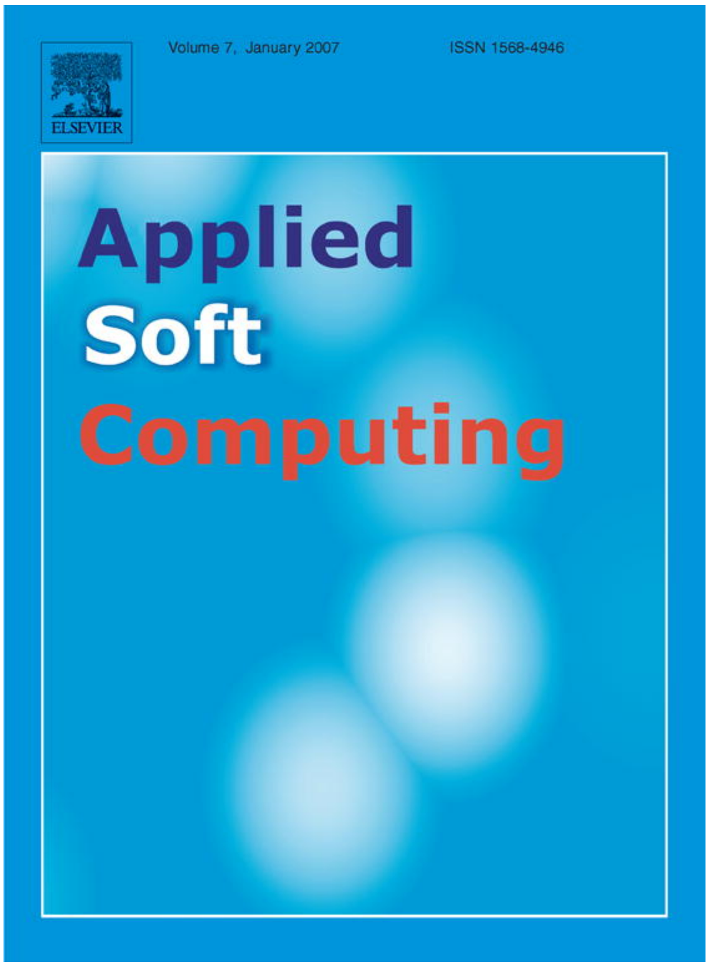
Volume 7, January 2007
ISSN 1568-4946
ELSEVIER
Applied Soft Computing

Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Applied Soft Computing 7 (2007) 29–40
Applied Soft Computing
www.elsevier.com/locate/asoc

Improving the COCOMO model using a neuro-fuzzy approach

Xishi Huang [a], Danny Ho [b], Jing Ren [a], Luiz F. Capretz [a,*]

[a] *Department of ECE, University of Western Ontario, London, Ont., Canada N6A 5B9*
[b] *Toronto Design Center, Motorola Canada Ltd., Markham, Ont., Canada L6G 1B3*

Received 6 December 2004; received in revised form 7 June 2005; accepted 26 June 2005

Abstract

Accurate software development cost estimation is important for effective project management such as budgeting, project planning and control. So far, no model has proved to be successful at effectively and consistently predicting software development cost. A novel neuro-fuzzy Constructive Cost Model (COCOMO) is proposed for software cost estimation. This model carries some of the desirable features of a neuro-fuzzy approach, such as learning ability and good interpretability, while maintaining the merits of the COCOMO model. Unlike the standard neural network approach, the proposed model can be interpreted and validated by experts, and has good generalization capability. 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. A detailed learning algorithm is also presented in this work. The validation using industry project data shows that the model greatly improves estimation accuracy in comparison with the well-known COCOMO model.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Software cost estimation; Neural network; Fuzzy set; COCOMO; Soft computing

1. Introduction

As software development has become an essential investment for many organizations, software estimation is gaining an ever-increasing importance in effective software project management. In practice, software estimation includes cost estimation, quality estimation, risk analysis, etc. Accurate software estimation can provide powerful assistance when software management decisions are being made; for instance, accurate cost estimation can help an organization to better analyze the feasibility of a project and to effectively manage the software development process, therefore, greatly reducing the risk.

Good software estimation is inherently a daunting task. Although many attempts [1,2,3,9,10,11,13] have been made to solve the problem in the last few decades, no approach has proven to be successful in

* Corresponding author. Tel.: +1 519 661 2111x85482; fax: +1 519 850 2436.
E-mail address: lcapretz@eng.uwo.ca (L.F. Capretz).

1568-4946/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2005.06.007

effectively and consistently predicting software output metrics. The principal challenges are: (1) the relationships between software output metrics and contributing factors exhibit strong complex nonlinear characteristics; (2) measurements of software metrics are often imprecise and uncertain; (3) there is difficulty in utilizing both expert knowledge and numerical project data in one model. Out of several current approaches to solve that problem, the neuro-fuzzy technique is a promising strategy. The neuro-fuzzy approach is a suitable solution due to its built-in learning ability, its robustness when faced with uncertain input and ease of utilizing different information from multiple sources [4].

We have taken into consideration the features of the cost estimation problem and various techniques and have proposed a novel neuro-fuzzy COCOMO model. The major difference between our work and previous works lies in that we have combined the COCOMO model [2], neural network techniques and fuzzy logic techniques into one scheme and have validated our approach with industry data.

Neural networks are used in our neuro-fuzzy COCOMO model to automatically tune the fuzzy rules from the numerical project data; fuzzy logic encodes expert knowledge directly using fuzzy rules with linguistic terms and neuro-fuzzy sub-models are used to calibrate the parameters of the COCOMO model. The parameters of standard COCOMO models are used to initialize the neuro-fuzzy model and therefore accelerate the learning process. The resultant model can be easily interpreted and has good generalization capability. First, the learning parameters in our model have concrete physical meaning and the whole decision process is clear to the users. As a result, our model can be interpreted and validated straightforwardly by experts and overcomes the "black box" problem that is common in neural network approaches. Second, as we choose triangular membership functions and use monotonic constraints for our model, our model can achieve good generalization. Another feature of our model is that it allows for continuous rating values and therefore avoids the problem of similar projects having large variances in cost estimations. Validation by industry project data shows that the proposed model can greatly improve cost estimation accuracy when compared with the COCOMO model.

2. Background

Because our neuro-fuzzy COCOMO model is based on the standard COCOMO model, the fuzzy logic model and neural network models, we briefly review these techniques.

The COCOMO cost and schedule estimation model originally published by Boehm [2] is one of most popular parametric cost estimation models of the 1980s. However, COCOMO'81 along with its 1987 Ada update experienced difficulties in estimating the costs of software developed according to new life-cycle processes and capabilities. Thus, the COCOMO II [1] research effort was started in 1994 at University of Southern California to address issues on non-sequential and rapid development process models, reengineering, reuse driven approaches, object-oriented approaches, etc.

COCOMO II Post Architecture Model is defined as:

$$\text{Effort} = A \times (\text{Size})^{B+0.01\times\sum_{i=1}^{5}\text{SF}_i} \times \prod_{i=1}^{17}\text{EM}_i$$

where A and B are baseline calibration constants, Size refers to the size of the software project measured in terms of thousands of Source Lines of Code (kSLOC), SF the scale factor, and EM is the effort multiplier.

2.1. Neural network models

Significant effort has been put into the research of developing software estimation models using neural networks [5,7,15]. Neural networks are based on the principle of learning from example with no prior information being specified. Neural networks are characterized in terms of three entities: neurons, interconnection structure and learning algorithms. Most of the software models developed using neural networks use multilayer feed-forward networks. The development of such a neural network model starts with an appropriate layout of neurons, or connections between network nodes. This includes defining the number of layers of neurons, the number of neurons within each layer, and the manner in which they are linked. The activation functions of the nodes and the specific training algorithm to be used must also be determined. Once the network has been built, the model must be trained by providing it with a set of historical project data input and the corresponding known actual values for project effort. The model then iterates on its training algorithm, automatically adjusting the weights (parameters) until the model weights converge. Once the training is complete and the appropriate weights for the network links are determined, new input can be presented to the neural network to predict the corresponding project effort. In general, large data sets are needed to accurately train neural networks.

2.2. *Fuzzy logic models*

A fuzzy system is a mapping between linguistic terms, such as ‘‘medium complexity’’ and ‘‘high cost’’ that are attached to variables. Thus an input into a fuzzy system can be either numerical or linguistic with the same applying to the output. A typical fuzzy system is made up of three major components: fuzzifier, fuzzy inference engine (fuzzy rules) and defuzzifier. The fuzzifier transforms the input into linguistic terms using membership functions that represent how much a given numerical value of a particular variable fits the linguistic term being considered. The fuzzy inference engine performs the mapping between the input membership functions and the output membership functions using fuzzy rules that can be obtained from expert knowledge of the relationships being modeled. The greater the input membership degree, the stronger the rule fires, thus the stronger the pull towards the output membership function. Since several different output membership functions can be contained in the consequents of rules triggered, a defuzzifier carries out the defuzzification process to combine the output into a single label or numerical value as required.

3. A novel neuro-fuzzy model

Our novel neuro-fuzzy model for software development effort estimation is shown in Fig. 1. The input for this model is the software size and ratings of 22 cost drivers including 5 scale factors (SFR_i) and 17 effort multipliers (EMR_i). The output is the software development effort estimation. Ratings of cost drivers can be continuous numerical values or linguistic terms such as ‘‘low’’, ‘‘nominal’’ and ‘‘high’’. The parameters in this model are calibrated by learning from industry project data.

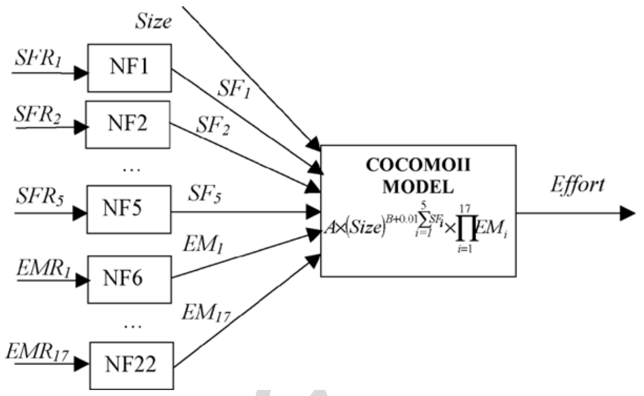

Fig. 1. A neuro-fuzzy model for software effort estimation.

There are two major components in our neuro-fuzzy model:

- Twenty-two sub-models NF_i: for each sub-model, the input is the rating value of a cost driver, and the output is the corresponding multiplier value, which is used as the input of the COCOMO model.
- COCOMO model: the input is the size of software and the output of NF_i. The output is software effort estimation.

3.1. *Sub-model NF_i*

There are 22 cost drivers in our neuro-fuzzy model. Each cost driver represents one factor that contributes to the development effort, such as application domain experience and product complexity. We use six qualitative rating levels to evaluate the contribution. When expressed in linguistic terms, these six rating levels are very low (VL), low (L), nominal (N), high (H), very high (VH) and extra high (XH). Each rating level of every cost driver relates to a value called a multiplier value, which is a quantitative value used in the COCOMO model.

Sub-model NF_i is used to translate the qualitative rating of a cost driver into a quantitative multiplier value and to calibrate these relations using industry project data. It should be noted that not all six rating levels are valid for all cost drivers.

A natural way to represent linguistic terms is to use fuzzy sets. We define a fuzzy set for each linguistic

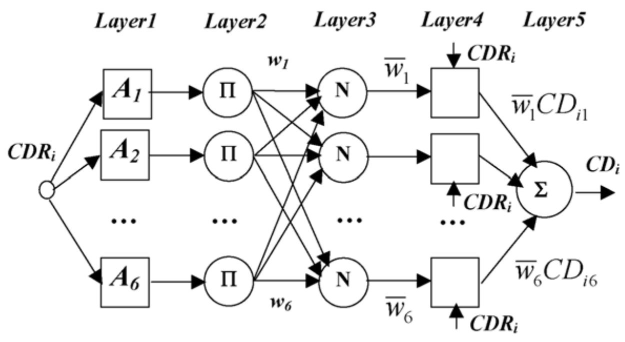

Fig. 2. ANFIS structure of sub-model NF$_i$.

term of every cost driver, i.e. "very low", "low", "nominal", "high", "very high", "extra high". The membership functions are either triangular functions or other functions, and the universe of discourse is the interval. We use fuzzy numbers "about 1", "about 2", . . ., "about 6" to represent linguistic terms "very low", "low", "nominal", "high", "very high", "extra high", respectively.

The Adaptive Neuro-Fuzzy Inference System (ANFIS) [8] is adopted for each NF$_i$. We denote CD$_i$ = SF$_i$, i = 1, 2, . . ., 5 and CD$_{i+5}$ = EM$_i$, i = 1, 2, . . ., 17. The input of NF$_i$ is the rating value CDR$_{ik}$ of the ith cost driver CD$_i$, and the output is the corresponding multiplier value CD$_i$. Fig. 2 shows the ANFIS structure of sub-model NF$_i$ for a 1-input–1-output system that is functionally equivalent to a Takaki and Sugeno's [14] type of fuzzy system having the rule base:

Fuzzy rule k: If CDR$_i$ is A_{ik}, then f_k = CD$_{ik}$, k = 1, 2, . . ., 6

where the fuzzy set A_{ik} represents a rating level that ranges from "very low" to "extra high", where CD$_{ik}$ is the corresponding multiplier value of the cost driver CD$_i$.

The node functions in the same layer are of the same function family as described below:

Layer 1: Every node k in this layer is a square node with a node function:

$$O_k^1 = \mu_{ik}(x)$$

where x is the input to the node k, and A_{ik} is the linguistic label (high, low, etc.) associated with this node function. In other words, $O_k^1 = \mu_{ik}(x)$ is the membership function of A_{ik} and it specifies the degree to which the given x satisfies the fuzzy set A_{ik}. Any continuous and piecewise differentiable functions, such as triangular-shaped membership functions, are qualified candidates for node functions in this layer. Parameters in this layer are referred to as *premise parameters*.

Layer 2: Every node in this layer is a circle node labeled *Π* which multiplies the incoming signals and outputs the product. For instance,

$$w_k = \mu_{ik}(x)$$

Each node output represents the firing strength of a rule. (In fact, other T-norm operators which perform generalized AND can be used as the node function in this layer.)

Layer 3: Every node in this layer is a circle node labeled *N*. The *k*th node calculates the ratio of the *k*th rule's firing strength to the sum of all rules' firing strengths:

$$\bar{w}_k = \frac{w_k}{\sum_{j=1}^{6} w_j}, \quad k = 1, 2, \ldots, 6$$

For convenience, the output of this layer are called *normalized firing strengths*.

Layer 4: Every node k in this layer is a square node with a node function:

$$O_k^4 = \bar{w}_k f_k = \bar{w}_k CD_{ik}$$

where $\bar{w}_k$ is the output of layer 3, and {CD$_{ik}$} is the parameter set. Parameters in this layer are referred to as *consequent parameters*.

Layer 5: The single node in this layer is a circle node labeled *Σ* that computes the overall output as the summation of all incoming signals, i.e.,

$$O_k^5 = \text{overall output} = \sum_k \bar{w}_k f_k.$$

In summary, the overall output of sub-model NF is:

$$\mathrm{CD}_i = \sum_k \bar{w}_k f_k = \sum_k \bar{w}_k \mathrm{CD}_{ik} \quad {}^{i}(1)$$

where

$$w_k = \mu_{ik}(x), \quad \bar{w}_k = \frac{w_k}{\sum_j w_j} \quad (2)$$

3.2. Interpretation of sub-model NF$_i$: linear interpolation

Theorem 1. *If membership functions are the following triangular functions*:

$$\mu_{ik}(x) = \begin{cases} x-(k-1), & k-1 \le x \le k \\ (k+1)-x, & k \le x \le k+1, \quad k=1,2,\ldots,6 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

for any continuous rating value CDR$_i$, the output CD$_i$ of sub-model NF$_i$ is a piecewise linear interpolation between the multiplier values CD$_{i1}$, CD$_{i2}$, . . ., CD$_{i6}$ of the ith cost driver CD$_i$, i.e.

$$\mathrm{CD}_i = \mathrm{CD}_{ik} + (\mathrm{CD}_{ik+1} - \mathrm{CD}_{ik})(\mathrm{CDR}_i - k), \quad k \le \mathrm{CDR}_i \le k+1, \quad k=1,2,\ldots,6 \tag{4}$$

Each sub-model NF$_i$ derived from this theorem gives the same results as a linear interpolation in mathematics when we select triangular membership functions. This is a very intuitive interpretation of our neuro-fuzzy model. In addition, linear interpolation has good generalization capability that is an important criterion for successful applications of neural networks and fuzzy logic techniques.

3.3. Learning algorithms

Here, we denote:

$$\mathrm{CD} \equiv \begin{bmatrix} \mathrm{SF}_{11} & \mathrm{SF}_{12} & \cdots & \mathrm{SF}_{16} \\ \mathrm{SF}_{21} & \mathrm{SF}_{22} & \cdots & \mathrm{SF}_{26} \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{SF}_{51} & \mathrm{SF}_{52} & \cdots & \mathrm{SF}_{56} \\ \mathrm{EM}_{11} & \mathrm{EM}_{12} & \cdots & \mathrm{EM}_{16} \\ \mathrm{EM}_{21} & \mathrm{EM}_{22} & \cdots & \mathrm{EM}_{26} \\ \vdots & \vdots & \vdots & \vdots \\ \mathrm{EM}_{N1} & \mathrm{EM}_{N2} & \cdots & \mathrm{EM}_{N6} \end{bmatrix}, \qquad X \equiv \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_5 \\ x_6 \\ x_7 \\ \vdots \\ x_{N_1} \\ x_{N_1+1} \end{bmatrix} = \begin{bmatrix} \mathrm{SFR}_1 \\ \mathrm{SFR}_2 \\ \vdots \\ \mathrm{SFR}_5 \\ \mathrm{EMR}_1 \\ \mathrm{EMR}_2 \\ \vdots \\ \mathrm{EMR}_N \\ \mathrm{Size} \end{bmatrix} \tag{5}$$

where $N = 17$, $N_1 = N + 5$ is the total number of cost drivers, x_i the rating value or linguistic term corresponding to the cost driver CD$_i$, $i = 1, 2, \ldots, N_1$, X the size of the software and rating values of the cost drivers, and CD is the corresponding multiplier values of cost drivers. If we choose triangular functions defined by Eq. (3) to be the membership functions, substituting Eq. (2) into Eq. (1), we have:

$$\mathrm{CD}_i = f_{\mathrm{NF}_i}(\mathrm{CD}_{i1}, \mathrm{CD}_{i2}, \ldots, \mathrm{CD}_{i6}) = \sum_{k=1}^{6} \mu_{ik}(x_i)\mathrm{CD}_{ik}, \quad i = 1,2,\ldots,N_1 \tag{6}$$

For our neuro-fuzzy model,

$$\text{Effort} = A \times (\text{Size})^{B+0.01\times\sum_{i=1}^{5}\mathrm{SF}_i} \times \prod_{i=1}^{N}\mathrm{EM}_i = A \times (\text{Size})^{B+0.01\times\sum_{i=1}^{5}\mathrm{CD}_i} \times \prod_{i=6}^{N_1}\mathrm{CD}_i \tag{7}$$

From Eqs. (6) and (7), we can rewrite our neuro-fuzzy model as follows:

$$\text{Effort} = f_{\mathrm{NF}}(X, \mathrm{CD}) \tag{8}$$

Given NN project data points (X_n, E_{dn}), $n = 1, 2, \ldots, NN$, the learning problem of parameters CD can be formulated as the following optimizing problem:

$$E = \sum_{n=1}^{NN} \frac{1}{2} w_n \left(\frac{E_n - E_{dn}}{E_{dn}}\right)^2 \tag{9}$$

subject to the following monotonic constraints:

$$\mathrm{CD}_{i1} \le \mathrm{CD}_{i2} \le \mathrm{CD}_{i3} \le \mathrm{CD}_{i4} \le \mathrm{CD}_{i5} \le \mathrm{CD}_{i6}, \quad i \in I_{\mathrm{INC}}(\mathrm{CD}) \tag{10}$$

$$\mathrm{CD}_{i1} \ge \mathrm{CD}_{i2} \ge \mathrm{CD}_{i3} \ge \mathrm{CD}_{i4} \ge \mathrm{CD}_{i5} \ge \mathrm{CD}_{i6}, \quad i \in I_{\mathrm{DEC}}(\mathrm{CD}) \tag{11}$$

where X_n is the software size and ratings of cost drivers for Project n, w_n the weight of Project n, E_{dn} the actual effort for Project n, $E_n = \mathrm{Effort}_n = f_{\mathrm{NF}}(X_n, \mathrm{CD})$ the corresponding effort estimation by our neuro-fuzzy model, $I_{\mathrm{INC}}(\mathrm{CD})$ the set of increasing cost drivers whose higher rating values correspond to higher development effort, and $I_{\mathrm{DBC}}(\mathrm{CD})$ is the set of decreasing cost drivers whose higher rating values correspond to lower development effort.

The learning algorithm for our neuro-fuzzy model is as follows:

$$\mathrm{CD}_{ik}^{l+1} = \mathrm{CD}_{ik}^{l} - \alpha \frac{\partial E}{\partial \mathrm{CD}_{ik}} \tag{12}$$

where $\alpha > 0$ is the learning rate,

$$\frac{\partial E}{\partial \mathrm{CD}_{ik}} = \sum_{n=1}^{NN} \frac{w_n}{E_{dn}^2}(E_n - E_{dn}) \frac{\partial E_n}{\partial \mathrm{CD}_{ik}} \tag{13}$$

$$\frac{\partial E_n}{\partial \mathrm{CD}_{ik}} = E_n \ln(\mathrm{Size}) \frac{\partial \mathrm{CD}_i}{\partial \mathrm{CD}_{ik}}, \quad \text{for } i = 1, 2, \ldots, 5 \tag{14}$$

$$\frac{\partial E_n}{\partial \mathrm{CD}_{ik}} = \frac{E_n}{\mathrm{CD}_i} \cdot \frac{\partial \mathrm{CD}_i}{\partial \mathrm{CD}_{ik}}, \quad i = 6, 7, \ldots, N_1 \tag{15}$$

$$\frac{\partial \mathrm{CD}_i}{\partial \mathrm{CD}_{ik}} = \frac{\partial f_{\mathrm{NF}_i}}{\partial \mathrm{CD}_{ik}} = \mu_{ik}(x_i). \tag{16}$$

3.4. Distinguishing features of the neuro-fuzzy model

- *Learning ability*: The proposed model has the learning/adaptation capability to model highly complex nonlinear relationships between software development effort and cost drivers, i.e. it can approximate any continuous nonlinear functions on a compact domain to arbitrary accuracy by learning.
- *Robust to imprecise and uncertain input*: The model can effectively deal with imprecise and uncertain input information, while remaining insensitive to imprecise and uncertain input such as ratings of the cost drivers. Because our model allows for continuous rating values (e.g. between one and six), it avoids the problem of similar projects with large different effort estimations (e.g. for two similar projects, COCOMO II produces estimates of 203 staff-months and 2886 staff-months, respectively; our model gives around 809 staff-months for both projects).
- *Good interpretability*: Although the neural network approach provides a powerful tool to model complex sets of relationships and learns from previous data, it has an inherent shortcoming: neither is it easy to understand nor is it easy to explain its decision process. However, our neuro-fuzzy model is clear to users during the whole decision process and its learning parameters EM_i and SF_i can be interpreted and validated by experts. In addition, if the rating value is a continuously variable value between one and six, the model produces the same result as that using mathematical linear interpolation; consequently, our neuro-fuzzy model is more easily accepted for project management.
- *Knowledge integration*: We can integrate expert knowledge with concrete numerical project data in our model using fuzzy rules. The model makes the best use of information from different sources in the decision-making process to achieve more accurate and reasonable cost estimations than conventional approaches. For example, we can integrate monotonic constraints that reflect expert knowledge of cost drivers into our model to guarantee that the calibration results are reasonable.
- *Good generalization*: Our model has good generalization by using linear interpolation and adding monotonic constraints.
- *Reduced number of learning parameters*: The parameters of membership functions in our model are fixed. EM_i and SF_i are the only parameters that need to be learned.
- *Local learning*: The learning models for the parameters are decoupled in our model. This feature allows the model to learn just some of the parameters each time; in other words, our model can accumulate knowledge locally.

3.4.1. *Ratings: discrete values versus continuous values*

The COCOMO model can only take on discrete ratings such as six linguistic terms: very low (VL), low (L), nominal (N), high (H), very high (VH) and extra high (XH), a constraint which may cause a problem in that the model could produce two rather different cost estimations for two similar projects. In order to illustrate this problem, let us consider two similar hypothetical projects P1 and P2:

- For cost drivers whose multiplier values are increasing, P1 is just below the lower limit of a linguistic value such as ‘‘high’’, and P2 is just above this limit.
- For those whose multiplier values are decreasing, P1 is just above the lower limit of a linguistic value such as ‘high’’, and P2 is just below this limit.

If we select the limit between two contiguous multiplier values with the largest difference for each driver, and P1 and P2 have the same nominal effort, say 100 staff-months, then for the COCOMO II 2000 model, the adjusted effort for P1 is 203 staff-months, but for P2 it is 2886 staff-months! So the difference between two estimations can be more than 14 times. The above problem results from using discrete rating values. Allowing continuously variable rating values as input, this serious problem can be eliminated, due to its gradual transition.

We use fuzzy sets in our neuro-fuzzy model rather than discrete rating values to represent the linguistic terms (‘‘very low’’, ‘‘low’’, etc.) for all cost drivers and triangular member functions that have the desirable linear interpolation property. As a result, our neuro-fuzzy model allows for continuous input and gives estimates of around 809 staff-months for both projects P1 and P2.

3.4.2. *Monotonic constraints*

Monotonic constraints are derived from expert knowledge and are critical for obtaining reasonable learning results. The following example illustrates their importance. The TURN cost driver represents computer turnaround time. Experience dictates that development effort should increase when the rating of TURN becomes higher. But if we do not use monotonic constraints on our model, we obtain the following fuzzy rules for the TURN cost driver:

If TURN is ‘‘very low’’, then $f_1 = 0.536$;
If TURN is ‘‘low’’, then $f_2 = 1.037$;
If TURN is ‘‘nominal’’, then $f_3 = 1$;
If TURN is ‘‘high’’, then $f_4 = 1.105$;
If TURN is ‘‘very high’’, then $f_5 = 1.019$.

This result can be validated by experts. Obviously, the second and fifth rules are not consistent. The calibrated multiplier value $f_2 = 1.037$ for the low TURN rating is greater than $f_3 = 1$ for the nominal TURN rating, indicating that the low rating requires more effort than the nominal rating. This result contradicts the above common sense that the low rating should need less effort than the nominal rating. Similarly, the calibrated multiplier value $f_5 = 1.019$ for the very high TURN rating is less than $f_4 = 1.105$ for the high TURN rating, implying that the very high rating requires less effort than the high rating. This also contradicts the above statement that the very high rating should require more effort than the high rating. When we place monotonic constraints on our model, we obtain the following reasonable rules:

If TURN is ‘‘very low’’, then $f_1 = 0.592$;
If TURN is ‘‘low’’, then $f_2 = 0.989$;
If TURN is ‘‘nominal’’, then $f_3 = 1$;
If TURN is ‘‘high’’, then $f_4 = 1.095$;
If TURN is ‘‘very high’’, then $f_5 = 1.095$.

These rules are consistent with the above-mentioned statements.

4. Validation with industry project data

In this section, we use industry project data to validate our neuro-fuzzy COCOMO model. There is a total of 69 project data available, including six project data from industry [6,12] and 63 project data from the original COCOMO’81 database [2]. We were not able to use the COCOMO II database due to the absence of individual cost driver ratings for the project data. Because most of the project data are compatible only with the intermediate COCOMO’81 model, we use the COCOMO’81 model for our

Table 1
Industrial project data

Project no.	Size (kESLOC)	Effort (staff-month)	High EMs	Low EMs	Very low EMs
P1	196.6	638.0	ACAP, PCAP, VEXP, LEXP, CPLX, TIME, STOR, TOOL, RUSE	DATA, VMVH, VMVT, SCED	
P2	51.8	185.0	ACAP, PCAP, AEXP, VEXP, LEXP, CPLX, STOR, TOOL, RUSE	DATA, VMVH, VMVT, SCED	
P3	64.1	332.0	VEXP, CPLX, STOR, TOOL, RUSE	DATA, VMVH, VMVT	
P4	130.0	619.9	RELY, STOR, RUSE	VEXP, LEXP, DATA, VMVH, VMVT	TURN
P5	13.3	64.8	VEXP, RELY, CPLX, STOR, RUSE	VMVH, VMVT, MODP	
P6	19.9	76.6	CPLX, TOOL, RUSE	AEXP, VEXP, LEXP, DATA, VMVH, VMVT	TURN

validation. The COCOMO'81 model contains only 15 effort multipliers, yet our neuro-fuzzy COCOMO model is completely compatible with the COCOMO II model.

Outliers are abnormal project data with large noise and thus we need to preprocess the project data and remove four outliers (with project data deviating from over 50% of the COCOMO'81 model output) from the original COCOMO'81 database. Hence, we use only 65 project data to train the neuro-fuzzy model, and use all 69 project data to test our neuro-fuzzy model.

The detailed industrial project data are shown in Table 1. Since all these six projects used a later COCOMO'87 model, which is slightly different from the original COCOMO'81 (e.g. three effort multipliers namely RUSE, VMVH and VMVT are not used in the COCOMO'81 model), the cost estimation that uses the COCOMO'81 model must be adjusted by multiplying a constant of 0.94116, which reflects a common rating of RUSE(H), VMVH(L), and VMVT(L) for these projects. Note also that the actual effort refers to the development effort adjusted so that it is compatible with the definition of the COCOMO model.

4.1. Performance evaluation criteria

We employ the following criteria to assess and compare the performance of cost estimation models. A common criterion for the evaluation of cost estimation models is the relative error (RE) or the magnitude of relative error (MRE), which are defined as:

$$\mathrm{RE}_i = \frac{\text{estimated effort}_i - \text{actual effort}_i}{\text{actual effort}_i}$$

$$\mathrm{MRE}_i = \frac{|\text{estimated effort}_i - \text{actual effort}_i|}{\text{actual effort}_i}$$

The RE and MRE values are calculated for each project i whose effort is predicted.

For N multiple projects, we can also use the mean magnitude of relative error (MMRE):

$$\mathrm{MMRE} = \frac{1}{N}\sum_{i=1}^{N} \frac{|\text{estimated effort}_i - \text{actual effort}_i|}{\text{actual effort}_i} = \frac{1}{N}\sum_{i=1}^{N}\mathrm{MRE}_i$$

Table 2
Effort estimation for all projects

RE (%)	COCOMO'81 model	Neuro-fuzzy model							
	PERC (%)	Case I		Case II		Case III		Case IV	
		PERC (%)	IMPRV (%)	PERC (%)	IMPRV (%)	PERC (%)	IMPRV (%)	PERC (%)	IMPRV (%)
20	71	86	15	88	17	88	17	89	18
30	81	92	11	92	11	92	11	94	13
50	94	97	3	97	3	97	3	95	1

Another criterion that is commonly used is the prediction at level p,

$$\mathrm{PRED}(p) = \frac{k}{N}$$

where k is the number of projects where MRE is less than or equal to p.

4.2. Case I: learning with all project data

We use the 65 project data points to calibrate the neuro-fuzzy model. The results are shown in Case I of Table 2 and Table 3. If we consider projects with relative error within 20% of actual efforts, it is noticed that cost estimation accuracy, using the neuro-fuzzy model, has improved by about 15% when compared with the COCOMO'81 model. If we are concerned only with the estimation accuracy for the industrial project data, there is an even larger improvement; that is, a more accurate estimation is achieved for every project, and MMRE improves from 15.7% to 7.1%.

4.3. Case II: learning with part of project data

In this case, we use 63 project data points to train our neuro-fuzzy model, saving two industrial project data points to test our model. The validation results are shown in Case II of Tables 2 and 3. The neuro-fuzzy model also gives better cost estimation than the COCOMO'81 model; for example, the neuro-fuzzy model predicts the effort more accurately for every industrial project data, and MMRE is improved from 15.7% to 8.4%. For all project data, the projects within 20% of actual efforts increase by 17% using the neuro-fuzzy model when compared with the COCOMO'81 model.

4.4. Case III: use larger weights for local organization data

Local organization data refers to industrial project data. We use the same 65 project data points here to train our neuro-fuzzy model but place different weights on project data from the COCOMO'81 database (weights = 1) and industrial project data (weights = 2). The validation results are shown in Case III of Tables 2 and 3. For local industrial project data, the results are better than Case I using the same weights for all projects, i.e. MMRE improves from 7.1% using the same weights to 4.6% using larger weights. Thus, we get a more accurate estimation for local project data when we assign more weights on local data.

According to Boehm's research [1], local calibration usually improves prediction accuracies because:

- The rating scale for COCOMO is subjective, leading to inconsistencies among different organizations.
- The life-cycle activities covered by COCOMO may vary slightly from the life-cycle activities covered by a particular organization.
- The definitions used by COCOMO may differ slightly from those being used by a particular organization.

Therefore, local project data are more accurate and consistent than multi-organization project data as they reflect the software development practice of one

Table 3
Effort estimation for industrial project data

Project no.	Actual effort	COCOMO'81 model		Neuro-fuzzy model							
				Case I		Case II		Case III		Case IV	
		Estimate	Error (%)	Estimate	Error (%)	Estimate	Error (%)	Estimate	Error (%)	Estimate	Error (%)
P1	638.0	827.0	29	745.0	16	739.7	15	728.6	14	697.2	9
P2	185.0	152.2	−17	167.3	−9	166.1	−10	163.6	−11	188.7	1
P3	332.0	279.8	−15	322.0	−3	306.8	−7	325.5	−1	338.5	1
P4	619.9	701.4	13	651.5	5	651.7	5	642.0	3	647.1	4
P5	64.8	71.1	9	60.8	−2	60.8	−6	64.1	−1	62.4	−3
P6	76.6	83.1	8	72.3	−5	72.3	−5	73.8	−3	72.9	−4
MMRE (%)		15.7		7.1		8.4		4.6		4.4	

particular organization. It is reasonable to assume that we should give larger weights for local project data when they are used to calibrate the cost estimation model.

4.5. Case IV. Learning without monotonic constraints

This case does not use monotonic constraints on multiplier values of cost drivers. The results are shown in Case IV of Tables 2–4. It is noticed that the estimated accuracy is higher in comparison with Case I (with monotonic constraints). For example, there are 89% of the total projects within 20% of actual efforts in this case versus 86% with monotonic constraints, and MMRE is improved from 7.1% to 4.4% for local industrial projects. However, some of the learned multiplier values are counter-intuitive, and are italicised in Table 4. For example, the multiplier value 1.019 corresponding to the ‘‘very high’’ rating is less than 1.105 of the ‘‘high’’ rating in the cost driver TURN. This implies that a project with very high computer turnaround time requires less effort than that of high computer turnaround time, which contradicts a common sense belief that development effort should increase as TURN increases. The ability to determine that the learned neuro-fuzzy model is not reasonable is one of the advantages of our neuro-fuzzy model. In order to overcome this counter-intuitive problem, monotonic constraints should be added to our neuro-fuzzy model to achieve a reasonable estimation.

Table 4
Learned effort multipliers without monotonic constraints

EM_i	VL (1)	L (2)	N (3)	H (4)	VH (5)	XH (6)
RELY	0.558	0.957	1.000	1.157	1.396	
DATA		*1.028*	1.000	1.164	1.429	
CPLX	0.695	0.739	1.000	1.173	1.493	1.868
TIME			1.000	*0.808*	1.076	1.383
STOR			1.000	*0.997*	1.113	1.544
VIRT		0.960	1.000	1.170	1.281	
TURN	0.536	*1.037*	1.000	1.105	*1.019*	
ACAP	2.754	1.575	1.000	0.885	0.714	
AEXP	1.187	1.041	1.000	0.974	0.878	
PCAP	2.582	*0.998*	1.000	0.700	0.646	
VEXP	*1.232*	1.381	1.000	*1.084*		
LEXP	1.164	1.120	1.000	0.782		
MODP	*0.788*	*0.830*	1.000	0.875	0.604	
TOOL	1.212	1.153	1.000	0.871	*1.867*	0.730
SCED	1.570	1.500	1.000	*0.880*	1.000	

Note: italicised values are counter-intuitive ratings.

5. Conclusion

Accurate software development cost estimation is very important in the budgeting, project planning and control, tradeoff and risk analysis of effective project management. This paper first reviewed major software cost estimation techniques such as model-based techniques, neural network models and fuzzy logic models. Each technique has its own advantages and shortcomings, but as yet no model has proved to be successful at effectively and consistently predicting software development effort.

Why is it so difficult to predict software development cost accurately? The principal reason is that software development is a complex process with the following characteristics: (1) there are highly complex nonlinear relationships between software development cost and cost drivers such as software size and other attributes of software product and the development process; (2) imprecise and uncertain measurement of software metrics, for example, cost drivers in the COCOMO model are usually measured qualitatively by selecting a fixed rating from a rating scale; (3) software technology and processes change rapidly.

Soft computing provides software developers with some promising techniques such as fuzzy logic, artificial neural networks, and evolutionary computation for software modeling. In particular, a neuro-fuzzy approach has been successfully used in many fields, and has demonstrated great potential to predict software cost more accurately. On the one hand, fuzzy logic is powerful in solving real world problems with imprecise and uncertain information and in dealing with semantic knowledge, as well as fuzzy logic models being easy to understand and interpret. However, difficulty is encountered in determining and fine-tuning fuzzy rules. On the other hand, artificial neural networks have the learning and adaptation ability to model complex nonlinear relationships and are capable of approximating any measurable functions; it can be difficult to understand and hard to explain its decision process due to its inherent ‘‘black box’’ nature.

The neuro-fuzzy approach that is based on these two well-established theories undoubtedly provides a promising tool to deal with many difficulties of software estimation; thus, neuro-fuzzy models are quite acceptable for project management. When we combine the neuro-fuzzy approach with the standard COCOMO models, we can take advantage of some desirable features of a neuro-fuzzy approach such as its learning/adaptation ability and good interpretability. In addition, we choose fixed triangular membership functions and put monotonic constraints on our model. Consequently, our model is capable of generalization, an important criterion for successful applications of neural networks and fuzzy logic techniques.

The learning parameters in our model have concrete physical meaning and the entire decision process is clear to the user. Therefore, this model can be interpreted and validated by experts. Another feature of our model is that it allows for continuous rating values as input, which eliminates the problem of similar projects with large different cost estimations. Validation by industry project data shows that our model can greatly improve cost estimation accuracy when compared with the standard COCOMO model.

Finally, the neuro-fuzzy technique allows the integration of numerical data and expert knowledge and can be a powerful tool when tackling important problems in software engineering such as cost and quality prediction. Therefore, a promising line of future work would be the extension of the neuro-fuzzy approach to other cost and quality estimation models and tools, such as COQUALMO, SLIM, SPR knowledgePLAN, and CA-Estimacs, among others.

References

[1] B.W. Boehm, et al. Software Cost Estimation with COCOMO II, Prentice Hall, 2000.

[2] B.W. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[3] S. Chulani, Bayesian Analysis of Software Cost and Quality Models, Ph.D. Dissertation, University of Southern California, Los Angeles, 1999.

[4] R. Fuller, Introduction to Neuro-Fuzzy Systems, Physica-Verlag, 2000.

[5] A.R. Gray, S.G. MacDonell, A comparison of techniques for developing predictive models of software metrics, Inf. Software Technol. 39 (1997) 425–437.

[6] D. Ho, Experience report on COCOMO and the costar tool from Nortel's Toronto Laboratory, in: Proceedings of 11th International Forum on COCOMO and Software Cost Modeling, University of Southern California, Los Angeles, October, 1996.

[7] A. Idri, T.M. Khoshgoftaar, A. Abran, Can neural networks be easily interpreted in software cost estimation? in: Proceedings of IEEE International Conference on Fuzzy Systems, 2002, 1162–1167.

[8] R.J.S. Jang, ANFIS: adaptive-network-based fuzzy inference system, IEEE Trans. Syst. Man Cybernetics 23 (1993) 665–685.

[9] S. MacDonell, A. Gray, A comparison of modeling techniques for software development effort prediction, in: Proceedings of the International Conference on Neural Information Processing and Intelligent Information Systems, Springer-Verlag, 1997), pp. 869–872.

[10] R.J. Madachy, Heuristic risk assessment using cost factors, IEEE Software 14 (3) (May/June 1997) 51–59.

[11] K.D. Maxwell, P. Forselius, Benchmarking software development productivity, IEEE Software 17 (1) (January/February 2000) 80–88.

[12] N. Panlilio-Yap, D. Ho, Deploying software estimation technology and tools: the IBM SWS Toronto Lab experience, in: Proceedings of the 9th International Forum on COCOMO and Software Cost Modeling, University of Southern California, Los Angeles, October, 1994.

[13] M. Shepperd, G. Kadoda, Comparing software prediction techniques using simulation, IEEE Trans. Software Eng. 27 (11) (November 1999) 1014–1022.

[14] T. Takaki, M. Sugeno, Fuzzy identification of systems and its application to modeling and control, IEEE Trans. Syst. Man Cybernetics 15 (1985) 116–132.

[15] G. Wittig, G. Finnie, Estimating software development effort with connectionist models, Inf. Software Technol. 39 (1997) 469–476.

Xishi Huang received his BE from Beijing Institute of Technology in 1987, and MESc from the University of Western Ontario in 2003. He is currently a PhD candidate in biomedical engineering at the University of Western Ontario, Canada. He is a student member of IEEE. His research interests include software engineering, soft computing and image processing.

Danny Ho works as a manager of Engineering Programs at Motorola Canada Limited. Prior to joining Motorola, he held management and senior technical positions at Nortel Networks Corporation and IBM Canada Limited. He is also an adjunct research professor in the Department of Electrical and Computer Engineering, University of Western Ontario. Throughout his professional career, he has led programs in the areas of wire-line, RF, and infrared development, desktop application deployment, reuse, and software development environment. His areas of special interest are software estimation, project management, object-oriented software development, and complexity analysis. Danny received his Honors BSc in computer science with electrical engineering, and MSc in computer science from the University of Western Ontario. He is currently a member of the Professional Engineers Ontario (PEO) and a Project Management Professional (PMP).

Jing Ren received her bachelor degree in 1993 from Shandong University, and a MESc degree in 2003 from the University of

Western Ontario. She is currently a PhD degree candidate in the Department of Electrical and Computer Engineering at the University of Western Ontario, Canada. She is a student member of IEEE. Her research interests include robotics, artificial intelligence and image processing.

Luiz F. Capretz has over 22 years of experience in the software engineering field as a practitioner, manager and educator. Before joining the University of Western Ontario, in Canada, he has worked since 1981 at both the technical and managerial levels, taught and carried out research on the engineering of software in Brazil, Argentina, England and Japan. He was the Director of Informatics and Coordinator of the computer science program at two universities (UMC and COC) in the State of Sao Paulo/Brazil. He has authored and co-authored over 50 peer-reviewed research papers on software engineering in leading international journals and conference proceedings, and has co-authored the book, *Object-Oriented Software: Design and Maintenance*, published by World Scientific. His current research interests are software engineering (SE), human factors in SE, software estimation, software product lines, and software engineering education. Dr. Capretz received his PhD from the University of Newcastle upon Tyne (U.K.), MSc from the National Institute for Space Research (INPE-Brazil), and BSc from UNICAMP (Brazil). He is a senior member of IEEE.