A Novel Soft Computing Model to Increase the Accuracy of Software Development Cost Estimation

Iman Attarzadeh, Siew Hock Ow
Department of Software Engineering
Faculty of Computer Science & Information Technology
University of Malaya, 50603 Kuala Lumpur, MALAYSIA
Email: attarzadeh@perdana.um.edu.my, show@um.edu.my

Abstract*— Software cost and time estimation is the process of estimating the cost and time required to develop a software system. Software cost and time estimation supports the planning and tracking of software projects. Effectively controlling the expensive investment of software development is one of the important issues in software project management. Estimating software development cost with high precision is still a great challenge for project managers, because it allows for considerable financial and strategic planning. Software cost estimation refers to the predictions of the likely amount of effort, time, and staffing levels required to build a software system. A very helpful form of cost estimation is the one made at an early stage during a project, when the costing of the project is proposed for approval. However, estimates at the early stages of the development are the most difficult to obtain. In this paper a novel Constructive Cost Model (COCOMO) based on soft computing approach is proposed for software cost estimation. This model carries some of the desirable features of neural networks approach, such as learning ability and good interpretability, while maintaining the merits of the COCOMO model. Unlike the standard neural networks approach, the proposed model can be interpreted and validated by experts, and has good generalisation capability. The model deals effectively with imprecise and uncertain input and enhances the reliability of software cost estimates. From the experimental results, it was concluded that, by the proposed neural network model, the accuracy of cost estimation can be improved and the estimated cost can be very close to the actual cost.

Keywords--Software engineering, software cost estimation models, COCOMO model, soft computing techniques, artificial neural networks.

I. INTRODUCTION

Estimating software development cost remains a complex problem, and one which continues to attract considerable research attention. Improving the accuracy of the cost estimation models available to project managers would facilitate more effective control of time and budgets during software development. The need for reliable and accurate cost estimation in software engineering was an ongoing challenge for software engineers in the last decade. In order to make accurate estimates and avoid large errors, several cost estimation techniques have been proposed. Among those techniques, Constructive Cost Model (COCOMO) is the most commonly used because of its simplicity for estimating the effort in person-month for a project at different stages. This paper proposed an effective model based on artificial neural networks and COCOMO II model to overcome the uncertainly problem and acquiring the accurate software cost estimation.

The paper is organised as follows. Section 2 describes the characteristics of cost estimation models, particularly COCOMO II model, and artificial neural networks concept that will be used in the proposed model. Section 3 and 4 discuss the related works and problem statement that it is inferred from the related works. Section 5 and 6 presents the new COCOMO II model based on artificial neural networks, and the training algorithm implemented. Section 7 discusses the system outputs and analysing of the results. Finally, section 8 concludes this paper.

II. SOFTWARE COST ESTIMATION MODELS AND ARTIFICIAL NEURAL NETWORKS

A. *Software Cost Estimation Models*

Software developers always interest to know the time estimation of software tasks. It could be done by comparing similar tasks that have already been developed. Although, estimating task has an uncertain nature, as it depends on several and usually not clear factors and it is hard to be modeled mathematically. Software schedule and cost estimation supports the planning and tracking of software projects. Effectively controlling the expensive investment of software development is of high importance [1]. The reliable and accurate cost estimation in software engineering is an ongoing challenge due to it allows for considerable financial and strategic planning. Software cost estimation techniques can be classified as algorithmic and non-algorithmic models. Algorithmic models are based on the statistical analysis of historical data (past projects), for example, Software Life Cycle Management (SLIM) [1] and Constructive Cost Model (COCOMO) [2].

Non-algorithmic techniques are based on new approaches such as, Parkinson, Expert Judgment, Price-to-Win and machine learning approaches. Machine learning is used to group together a set of techniques that represent some of the facets of human mind [3], for example regression trees, rule induction, fuzzy systems, genetic

978-1-4244-5586-7/10/$26.00 © 2010 IEEE

algorithms, artificial neural networks, Bayesian networks and evolutionary computation. The last five of these approaches are classified as soft computing group.

B. The COCOMO II Model

The COCOMO model is a regression based software cost estimation model. It was developed by Barry Bohem [2] in 1981 and thought to be the most cited, best known and the most plausible of all traditional cost prediction models. COCOMO model can be used to calculate the amount of effort and the time schedule for software projects. COCOMO 81 was a stable model on that time. One of the problems with using COCOMO 81 today is that it does not match the development environment of the late 1990's. Therefore, in 1997 COCOMO II was published and was supposed to solve most of those problems. COCOMO II has three models also, but they are different from those of COCOMO 81. They are [2]:

- Application Composition Model – Suitable for projects built with modern GUI-builder tools. Based on new Object Points.
- Early Design Model – To get rough estimates of a project's cost and duration before have determined its entire architecture. It uses a small set of new Cost Drivers, and new estimating equations. Based on Unadjusted Function Points or KSLOC.
- Post-Architecture Model – The most detailed on the three, used after the overall architecture for the project has been designed. One could use function points or LOC as size estimates with this model. It involves the actual development and maintenance of a software product.

COCOMO II describes 17 cost drivers that are used in the Post-Architecture model [2]. The cost drivers for COCOMO II are rated on a scale from Very Low to Extra High in the same way as in COCOMO 81. COCOMO II post architecture model is given as:

$$\mathrm{Effort} = \mathrm{A} \times [\mathrm{Size}]^{\mathrm{B}} \times \prod_{i=1}^{17} \mathrm{Effort\ Multiplier}_i \quad (1)$$

$$\text{where } \mathrm{B} = 1.01 + 0.01 \times \textstyle\sum_{j=1}^{5} \mathrm{Scale\ Factor}_j$$

In "1":

A: Multiplicative Constant

Size: Size of the software project measured in terms of KSLOC (thousands of Source Lines of Code, Function Points or Object Points)

The selection of scale factors (SF) is based on the rationale that they are a significant source of exponential variation on a project's effort or productivity variation.

C. Artificial Neural Networks

Artificial neural systems can be considered as simplified mathematical models of brain-like systems and they function as parallel distributed computing networks. However, in contrast to conventional computers, which are programmed to perform specific task, most neural networks must be taught, or trained. They can learn new associations, new functional dependencies and new patterns. The main idea for the field of Neural Networks (NN) originated from the desire to produce artificial systems capable of sophisticated, perhaps "intelligent", computations similar to the biological neurons in brain structures. Neural networks consist of layers of interconnected nodes, where each node produces a non-linear function of its input [3]. The nodes in the network are divided into the ones from the input layer going through the network to the ones at the output layer through some nodes in a hidden layer. The NN process starts by developing the structure of the network and establishing the technique used to train the network with using an existing data set. Therefore, there are three main entities: the neurons (nodes), the interconnection structure, and the learning algorithm. The most common technique in the use of the neural network for prediction is known as back-propagation trained feed-forward networks. Neural networks have been used in the software reliability modeling domain as well as software risk analysis [4]. Fig. 1 shows a simple neural network.

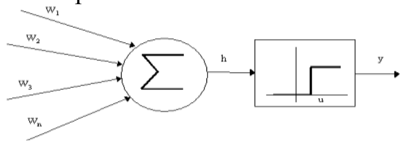

Figure 1. A simple of neural networks

As has been stated, an artificial neuron computes the weighted sum of its N inputs, x_j, where j = 1,2,……n, and generates an output of 1 if this sum is above a certain threshold u. Otherwise, an output of 0 results. It shows in below equation:

$$\mathrm{y} = \theta\left(\textstyle\sum_{j=1}^{n} W_j X_j - u\right) \quad (2)$$

Where θ is a unit step function at 0 and w_j is the synapse weight associated with the j-th input. U is considered as another weight i.e. w0 = -u attached to the neuron with a constant input of x0 = 1. Positive weights model excitatory synapses, while negative weights model inhibitory ones. The activation function in Fig. 1 is known as a step function however, there are a number of functions that can be utilised such as Gaussian, Linear, Sigmoid and Tanh. It is the Sigmoid function that is the most frequently used in neural nets. Neural network architectures are divided into two groups:

- feed-forward networks where no loops in the network path occur and
- feedback networks that have recursive loops

Of the different architectures, the feed-forward back-propagation is the most commonly used [5, 6].

III. Related Work

There are many software cost estimation models have been developed over the last decades. Also neural networks have learning ability and are good at modeling complex nonlinear relationships; provides more flexibility to integrate expert knowledge into the model. Prediction of software development effort using Artificial Neural Networks has focused mostly on the accuracy comparison of algorithmic models rather than on the suitability of the approach for building software effort prediction systems. The use of back propagation learning algorithms on a multilayer perceptron in order to predict development effort is well described by Witting and Finnie [7]. Karunanithi [8] reports the use of neural networks for predicting software reliability; including experiments with both feed forward and Jordon networks. Samson [9] uses an Albus multiplayer perceptron in order to predict software effort. They use Boehm's COCOMO dataset. Tadayon [10] also reports the use of a neural network with a back propagation learning algorithm. However it is not clear how the dataset was divided for training and validation purposes. Khoshgoftaar [11, 12] presented a case study considering real time software to predict the testability of each module from source code static measures. They consider Artificial Neural Networks as promising techniques to build predictive models. Artificial Neural Networks have been successfully applied to several problem domains. They can be used as predictive models because they are modeling techniques capable of modeling complex functions [13, 14].

IV. Problem Statement

Understanding and calculation of models based on historical data are difficult due to inherent complex relationships between the related attributes, are unable to handle categorical data as well as lack of reasoning capabilities. Besides, attributes and relationships used to estimate software development effort could change over time and differ for software development environments. In order to address and overcome to these problems, a new model with accurate estimation will be considerable.

V. The Proposed COCOMO II Model Based on Artificial Neural Networks

The proposed structure of the neural networks is customised to accommodate the COCOMO II Post-architecture model, which was identified in the previous section by "1". There are five scale factors denoted by SF and up to seventeen effort multipliers denoted by EM. The use of the neural network to estimate PM (person-month) in the "1" requires twenty-four input nodes in the input layer in the proposed neural network that corresponds to all EM and SF as well as two bias values. However, in order to structure the network to accomplish the COCOMO II post-architecture model, a specific hidden layer and a sigmoid activation function with some pre-processing of data for input layer is considered. The proposed network is not a fully connected network but specified hidden layer nodes that take into account the contribution of EM and SF separately as shown in Fig. 2 and 3.

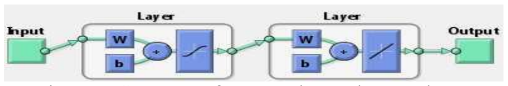

Figure 2. Structure of proposed neural network

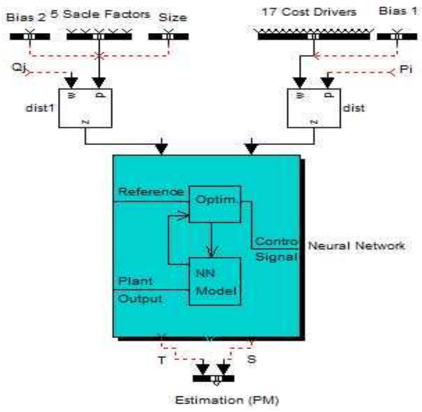

Figure 3. The proposed COCOMO II model based on neural network

Note that in the above network, all EM_i values used in COCOMO II are pre-processed to $log(EM_i)$ and the size of the product in KSLOC is not considered as one of the inputs to the network but as a cofactor for the initial weights for scale factors (SF). The activation function is the hidden layer is the sigmoid function defined by $f(x) = \frac{1}{1+e^{-x}}$. The weights associated to the input nodes connected to the hidden layer are denoted by P_i for Bias1 and each input $log(EM_i)$ for $1 \leq i \leq 17$. On the other hand, the weights associated with each SF_j input nodes to the hidden layer are q_j + log(size) for $1 \leq j \leq 5$ and the bias denoted by Bias2. S and T as shown in Fig. 2 denote the weights of the arcs from the hidden layer nodes to the output node. The weights S and T are relevant to the values of the nodes in the hidden layer. The output node has the identity function associated with it. One new concept to traditional neural networks is the addition of Log(Size) to the weight q_j of SF's input, which adjusts the weights q_j . Another major difference in this network compared to traditional neural network is the training aspect of neural network. In order to accommodate the COCOMO II formula, we adjust the initial values of weights S and T to the offset of the values of the nodes in the hidden layer. However, once we collect sufficient data set, we can use the back-propagation method to train the system. Initially, if the data set is not available or not reliable, the weights in the network results the COCOMO II estimation using random generated input/output. The output of the network (PM) would be derived from COCOMO II, "1" by considering the initial values of Bias1 as Log(A) and Bias2 as 1.01. The weights in the network are initialized as $p_i = 1$ for $1 \leq i \leq 17$ and $q_j = 1$ for $1 \leq j \leq 5$. Propagating the inputs forward, we get the values of nodes in the hidden layer as:

$$f(\,p_0\, Bias1 + \textstyle\sum_1^{17} p_i * log(EM_i)) = sigmoid\ (Bias1 + \sum_1^{17} p_i * log(EM_i)) = \frac{A * \prod_{i=1}^{17} EM_i}{1 + A * \prod_{i=1}^{17} EM_i} = \alpha \qquad (3)$$

Volumn 3

$$f((q_0 + log(size)) * Bias2 + \sum_{j=1}^{5}(q_j + log(size))(SF_j) = sigmoid\ (log(size) *$$

$$(\mathrm{Bias2} + \sum_{j=1}^{5} \mathrm{SF_j}) = \frac{\mathrm{Size}^{1.01+\sum_{j=1}^{5} \mathrm{SF}_j}}{1+\ \mathrm{Size}^{1.01+\sum_{j=1}^{5} \mathrm{SF}_j}} = \beta \qquad (4)$$

Then initialisation of weights S and T as follow:

$$\mathrm{S} = \frac{\beta}{2\,(1-\alpha)(1-\beta)} \quad \text{and} \quad \mathrm{T} = \frac{\alpha}{2\,(1-\alpha)(1-\beta)} \qquad (5)$$

The output of the network is calculated as:

$$\mathrm{PM} = \mathrm{S} * \alpha + \mathrm{T} * \beta = \frac{\alpha\beta}{(1-\alpha)(1-\beta)} = \mathrm{A.Size}^{1.01+\sum_{j=1}^{5} \mathrm{SF}_j} * \prod_{i=1}^{17} \mathrm{EM_i} \qquad (6)$$

Note that the initial values of S and T are adjusted so that the nodes in the hidden layer have same contribution to the output node PM. The following section describes the procedure for training the network.

VI. Training Algorithm

Using this approach, we iterate forward and backward until the terminating condition is satisfied. This terminating condition is when all changes in weights are below some threshold or a specific number of iterations have been done. The training algorithm is according to below steps.

- Choose a training sample and propagate the input vector across the network to get the output.
- Determine the error in output, and the error gradient in all the other layers.
- Determine the parameter changes for the neural networks weights and update the neural networks weights.
- Repeat until the neural networks error is sufficiently small after an epoch is complete.

The learning rate that affects the changes in weights corresponding to EM's and SF's has been subscripted with their related nodes.

VII. Results and discission

Experiments were done by taking two datasets, first one was original data from COCOMO dataset and second one was artificial dataset.

A. Datasets Description

The first used dataset for evaluating the proposed model is based on 93 historical projects from NASA, the NASA dataset is a public dataset and the information retrieved form 93 projects in NASA. The second attempt was to create an artificial dataset, Table 1, based on COCOMO model and training algorithm that it explained in previous section.

Table 1. The artificial dataset generated for system validation consists of 100 data samples

No.	Mode	Size	Effort
1	1.1200	51.2500	246.5900
2	1.2000	12.5500	58.2800
3	1.0500	81.5200	550.4000
…	...	…	…
97	1.2000	56.5300	354.7300
98	1.0500	16.0400	67.1400
100	1.1200	54.1700	262.3800

Therefore, this work has used two datasets for evaluation of the proposed model. Finally, by aggregate the accuracy across all testing datasets as the mean result.

B. Evaluation Method

For evaluating the different software effort estimation models, the most widely accepted evaluation criteria are the mean magnitude of relative error (MMRE) and probability of a project having a relative error of less than or equal to 0.25 (Pred(l)). The Magnitude of Relative Error (MRE) is defined as follows:

$$\mathrm{MRE_i} = \frac{|\mathrm{Actual\ \ Effort}_{\ i} - \mathrm{Predicted\ \ Effort}_{\ i}|}{\mathrm{Actual\ \ Effort}_{\ i}} \qquad (7)$$

The MRE value is calculated for each observation i whose effort is predicted. The aggregation of MRE over multiple observations (N) can be achieved through the Mean MRE (MMRE) as follows:

$$\mathrm{MMRE} = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}}^{\mathrm{N}} \mathrm{MRE_i} \qquad (8)$$

Another measure similar to MRE, the Magnitude of error Relative to the Estimate (MER), has been proposed. Intuitively, it seems preferable to MRE since it measures the error relative to the estimate. MER uses Predicted Efforti as denominator in "6". The notation MMER is used to the mean MER in "7". However, the MMRE and MMER are sensitive to individual predictions with excessively large MREs or MERs. Therefore, an aggregate measure less sensitive to extreme values is also considered, namely the median of MRE and MER values for the N observations (MdMRE and MdMER respectively). A complementary criterion is the prediction at level l, Pred(l) = k/N, where k is the number of observations where MRE (or MER) is less than or equal to l, and N is the total number of observations. Thus, Pred(25) gives the percentage of projects which were predicted with a MRE (or MER) less or equal than 0.25.

The proposed neural networks model was validated by two approaches. In the first approach, has used the NASA dataset that consists of 93 projects (Dataset #1). In the second approach, has used the artificial dataset that consists of 100 sample projects (Dataset #2). Then both datasets are applied to the new neural networks model and COCOMO II model. The validation of the new neural networks model to building trained neural networks model for effort estimation has been done using artificial dataset and NASA dataset. The comparison between the results of NASA dataset and artificial dataset that applied on the new neural networks model and COCOMO II model shows more accuracy in case of effort estimation by the new neural networks model. The comparisons between results are shown in Tables 2 and 3.

Table 2. Comparison between performance of new model and COCOMO II

Data set	Model	Evaluation	
		MMRE	Pred (25%)
Data set #1	COCOMO II	0.542561832	45%
	Proposed Model	0.467257017	52%

Data set #2	COCOMO II	0.462579313	30%
	Proposed Model	0.428617366	39%
Mean	COCOMO II	0.502570573	37.5%
	Proposed Model	0.447937192	45.5%

In this research, each dataset separately applied to the COCOMO II model and proposed model. Then for each model, the MMRE and Pred were calculated. Finally mean of those calculations are used to compare both models. The result for 193 applied projects shows the MMRE for COOCMO II model is 0.502570573 and for proposed model the value equals to 0.447937192. It shows the proposed model has MMRE less than COCOMO II model, so it means the accuracy of proposed model is better than COCOMO II. In case of Pred , the final result shows the proposed model value is 45.5% in Pred(25%) and COCOMO II value is 37.5% in same Pred. As it mentioned above, Pred shows the number of projects that they have MMRE lass than 25%. According to this definition, the proposed model shows better accuracy. Table 3 shows how much the proposed model is accurate than COCOMO II model.

Table 3. Accuracy of the proposed model

Model	Evaluation	
		MMRE
Proposed Model vs. COCOMOII	COCOMO II	0.502570573
	Proposed Model	0.447937192
	Improvement %	12.7

For comparing proposed model with COCOMO model, the improvement is 12.7% based on the MMRE 0.502570573 and 0.447937192. The experimental results show that the proposed software effort estimation model shows better estimation accuracy than the other two models, i.e., COCOMO. In summary, an output with more terms or neural networks sets provided a better performance due to the high granularity demanded from the results. Most of the sample data in the dataset with the proposed neural networks model resulted in a more accurate estimation when compared to the COCOMO II model.

VIII. Conclusion

An essential issue for project managers is the accurate and reliable estimates of the required software development effort, especially in the early stages of the software development life cycle. Software effort drivers usually have properties of uncertainty and vagueness when they are measured by human judgment. A software effort estimation model utilising neural networks can overcome these characteristics of uncertainty and vagueness exist in software effort drivers. However, the determination of the suitable neural networks model plays an important role in coming up with accurate and reliable effort estimates. Software effort estimation using neural networks is an attempt in the area of software project estimation. The objective of this work is to provide a technique for software cost estimation that performs better than other techniques on a given set of test cases. This paper presented a new model for handling imprecision and uncertainty by using the neural networks. The objective of this work is to provide a technique for software cost estimation that performs better than other techniques on the accuracy of effort estimation. This work has shown by applying neural networks on the algorithmic and non-algorithmic software effort estimation models accurate estimation is achievable. The proposed neural networks model showed better software effort estimates in view of the MMRE, Pred(0.25) evaluation criteria as compared to the traditional COCOMO. The above-mentioned results demonstrate that applying neural networks method to the software effort estimation is a feasible approach to addressing the problem of uncertainty and vagueness existed in software effort drivers. Furthermore, the neural networks model presents better estimation accuracy as compared to the COCOMO model. The utilisation of neural networks for other applications in the software engineering field can also be explored in the future.

References

[1] Boehm B. W. *"Software Engineering Economics"*, Englewood Cliffs, NJ, Prentice-Hall, 1981.

[2] Boehm B., Abts, C., and Chulani, S., Software Development Cost Estimation Approaches – A Survey, *University of Southern California Center for Software Engineering, Technical Reports*, USC-CSE-2000-505, 2000.

[3] Putnam, L. H., "A General Empirical Solution to the Macro Software Sizing and Estimating Problem", *IEEE Transactions on Software Engineering*, Vol. 4, No. 4, pp. 345 – 361, 1978.

[4] Srinivasan, K. and Fisher D., "Machine Learning Approaches to Estimating Software Development Effort", *IEEE Transactions on Software Engineering*, Vol.21, No. 2, 1995.

[5] Molokken, K., Jorgensen, M., "A review of software surveys on software effort estimation", *in Proceedings of IEEE International Symposium on Empirical Software Engineering, ISESE 2003*, pp: 223 – 230, Oct. 2003.

[6] Huang, S., Chiu, N., "Applying fuzzy neural network to estimate software development effort" , *in Proceedings of Applied Intelligence Journal*, Vol. 30, No. 2, pp. 73-83 , Apr. 2009.

[7] Witting, G., Finnie, G., "Using Artificial Neural Networks and Function Points to Estimate 4GL Software Development Effort", *Journal of Information Systems*, Vol .1, no.2, pp.87-94, 1994.

[8] N. Karunanitthi, D.Whitely, and Y.K.Malaiya, "Using Neural Networks in Reliability Prediction," *IEEE Software Engineering*,Vol.9, no.4, pp.53-59, 1992.

[9] Samson, B., etal. "Software cost estimation using an Albus perceptron" *Journal of Information and software.*, pp. 55-60, 1997.

[10]Tadion, N., "Neural Network Approach for Software Cost Estimation", *Proceedings of the International Conference on Information Technology: Coding and Computing,* ITCC, 2005.

[11]T.M.Khoshgoftaar, E.B.Allen, and Z.Xu, "Predicting testability of program modules using a neural network," *Proc.3rd IEEE Symposium on Application-Specific Systems and Sof. Eng. Technology*, pp.57-62, 2000.

[12]Jingzhou, L., Guenther, R., "Analysis of attribute weighting heuristics for analogy-based software effort estimation method AQUA+", *in Proceedings of Empirical Software Engineering Journal (2008)*, Vol. 13, No. 1, pp. 63–96, Feb. 2008.

[13]Liu. H , Yu, L., Toward Integrating Feature Selection Algorithms for Classification and Clustering, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, No. 4, pp. 491-502, Apr. 2005.

[14]Chiu N. H., Huang S. J., "The adjusted analogy-based software effort estimation based on similarity distances*". Journal of Systems and Software*, Vol. 25, pp. 628–640, 2007.