82
CompuTop	20,573	47,327	570	3248	224,890	3.26	10.69
ISBSG	6081	9667	167	2554	57,156	3.78	16.94

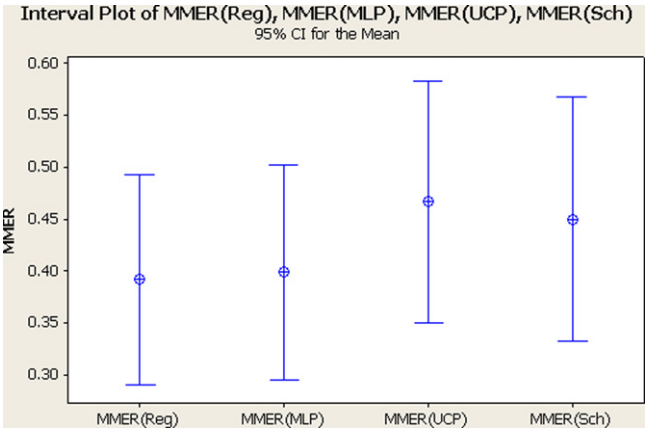

Fig. 12. MMER main dataset.

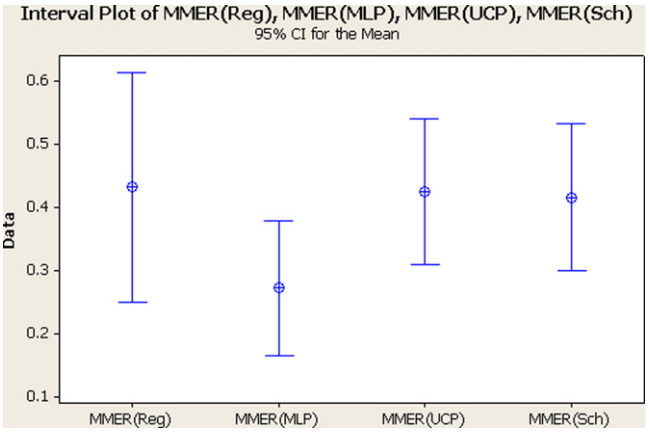

Fig. 13. MMER small dataset.

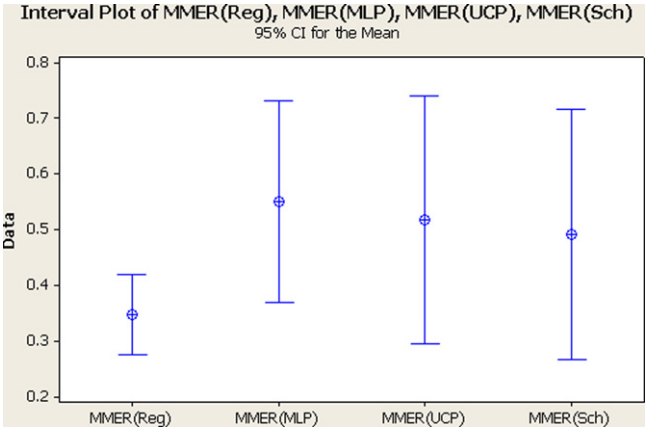

Fig. 14. MMER large dataset.

- Western University, Canada: Sixty five projects were collected from the fourth year software engineering students and from Master's students in the Computer Science department. The projects were developed and implemented using UML and object oriented languages. IBM Rational software was used as a CASE tool. Among these 65 projects, 55 projects were used in developing the models and 10 projects were used in the evaluation process.
- CompuTop: This is a medium-sized company overseas 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 were collected. Among these 45 projects, 35 projects were used in developing the models and 10 projects were used in the evaluation process.

4.2. Evaluation of MLP and log-linear regression models

Among the 160 projects in the three datasets, 90 projects were used to train the MLP model and 70 projects were used to evaluate (test) the model. Three main experiments were conducted to evaluate the MLP and Regression models. First, the proposed models were evaluated using the main dataset that contains the whole evaluating data points (70 projects). These 70 projects are different from the 90 projects used to train the model. In the second experiment, the models were evaluated using a dataset, named "small", that contains 38 projects of efforts less than 3000 person-hours. In the third experiment, the models were evaluated using a dataset, named "large", that contains 32 projects of efforts larger than 3000 person-hours. The main purpose of conducting three experiments is to study the performance of the MLP and log-linear regression models based on the size of the projects used in the evaluation (small versus large projects). Table 9 shows the evaluation results for the main, small and large datasets. The columns Reg, MLP, UCP and Sch correspond to the log-linear regression model, MLP model, UCP model and Schneider's model, respectively. Figs. 12–14 show the MMER Interval plots at 95% confidence level of the main, small and large datasets, respectively. Figs. 15–17 show the MAE Interval plots at 95% confidence level of the main, small and large datasets, respectively. Additionally, Figs. 18–29 depict the actual versus predict effort relationship of the regression model, MLP model, UCP model and Schneider's model based on the main, small and large datasets. Section 4.3 demonstrates a detailed explanation of Table 9 as well as Figs. 12–29.

4.3. Comparison among models

Table 9 shows the evaluation results of the proposed log-linear regression and MLP models, as well as the UCP and Schneider's models. The evaluation was conducted on five different criteria. These include the MMER, PRED, RMSE, MAE and SD of the mean error. Low values of MMER, RMSE, MAE and SD indicate good results. On the contrary, high PRED values indicate good results. Based on the main dataset (when all evaluation projects were used), the log-linear regression model and the MLP model have similar MMER and PRED values. However, RMSE, MAE and SD values show

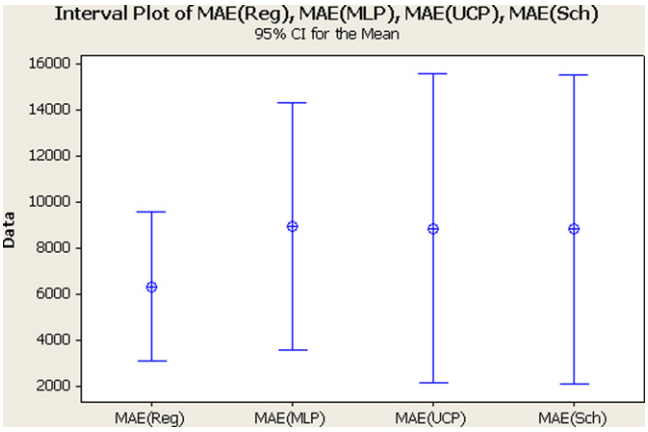

Fig. 15. MAE main dataset.

Table 9
Models evaluation.

Criteria	Main dataset				Small dataset				Large dataset			
	Reg	MLP	UCP	Sch	Reg	MLP	UCP	Sch	Reg	MLP	UCP	Sch
MMER	39.2	40	46.7	45	43.2	27.2	42.4	41.6	34.7	55	51.7	49.1
PRED (25)	37.1	45.7	34.2	40	36.8	60.5	36.8	39.4	37.5	28.1	31.2	40.6
PRED (50)	75.7	72.8	71.4	72.8	78.9	86.8	71	73.6	71.8	56.2	71.8	71.8
PRED (75)	94.2	90	90	90	89.4	94.7	89.4	89.4	100	84.3	90.6	90.6
PRED (100)	97.1	92.8	92.8	92.8	94.7	97.3	94.7	94.7	100	87.5	90.6	90.6
RMSE	14,922	24,136	29,322	29,317	872	747	1120	1204	22,126	35,689	43,351	43,341
MAE	6338	8940	8852	8830	690	484	882	942	13,181	18,981	18,317	18,196
SD	13,607	22,581	28,155	28,158	540	576	700	759	18,055	30,706	39,919	39,966

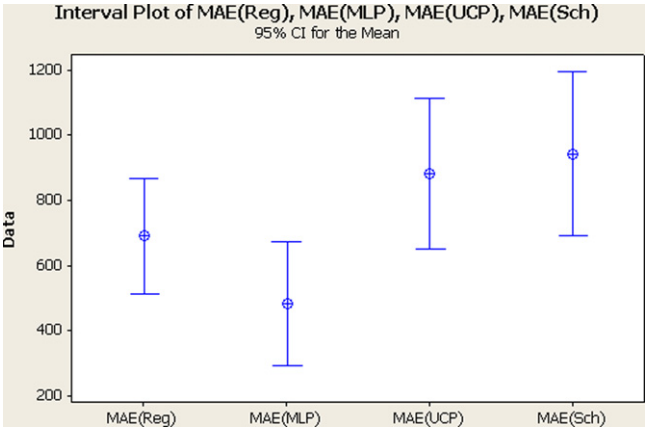

Fig. 16. MAE small dataset.

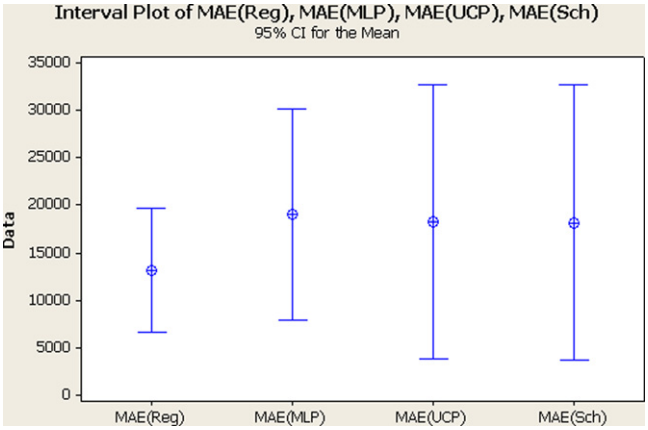

Fig. 17. MAE large dataset.

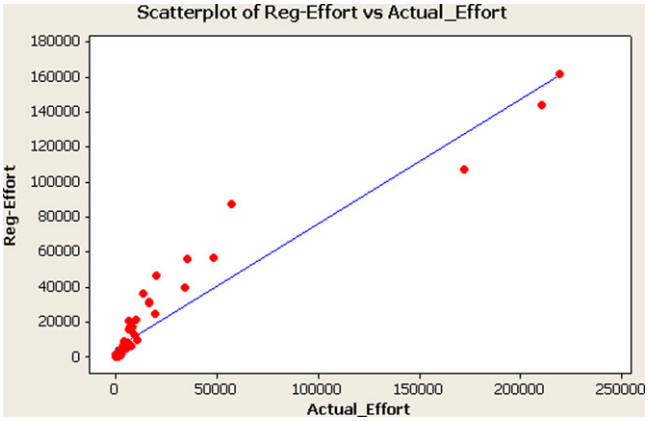

Fig. 18. Regression main dataset.

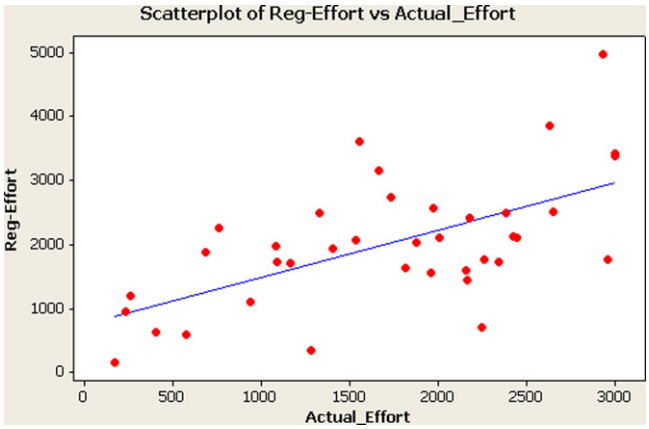

Fig. 19. Regression small dataset.

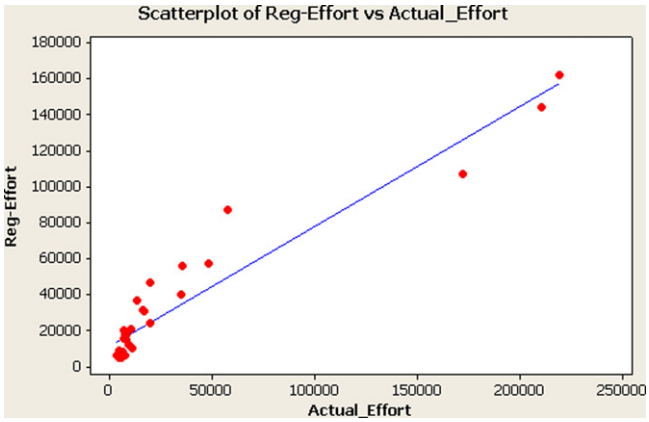

Fig. 20. Regression large dataset.

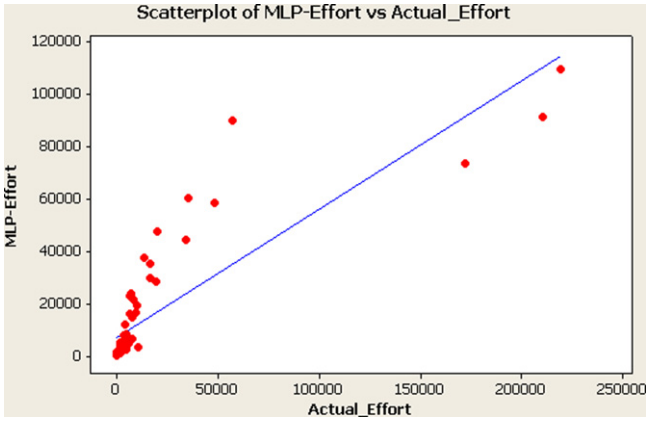

Fig. 21. MLP main dataset.

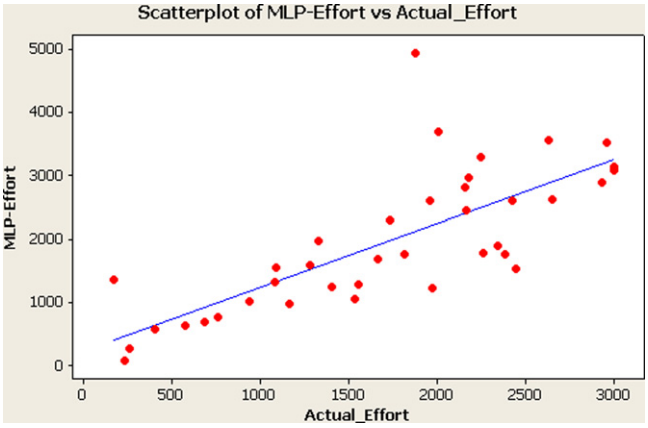

Fig. 22. MLP small dataset.

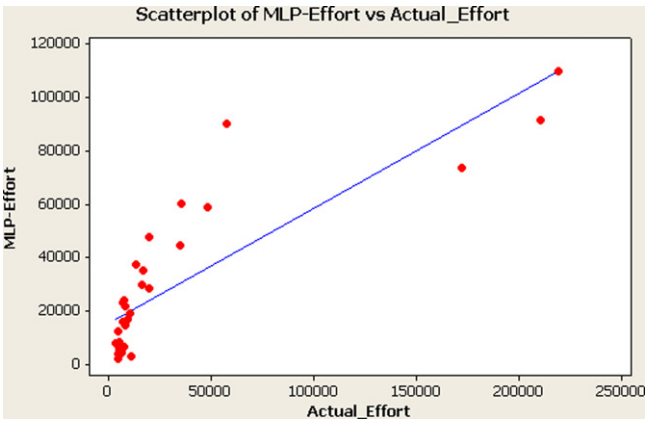

Fig. 23. MLP large dataset.

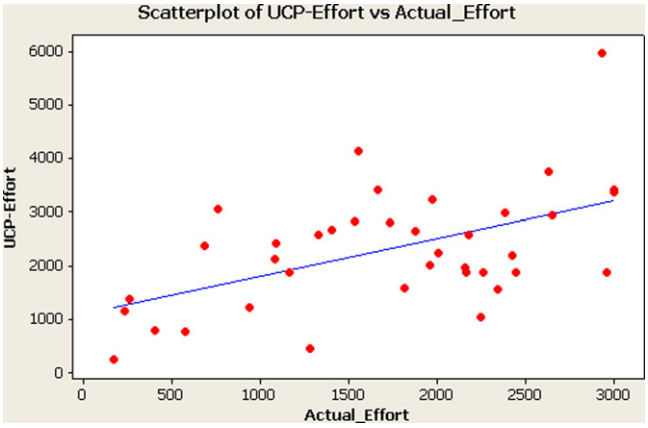

Fig. 25. UCP small dataset.

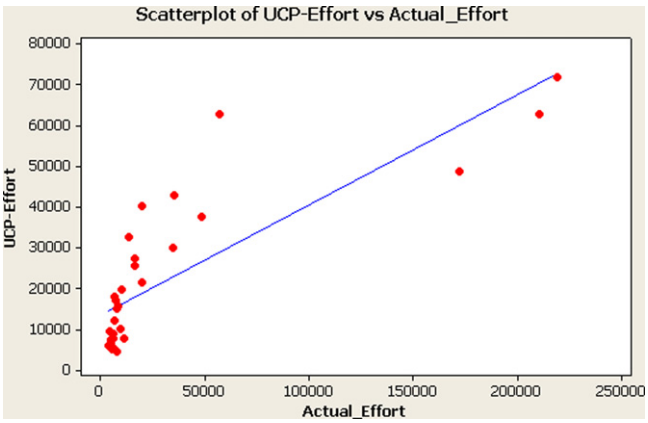

Fig. 26. UCP large dataset.

that the log-linear regression model surpasses the MLP model as well as the other models. When the small dataset was used, it is clear that the MLP model outperforms the log-linear regression model as well as the other models. The large dataset results show that the log-linear regression model is the best model and the performances of all other models deteriorate when large projects are used for effort estimation.

Figs. 12–14 depict the interval plots at 95% confidence level of the MMER criterion of the regression, MLP, UCP and Schneider's models for the three experiments (main, small and large). The centers of the intervals correspond to the MMER value of the corresponding models. Models with larger interval plots width indicate that there is a large difference between the minimum and maximum values of the MMER. This points out that models with shorter intervals width and low centers (MMER) perform better. Based on this analysis, Figs. 12 and 14 show that the log-linear regression model outperforms the other models. On the other hand, Fig. 13 shows that the MLP model exceeds the other models.

Similarly, based on Figs. 15 and 17, the log-linear regression model has the best results based on the MAE criterion. However, Fig. 16 shows that the MLP model outperforms the other models.

Figs. 18–29 show the relationship between the actual effort (x-axis) and the predicted effort (y-axis) of each of the four models (regression, MLP, UCP and Schneider) in each experiment (main

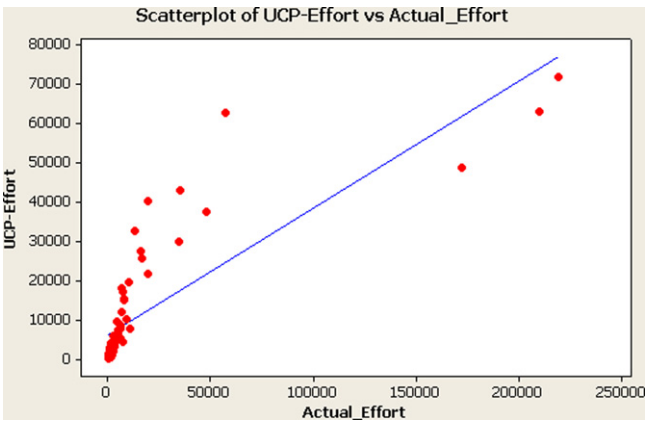

Fig. 24. UCP main dataset.

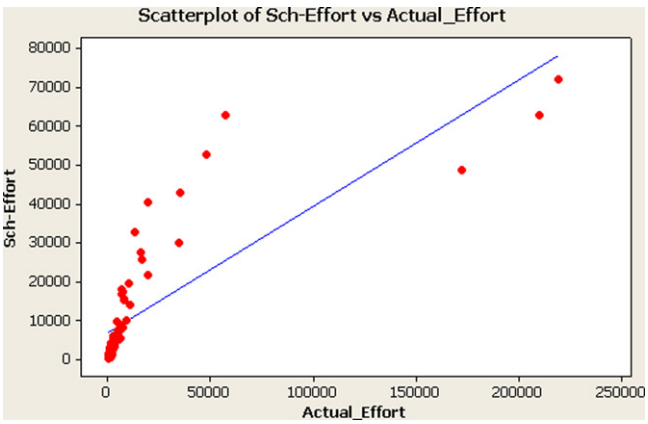

Fig. 27. Schneider main dataset.

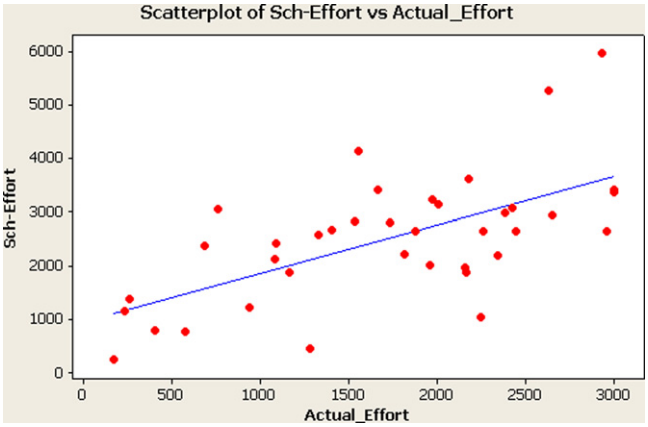

Fig. 28. Schneider small dataset.

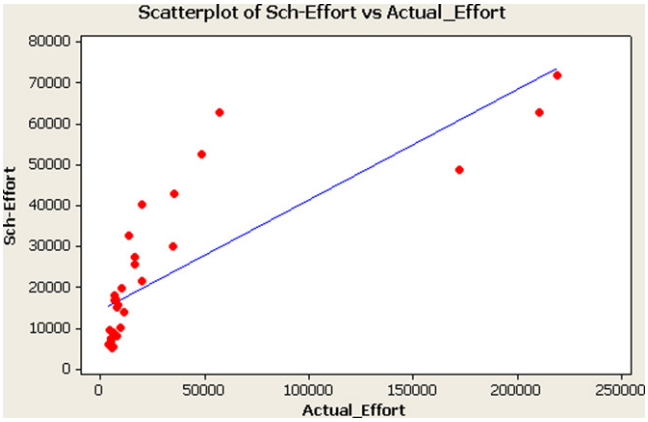

Fig. 29. Schneider large dataset.

dataset, small dataset and large dataset). The straight line represents the regression line which is the best straight line passing through the points. In the ideal situation, when all points fall on the straight line, this indicates that the predicted effort is equal to the actual effort which is not the case in real life. Based on the main dataset (Figs. 18, 21, 24 and 27), the log-linear regression model shows that it has the best performance with respect to the other models. However, based on the small dataset (Figs. 19, 22, 25 and 28), the MLP model outperforms the other models since the points are closer to the regression line. Similarly, based on the large dataset (Figs. 20, 23, 26 and 29), the log-linear regression model surpasses the other models.

As a conclusion from the evaluation results, the MLP model gives promising results for estimating projects in the Small Dataset (less than 3000 person-hours). However, the MLP model did not perform well with large and all datasets. To confirm these conclusions, a statistical test was used. The Anderson–Darling normality test was applied on the absolute residuals and we found that absolute residuals of all models are not normally distributed. For this reason, we used the non-parametric Mann–Whitney test and the results are reported in Table 10. Results show that, based on the Small Dataset, the MLP model is statistically significant in comparison with the regression, UCP and Schneider's models at the 95% confidence level. The Mann–Whitney test results support the results obtained in Table 9.

Table 10
Mann–Whitney test for absolute residuals.

	p-Value (small dataset)	*p*-Value (large dataset)	*p*-Value (all dataset)
MLP vs UCP	0.0015	0.3172	0.1858
MLP vs Schneider	0.010	0.2738	0.1942
MLP vs regression	0.0373	0.5682	0.3826

On the other hand, the log-linear regression model should be used for estimating projects of size larger than 200 UCP. Moreover, the original UCP model becomes inappropriate to estimate projects of efforts more than 10,000 person-hours.

5. Threats to validity

Threats to validity can be summarized as follows:

- The largest project used in the evaluation has an effort of 218,900 person-hours. The proposed log-linear regression and MLP models might be appropriate to estimate projects that are larger than 218,900 person-hours. Nevertheless, the limitation of the proposed models is set to the estimation of projects of maximum effort of 218,900 person-hours.
- One of the reasons that the MLP model did not perform well with large projects is because of the lack of more projects. This model was trained using 90 projects and the performance of this model would be better if more training projects were used.
- It was difficult to elicit the environmental factors (Table 4) 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.
- Because of the lack of industrial projects, some educational projects were used. Educational projects are mainly developed by students who work with these projects part time. Projects developed by inexperienced students might incur errors when the actual software effort is estimated.

6. Conclusions

This paper 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 where the environmental factors represent the team productivity. After the UCP size is calculated, software effort can be estimated by multiplying the UCP size 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 log-linear regression model was proposed to tackle these limitations. A multiple linear regression model was developed to predict the values of the productivity factor used in the proposed regression model. Additionally, a Mamdani fuzzy logic approach was used to adjust the values of the productivity factor.

Another contribution in this paper 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 log-linear regression and MLP models were evaluated using 70 industrial and educational projects based on five different criteria such as the MMER, PRED, RMSE, MAE and SD. A comparison among the log-linear regression model, the MLP model and two other models (the UCP and Schneider's models) that predict software effort from use case diagrams was conducted according to three different experiments. In the first experiment, all available data points (70 projects) that were not part of the training data points were used for evaluation. In the second experiment, 38 data points of efforts less than 3000 person-hours were used for evaluation, while in the third experiment, 32 data points of efforts greater than 3000 person-hours were used to evaluate the models. The results show that the proposed log-linear regression model surpasses all the models in the first and third experiments (all data points and large data points). On the other hand, the MLP model outperforms all other models in the second experiment (small data points) and this has been confirmed using the non-parametric Mann-Whitney Test. This had led to the conclusion that an MLP model can be used as an alternative to relevant regression models to estimate projects of effort less than 3000 person-hours. Furthermore, the proposed log-linear regression model can be used with promising results to estimate software effort especially with projects of effort more than 3000 person-hours.

The next step in this investigation will focus on improving the regression and the MLP 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

Ahmed, M.A., Omolade Saliu, M., AlGhamdi, J., 2005. Adaptive fuzzy logic-based framework for software development effort prediction. Information and Software Technology 47, 31–48.

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

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

Andreou, A.S., Papatheocharous, E., 2008. Software cost estimation using fuzzy decision trees. In: 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE2008), pp. 371–374.

Anvik, J., Murphy, G.C., 2011. Reducing the effort of bug report triage: recommenders for development-oriented decisions. ACM Transactions on Software Engineering and Methodology 20 (August), 10:1–10:35.

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

Attarzadeh, I., Ow, S.H., 2011. Software development cost and time forecasting using a high performance artificial neural network model. Intelligent Computing and Information Science 134, 18–26.

Azzeh, M., Neagu, D., Cowling, P., 2010. Fuzzy grey relational analysis for software effort estimation. Empirical Software Engineering 15, 60–90.

Azzeh, M., Neagu, D., Cowling, P.I., 2011. Analogy-based software effort estimation using fuzzy numbers. Journal of Systems and Software 84, 270–284.

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

Boehm, B.W., 1981. Software Engineering Economics. Prentice-Hall.

Boehm, B., Abts, C., Chulani, S., 2000a. Software development cost estimation approaches: a survey. Annals of Software Engineering 10, 177–205.

Boehm, B., Abts, C., Brown, W., Chulani, S., 2000b. Software Cost Estimation with COCOMO II. Addison Wesley, Upper Saddle River, NJ.

Braz, M.R., Vergilio, S.R., 2006. Software effort estimation based on use cases. In: COMPSAC'06, pp. 221–228.

Briand, L.C., Wieczorek, I., 2002. Resource estimation in software engineering. Encyclopedia of Software Engineering 2, 1160–1196.

Cameron, A.C., Trivedi, P.K., 1998. Regression Analysis of Count Data. Cambridge University Press, Cambridge, UK.

de Barcelos Tronto, I.F., da Silva, J.D.S., Sant'Anna, N., 2008. An investigation of artificial neural networks based prediction systems in software project management. Journal of Systems and Software 81, 356–367.

Demirors, O., Gencel, C., 2004. A comparison of size estimation techniques applied early in the life cycle. Software Process Improvement 3281, 184–194.

Diev, S., 2006. Use cases modeling and software estimation: applying use case points. SIGSOFT Software Engineering Notes 31, 1–4.

D. Eck, B. Brundick, T. Fettig, J. Dechoretz, J. Ugljesa, Parametric Estimating Handbook, The International Society of Parametric Analysis (ISPA), 4th edn., 2009.

Edwards, A., 1976. An Introduction to Linear Regression and Correlation. W. H. Freeman and Company.

Foss, T., Stensrud, E., Kitchenham, B., Myrtveit, I., 2003. A simulation study of the model evaluation criterion MMRE. IEEE Transactions on Software Engineering 29 (11), 985–995.

Galorath, D.D., Evans, M.W., 2006. Software Sizing, Estimation and Risk Management. Auerbach Publications, Boston, MA, USA.

Gauss, C.F., Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections (Theoria motus corporum coelestium in sectionibus conicis solem ambientum) (First published in 1809, Translation by C.H. Davis), Dover, New York, 1963.

Heiat, A., 2002. Comparison of artificial neural network and regression models for estimating software development effort. Information and Software Technology 44, 911–922.

Huang, S., Chiu, N., 2006. Optimization of analogy weights by genetic algorithm for software effort estimation. Information and Software Technology 48, 1034–1045.

Huang, X., Ho, D., Ren, J., Capretz, L.F., 2007. Improving the COCOMO model using a neuro-fuzzy approach. Applied Soft Computing 7 (1), 29–40.

Humphrey, W., 1995. A Discipline for Software Engineering. Addison Wesley.

Idri, A., Abran, A., 2000. COCOMO cost model using fuzzy logic. In: 7th International Conference on Fuzzy Theory and Technology, pp. 1–4.

Idri, A., Zahi, A., Mendes, E., Zakrani, A., 2008. Software cost estimation models using radial basis function neural networks. Software Process and Product Measurement 4895, 21–31.

Idri, A., Zakrani, A., Zahi, A., 2010. Design of radial basis function neural networks for software effort estimation. International Journal of Computer Science Issues 7, 11–17.

Jiang, Z., Naudé, P., Jiang, B., 2007. The effects of software size on development effort and software quality. International Journal of Computer and Information Science and Engineering 1, 230–234.

Jorgensen, M., 1995. Experience with the accuracy of software maintenance task effort prediction models. IEEE Transactions on Software Engineering 21, 674–681.

Jørgensen, M., 2007. Forecasting of software development work effort: evidence on expert judgement and formal models. International Journal of Forecasting 23, 449–462.

Karner, G., 1993. Resource estimation for objectory projects. Objective Systems.

Kitchenham, B.A., Pickard, L.M., MacDonell, S.G., Shepperd, M.J., 2001. What accuracy statistics really measure. IEEE Proceedings-Software 148 (3 (June)), 81–85.

Koirala, S., 2009. How to Prepare Software Quotation. bpb publications.

Kok, P., Kitchenham, B.A., Kirakowski, J., 1990. The MERMAID approach to software cost estimation. Esprit Technical Week.

Kumar, K., Ravi, V., Carr, M., Kiran, N., 2008. Software development cost estimation using wavelet neural networks. Journal of Systems and Software 81, 1853–1867.

Legendre, A., 1805. Nouvelles méthodes pour la détermination des orbites des comètes. Sur La Méthode Des Moindres Quarrés.

Lehmann, E.L., 1998. Nonparametrics: Statistical Methods Based on Ranks. Prentice Hall.

Li, Y., Xie, M., Goh, T., 2010. Adaptive ridge regression system for software cost estimating on multi-collinear datasets. Journal of Systems and Software 83, 2332–2343.

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

Lippman, R.P., 1987. An introduction to computing with neural nets. IEEE ASSP Magazine 3 (2), 4–22.

Longstreet, D., 2008. Estimating software effort. Software Metrics.

Lopez-Martín, C., 2011a. A fuzzy logic model for predicting the development effort of short scale programs based upon two independent variables. Applied Soft Computing 11 (1), 724–732.

Lopez-Martín, C., 2011b. Applying a general regression neural network for predicting development effort of short-scale programs. Neural Computing & Applications 20, 389–401.

Lopez-Martín, C., Isaza, C., Chavoya, A., 2011. Software development effort prediction of industrial projects applying a general regression neural network. Empirical Software Engineering 17, 1–19.

Lopez-Martín, C., Yanez-Marquez, C., Gutierrez-Tornes, A., 2008. Predictive accuracy comparison of fuzzy models for software development effort of small programs. Journal of Systems and Software 81, 949–960.

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

Mamdani, E.H., 1977. Application of fuzzy logic to approximate reasoning using linguistic synthesis. IEEE Transactions on Computers C-26, 1182–1191.

McConnell, S., 2006. Software Estimation: Demystifying the Black Art. Microsoft, Redmond, Washington.

Mendes, E., Mosley, N., Watson, I.,2002. A comparison of case-based reasoning approaches. In: Proceedings of the 11th International Conference on World Wide Web. Honolulu, Hawaii, USA, pp. 272–280.
Mohagheghi, P., Anda, B., Conradi, R., 2005. 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, pp. 303–311.
Nassif, A.B., Ho, D., Capretz, L.F., 2011a. Regression model for software effort estimation based on the use case point method. In: 2011 International Conference on Computer and Software Modeling, Singapore, pp. 117–121.
Nassif, A.B., Capretz, L.F., Ho, D., 2011b. 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, pp. 393–398.
Ochodek, M., Nawrocki, J., 2008. Automatic transactions identification in use cases. In: Meyer, B., Nawrocki, J.R., Walter, B. (Eds.), Balancing Agility and Formalism in Software Engineering. Springer-Verlag, Berlin, Heidelberg, pp. 55–68.
Ochodek, M., Nawrocki, J., Kwarciak, K., 2011. Simplifying effort estimation based on use case points. Information and Software Technology 53, 200–213.
Papatheocharous, E., Andreou, A.S., 2007. Software cost estimation using artificial neural networks with inputs selection. In: 9th International Conference on Enterprise Information Systems (ICEIS2007), Volume DISI – Databases and Information Systems Integration, pp. 398–407.
Papatheocharous, E., Papadopoulos, H., Andreou, A.S., 2010. Feature subset selection for software cost modelling and estimation. Engineering Intelligent 18, 233–246.
Park, H., Baek, S., 2008. An empirical validation of a neural network model for software effort estimation. Expert Systems with Applications 35 (10), 929–937.
Pendharkar, P.C., Rodger, J.A., 2009. The relationship between software development team size and software development cost. Communications of the ACM 52 (January), 141–144.
Pendharkar, P.C., Subramanian, G.H., Rodger, J.A., 2005. A probabilistic model for predicting software development effort. IEEE Transactions on Software Engineering 31, 615–624.
Periyasamy, K., Ghode, A., 2009. Cost estimation using extended use case point (e-UCP) model. In: International Conference on Computational Intelligence and Software Engineering.
Putnam, L.H., 1978. A general empirical solution to the macro software sizing and estimating problem. IEEE Transactions on Software Engineering 4, 345–361.
Reddy, C.S., Rao, P.S., Raju, K., Kumari, V.V., 2008. A new approach for estimating software effort using RBFN network. International Journal of Computer Science and Network Security 8, 237–241.
Robiolo, G., Orosco, R., 2008. Employing use cases to early estimate effort with simpler metrics. Innovations in Systems and Software Engineering 4, 31–43.
Robiolo, G., Badano, C., Orosco, R., 2009. Transactions and paths: two use case based metrics which improve the early effort estimation. In: International Symposium on Empirical Software Engineering and Measurement, pp. 422–425.
Schneider, G., Winters, J.P., 2001. Applied use Cases, Second Edition, A Practical Guide. Addison-Wesley.
Shin, M., Goel, A.L., 2000. Empirical data modeling in software engineering using radial basis functions. IEEE Transactions on Software Engineering 26, 567–576.
Tan, H.B.K., Zhao, Y., Zhang, H., 2009. Conceptual data model-based software size estimation for information systems. ACM Transactions on Software Engineering and Methodology 19 (October), 4:1–4:37.
Wang, F., Yang, X., Zhu, X., Chen, L., 2009. Extended use case points method for software cost estimation. In: International Conference on Computational Intelligence and Software Engineering.
Xia, W., Capretz, L.F., Ho, D., Ahmed, F., 2008. A new calibration for function point complexity weights. Information and Software Technology 50, 670–683.
Xu, Z., Khoshgoftaar, T.M., 2004. Identification of fuzzy models of software cost estimation. Fuzzy Sets and Systems 145, 141–163.
Zadeh, L.A., 1965. Fuzzy sets. Information and Control 8, 338–353.

Ali Bou Nassif is currently an adjunct professor at King's University College, as well as holding a position as a post-doctoral fellow at Western University, Canada. He obtained a Master's degree in Computer Science and a Ph.D. degree in Electrical and Computer Engineering from Western University in 2009 and 2012, respectively. Prior to joining Western, Ali worked in the IT field and provided IT services including, but not limited to, IT sales and consulting for several years. He has also taught many courses in Computer Science at the undergraduate level. His research areas include software effort estimation, requirements engineering, cloud computing and service oriented architecture.

Danny S.K. Ho is an independent management consultant and advisor for two startup companies. Prior to this, he held senior management and technical positions with Motorola Canada Limited, Nortel Networks Corporation and IBM Canada Limited. He is also appointed as an Adjunct Research Professor at the Department of Software Engineering at Western university, Canada. Throughout his professional career, he has led programs in the areas of wireline, RF, and infrared development; eMarketing, mCommerce, and software development environment. His areas of special interest include software estimation, project management, object-oriented software development, and complexity analysis. Danny received his Honors Bachelor of Science in Computer Science with Electrical Engineering, and Master of Science in Computer Science from Western. He is currently a member of the Professional Engineers Ontario (PEO) and a Project Management Professional (PMP).

Dr. Luiz Fernando Capretz has over 30 years of experience in the software engineering field as practitioner, manager and educator. He is currently the Assistant Dean for IT and e-Learning, and former Director of the Software Engineering Program at Western University, Canada. He has authored over 100 peer-reviewed research papers on software engineering in leading international journals and conference proceedings, and co-authored two books. His current research interests are software engineering, human aspects of software engineering, software product lines, and software engineering education. Dr. Capretz received his Ph.D. from the University of Newcastle upon Tyne (U.K.), M.Sc. from the National Institute for Space Research (INPE-Brazil), and B.Sc. from UNICAMP (Brazil). He is a senior member of IEEE, a distinguished member of the ACM, a MBTI Certified Practitioner, and a Certified Professional Engineer in Canada.