Software Development Effort Estimation in Academic Environments Applying a General Regression Neural Network Involving Size and People Factors

Cuauhtémoc López-Martín, Arturo Chavoya, and M.E. Meda-Campaña

Department of Information Systems
University of Guadalajara, México
{cuauhtemoc,achavoya,emeda}@cucea.udg.mx

Abstract. In this research a general regression neural network (GRNN) was applied for estimating the development effort in software projects that have been developed in laboratory learning environments. The independent variables of the GRNN were two size measures as well as a developer measure. This GRNN was trained from a dataset of projects developed from the year 2005 to the year 2008 and then this GRNN was validated by estimating the effort of a new dataset integrated by projects developed from the year 2009 o the first months of the year 2010. Accuracy results from the GRNN model were compared with a statistical regression model. Results suggest that a GRNN could be used for estimating the development effort of software projects when two kinds of lines of code as well as the programming language experience of developers are used as independent variables.

Keywords: Software engineering, software effort estimation, general regression neural network, statistical regression, programming language experience.

1 Introduction

The software process perspectives can be classified as follows [10]: organizations, teams and people. The performance of a software development organization is determined by the performance of its engineering teams, which in turn is determined by the performance of the individual team members, and finally the performance of the latter is, at least in part, determined by the practices these developers follow in doing their work [22]. The levels of software engineering education and training of each developer could be classified in the small and in the large software projects [6]. Software development effort estimation is one of the three main practices used for training developers of small projects at the personal level (the other two are related to software defects and software size [22]). Software development estimation techniques can be classified into two general categories:

1) Expert judgment. This technique implies a lack of analytical argumentation and aims at deriving estimates based on the experience of experts on similar projects; this technique is based on a tacit (intuition-based) quantification step [11].

© Springer-Verlag Berlin Heidelberg 2011

2) Model-based technique. It is based on a deliberate (mechanical) quantification step [11], and it could be divided into the following two subcategories:

a) Models based on Statistics: Its general form is a linear or nonlinear statistical regression model [3].

b) Models based on computational intelligence: These techniques have emerged in the software engineering field [21], and they include fuzzy logic [15], neural networks [5], genetic programming [4], and other evolutionary algorithms [1].

Based on the assumption of that no single technique is best for all situations and that a careful comparison of the results of several approaches is most likely to produce realistic estimates [2], this study compares the following models with each other: statistical regression and a neural network. Accuracy comparison between neural networks and statistical techniques has been a concern when applying these techniques to several fields such as accounting, finances, health care, medicine, engineering, manufacturing and marketing [18]. The two models were generated from data of small projects developed using practices of the Personal Software Process (PSP) because when this approach was applied to this kind of projects, the PSP has proved useful for delivering quality products on predictable schedules [22]. The models used in this research were generated from a dataset of 156 projects developed by 51 persons from the year 2005 to the year 2008 and then these two models were applied for estimating the development effort of a new dataset consisting of 156 projects developed by 47 persons from the year 2009 to the first months of the year 2010.

1.1 Description of Dependent and Independent Variables

The development effort of the software projects involved in this study was measured in minutes, whereas the size was measured as lines of source code (LOC); in fact, LOC remain in favour in many models [17]. There are two measures of source code size [20]: physical and logical. In this study, physical source lines are considered and all projects were developed based upon a similar coding standard as well as counted by the same counting standard.

The two kinds of physical lines of code that were used for estimating the development effort of this study were New and Changed (N&C) lines of code as well as reused code. Both Added plus Modified code make up the N&C, whereas reused code corresponds to LOC of previously developed projects that are used without modification [9].

Considering that, after software product size, people factors have the strongest influence in determining the amount of effort required in developing a software product [3], the models used in this paper involved an additional independent variable previously considered in [3]: programming language experience, which was measured in months for each developer.

1.2 Criterion for Accuracy Evaluation

The Magnitude of error Relative to the Estimate (MER) is used as criterion for evaluating and comparing the estimation models of this paper. MER was selected because the Magnitude of Relative Error (or MRE, which has commonly been used as criterion) is not strongly recommended and MER has showed better results than MRE [7]. The MER is defined as follows:

$$MER_i = \frac{|\text{Actual Effort}_i - \text{Estimated Effort}_i|}{\text{Estimated Effort}_i}$$

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

$$MMER = (1/N)\sum_{i=1}^{N} MER_i$$

The accuracy of an estimation technique is inversely proportional to the MMER. A reference for an acceptable value of Mean of Magnitude of error Relative to the Estimate (MMER) has not been found. In several papers, a Mean of the Magnitude of Relative Error or MMRE≤0.25 has been considered as acceptable, however, the authors who have proposed this value neither present any reference to studies nor any argumentation providing evidence [12].

1.3 Related Work

Models based upon neural networks and statistical regressions have already been applied for estimating development effort of large software projects [5] [8]. As for neural networks, the feedforward neural network (the most commonly used in the effort estimation field [19]), and a general regression neural network (GRNN) have already been applied for estimating software development effort [13] [16], however, in these two studies the developers' programming language experience was not considered because this independent variable was not found statistically significant in order to use it for comparison with other models.

2 Experimental Design

In this work, collected data were gathered from the same instruments (logs), phases, and standards suggested by the PSP. The experiment was done within a controlled environment having the following characteristics:

- This research involved only graduate students, because previous qualitative analysis have shown that within a PSP course, undergraduate students were more concerned with programming than with the software process issues [14] [23].
- All of the developers were employed and experienced, doing software development in their working environment; however, none of them had taken a course related to personal practices for developing software at the individual level.
- All developers were studying a graduate program related to computer science.
- Each developer wrote seven project assignments (one each day). However, only four of them were selected from each developer. The first three projects were not considered because they had differences in their process phases and logs, whereas in

the last four projects, phases were the same: plan, design, design review, code, code review, compile, testing and post-mortem, and they were based on the same logs.

• Each developer selected his/her own imperative programming language whose coding standard had the following characteristics: each compiler directive, variable declaration, constant definition, delimiter, assign sentence, as well as flow control statement was written in a line of code.

• In all projects the following instruments were used by all developers: project plan summary, defect type standard, time recording log, defect recording log and process improvement proposal. A test report template was introduced from the second to the seventh projects in the testing phase. A code review checklist was introduced from the third to the seventh projects, and design review checklist was used from the fourth to the seventh projects. Thus, from the fourth project on, all the developers used all practices and logs planned for this study. Hence, the first, second and third projects were excluded from this study; otherwise the comparison of the development time results would have been unfair.

• Developers had already received at least one formal course on the object oriented programming language of their choice and they had good programming experience in that language. The sample of this study reduced the bias because it only involved developers whose projects were coded in C++ or Java.

• As this study was an experiment with the aim of reducing bias, the developers were not informed about the experimental goal.

• Developers filled out a spreadsheet for each task and submitted it electronically for examination.

• All of the developers followed the same counting standard.

• Developers were constantly supervised and advised about the process.

• The code written in each project was designed to be reused in subsequent projects.

• The developed projects had complexity similar to that suggested by the original PSP [9] and are described in [15].

3 Description of Estimation Models

The data sample of projects for generating the models had the following characteristics:

1. It considered New and Changed code and Reused code.
2. Programming language experience was considered.
3. Data for all projects were correct, complete, and consistent.

The general regression neural network and the statistical regression were either trained or generated from a dataset of 156 projects developed by 51 persons from the year 2005 to the year 2008.

3.1 Statistical Regression Model

Equation (1) was generated using the actual data from the 156 projects.

$$\text{Effort} = 57.4098 + (1.1 * \text{N\&C}) - (0.18 * \text{Reused}) - (0.36 * \text{Programming Language Experience}) \quad (1)$$

The intercept value of 57.40 is the value of the line where independent variables are equal to zero. Signs of each of the three parameters meet with the following assumptions related to the software development:

1) The higher the value of new and changed code (N&C), the higher the development effort is.
2) The higher the value of reused code, the lower the development effort is.
3) The higher the value of programming language experience of developers, the lower the development effort is.

An acceptable value for the coefficient of determination is $r^2 \geq 0.5$ [9]. This equation had an $r^2 = 0.51$. The ANOVA for this equation had a statistically significant relationship between the variables at the 99% confidence level. To determine whether the model could be simplified, a parameter analysis of the multiple regression was done; results from this analysis showed that the highest p-value on the three independent variables was 0.0250, corresponding to reused code. Since this p-value is less than 0.05, reused code is statistically significant at the 95% confidence level (the software tool used was Statgraphics 4.0); consequently, the independent variable of reused code was not removed.

3.2 General Regression Neural Network

A General Regression Neural Network (GRNN) has the following advantages: (a) fast learning and (b) convergence to the optimal regression surface as the number of samples becomes very large. The GRNN has shown that, even with sparse data in a multidimensional measurement space, the algorithm provides smooth transitions from one observed value to another [24].

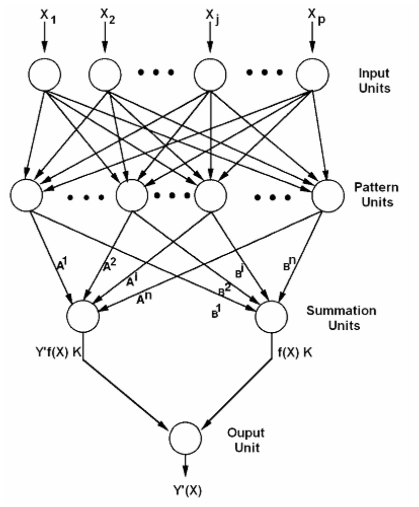

Fig. 1. General Regression Neural Network Diagram

Figure 1 shows the architecture of the implemented GRNN. Input units provide all the scaled measurement independent variables *X* to all neurons on the second layer, the pattern units. Each pattern unit can represent one sample exemplar, or, when the number of sample exemplars is large, a cluster center representing a subset of related exemplars in the sample [24]. When a new vector $\boldsymbol{X}$ is fed into the network, it is subtracted from the stored vector in the pattern units. The absolute values of the differences are summed and passed on to an exponential activation function. The pattern units output are fed into the summation units, which perform a dot product between a weight vector and a vector representing the outputs from the pattern units. The summation unit that generates an estimate of $f(X)K$ (where K is a constant determined by the Parzen window used, but that does need computing) sums the outputs of the pattern units weighted by the number of observations represented by each cluster center. The summation unit that estimates $Y'f(X)K$ multiplies each value from a pattern unit by the sum of the samples Y^j associated with cluster center X^i. The output unit merely divides $Y'f(X)K$ by $f(X)K$ to produce the desired estimate of Y [24].

4 Analysis

4.1 Verification of Models

The models presented in Section 3 were applied to the original dataset generated from 2005 to 2008 (the software tool used for training the GRNN was MatLab 6.1 having a spread=10 as best value for fitting) and the MER by project as well as the MMER by model were then calculated. For example, the actual data from a project was developed (design, design review, code, code review, compile and testing phases) in 171 minutes, it contained 95 N&C and 8 reused lines of code, respectively, and it was developed by a person having 14 months of experience in the programming language used. The multiple regression equation generated an effort equal to 155.49 minutes, whereas the GRNN yielded 168.83 minutes. The MER by model is the following:

Statistical regression

$$MER_i = \frac{|171 - 155.49|}{155.49} = 0.10$$

GRNN

$$MER_i = \frac{|171 - 168.83|}{168.83} = 0.01$$

The MMER values by model were the following:

- Multiple Linear Regression = 0.27
- General Regression Neural Network = 0.23

The ANOVA for MER for the projects showed that there was a statistically significant difference amongst the accuracy of estimation for the two techniques at the 95.0% confidence level. The following three assumptions of residuals for MER ANOVA were analysed: independent samples, equal standard deviations and normal populations.

4.2 Validation of Models

A new set of 47 programmers developed 156 projects in the year 2009 and the first months of 2010. These experiments had the same characteristics as the projects presented in Section 2. Once the two models for estimating the development effort were applied to these new data, the MER by project as well as the MMER by model were calculated. The MMER results by model were the following:

- Multiple Linear Regression = 0.24
- General Regression Neural Network = 0.26

Results from an ANOVA for MER showed that there was not a statistically significant difference between the two models at the 95% confidence level. In addition, the assumptions of residuals for MER ANOVA by group were analysed and the equal deviation as well as the normality were met.

5 Conclusions and Future Research

Based on the fact that after product size, people factors have the strongest influence in determining the amount of effort required to develop a software product, the models developed in this research include the developers' programming language experience.

This study compared two models for estimating software development effort of projects developed in a controlled experiment. One of the models developed is one of the most used in the software estimation field: statistical regression, whereas the other model corresponded to a computational intelligence technique: a neural network. These models were generated from a dataset composed of 156 projects developed by 51 persons. These two models were then applied to a new group of 156 projects developed by 47 programmers. All projects were developed in accordance with a similar experiment design within a laboratory learning environment and using a technology specifically designed for that kind of environment: Personal Software Process. In the verification stage, the general regression neural network (GRNN) had a better accuracy than the regression model, whereas in the validation stage, there was not a difference with statistical significance between the two models. These results suggest that a GRNN having as input new and changed code, reused code and programming language experience can be used for estimating the development effort of projects when developed in a laboratory learning environment and based upon a disciplined process as the one suggested by the Personal Software Process.

Based upon complexity increases with size of projects, future research involves the use of a GRNN for estimating the development effort of large scale projects.

Acknowledgment. The authors of this paper would like to thank CUCEA of Guadalajara University, Jalisco, México, Programa de Mejoramiento del Profesorado (PROMEP), as well as Consejo Nacional de Ciencia y Tecnología (Conacyt).

References

1. Aguilar-Ruiz, J.S., Ramos, I., Riquelme, J.C., Toro, M.: An evolutionary approach to estimating software development projects. Journal of Information and Software Technology 43, 875–882 (2001)

2. Boehm, B., Abts, C., Chulani, S.: Software development cost estimation approaches: A survey. Journal of Annals of Software Engineering 10, 177–205 (2000)
3. Boehm, B., Abts, C., Brown, A.W., Chulani, S., Clarck, B.K., Horowitz, E., Madachy, R., Reifer, D., Steece, B.: COCOMO II. Prentice Hall, Englewood Cliffs (2000)
4. Burguess, C.J., Lefley, M.: Can genetic programming improve software effort estimation? A comparative evaluation. Journal of Information and Software Technology 43(14), 863–873 (2001)
5. De Barcelos, T.I.F., Simies da Silva, J.D., Sant Anna, N.: An investigation of artificial neural networks based prediction systems in software project management. Journal of Systems and Software 81(3), 356–367 (2008)
6. Donald, J.B., Hilburn, T.B., Hislop, G., Lutz, M., McCracken, M., Mengel, S.: Guidelines for Software Engineering Education. Carnegie mellon University, CMU/SEI-99-TR-032 (1999)
7. Foss, T., Stensrud, E., Kitchenham, B., Myrtviet, I.: A Simulation Study of the Model Evaluation Criterion MMRE. IEEE Transactions on Software Engineering 29(11), 985–995 (2003)
8. Heiat, A.: Comparison of artificial neural network and regression models for estimating software development effort. Journal of Information and Software Technology 44(15), 911–922 (2002)
9. Humphrey, W.: A Discipline for Software Engineering. Addison-Wesley, Reading (1995)
10. Humphrey, W.: Three Process Perspectives: Organizations, Teams, and People. Journal of Annals of Software Engineering 14, 39–72 (2002)
11. Jørgensen, M.: Forecasting of Software Development Work Effort: Evidence on Expert Judgment and Formal Models. Journal of Forecasting 23(3), 449–462 (2007)
12. Jørgensen, M.: A Critique of How We Measure and Interpret the Accuracy of Software Development Effort Estimation. In: 1st International Workshop on Software Productivity Analysis and Cost Estimation, pp. 15–22 (2007)
13. Kalichanin-Balich, I., Lopez-Martin, C.: Applying a Feedforward Neural Network for Predicting Software Development Effort of Short-Scale Projects. In: International Conference in Software Engineering Research and Applications, SERA, pp. 269–275 (2010)
14. Lisack, S.K.: The Personal Software Process in the Classroom: Student Reactions (An Experience Report). In: 13th IEEE Conference on Software Engineering Education & Training, pp. 166–175 (2000)
15. Lopez-Martin, C.: A fuzzy logic model for predicting the development effort of short scale programs based upon two independent variables. Journal of Applied Soft Computing 11(1) (2011)
16. López-Martín, C.: Applying a general regression neural network for predicting development effort of short-scale programs. Journal of Neural Computing and Applications 20(3), 389–401 (2011)
17. MacDonell, S.G.: Software source code sizing using fuzzy logic modelling. Journal of Information and Software Technology 45(7), 389–404 (2003)
18. Paliwal, M., Kumar, U.A.: Neural networks and statistical techniques: A review of applications. Journal of Expert Systems with Applications 36, 2–17 (2009)
19. Park, H., Baek, S.: An empirical validation of a neural network model for software effort estimation. Journal of Expert Systems with Applications 35, 929–937 (2008)
20. Park, R.E.: Software Size Measurement: A Framework for Counting Source Statements. Software Engineering Institute, Carnegie Mellon University, CMU/SEI-92-TR-020 (1992)

21. Pedrycz, W.: Computational Intelligence as an Emerging Paradigm of Software Engineering. In: 14th international conference on Software Engineering and Knowledge Engineering, vol. I, pp. 7–14 (2002)
22. Rombach, D., Münch, J., Ocampo, A., Humphrey, W.S., Burton, D.: Teaching disciplined software development. Journal Systems and Software 81(5), 747–763 (2008)
23. Runeson, P.: Experiences from Teaching PSP for Freshmen. 14th IEEE Conference on Software Engineering Education and Training (2001)
24. Specht, D.F.: A General Regression Neural Network. IEEE Transactions on Neural Networks 7(3), 568–576 (1991)