Software Development Effort Estimation by Means of Genetic Programming

Arturo Chavoya, Cuauhtemoc Lopez-Martin, M.E. Meda-Campaña
Department of Information Systems
University of Guadalajara
Guadalajara, Mexico

***Abstract*—In this study, a genetic programming technique was used with the goal of estimating the effort required in the development of individual projects. Results obtained were compared with those generated by a statistical regression and by a neural network that have already been used to estimate the development effort of individual software projects. A sample of 132 projects developed by 40 programmers was used for generating the three models and another sample of 77 projects developed by 24 programmers was used for validating the three models. Results in the accuracy of the model obtained from genetic programming suggest that it could be used to estimate software development effort of individual projects.**

Keywords—genetic programming; feedforward neural network; software effort estimation; statistical regression

I. INTRODUCTION

The estimation of how long it takes to develop specific software projects is an ongoing concern for project managers [1]. The software development effort estimation can begin with individual projects within academic environments [2], as is the case in this study. There are several techniques for estimating development effort, which could be classified into: 1) expert judgment that aims at deriving estimates based on the experience of experts on similar projects [3][4]; 2) those based on models such as a statistical regression model [5][6]; and 3) those based on techniques from computational intelligence [7], such as fuzzy logic [8][9], neural networks [10] and genetic programming [11].

Considering that no single estimation technique is best for all situations, and that a careful comparison of the results from several approaches is most likely to produce realistic estimates [12], this study compares estimates generated with a genetic programming model against the results obtained with a neural network and with the most commonly used model: statistical regression[4].

Data samples for this study were integrated by 132 and 77 projects for generating (verifying) and validating the models, respectively, and were developed by 40 and 24 programmers, respectively. All of the projects were created following practices of the Personal Software Process (PSP) [13].

The three models were generated from data of small projects individually developed using practices of PSP because this approach has proven its usefulness when applied to individual projects [2].

The hypothesis of this research is the following: Prediction accuracy of a model based on genetic programming is statistically better or equal than a statistical regression model or a model obtained with a feedforward neural network, when these three models are generated from two kinds of lines of code and are applied to the prediction of software development effort of individual projects that have been developed with personal practices.One reason for choosing genetic programming in this work was that this technique is capable of modeling non-linear behaviors, which are common when correlating independent variables with the development effort of software projects [14].

The rest of the paper starts with a section describing the genetic programming algorithm used to generate the corresponding model, followed by a section with the related work. The next section presents the methods used for evaluating the three models, followed bya section on the generation of the models. Respective sections on the verification and validation of the models are presented next. The paper ends with a section of conclusions.

II. GENETIC PROGRAMMING

Genetic programming (GP) is a field of evolutionary computation that works by evolving a population of data structures that correspond to some form of computer programs [15]. These programs typically represent trees varying in shape and size where the internal nodes correspond to functions and the leaves represent terminals such as constants and variable names. The trees can be implemented as the list-based structures known as S-expressions, with sublists representing subtrees.

Fig. 1 presents the flowchart followed by a typical implementation of the GP algorithm [15]. The GP algorithm starts with a population of M randomly generated programs consisting of functions and terminals appropriate to the problem domain. If the termination criterion has not been reached, each program is then evaluated according to some fitness function that measures the ability of the program to solve a particular problem.The fitness function typically evaluates a problem against a number of different fitness cases and the final fitness value for the program is the sum or the average of the values of the individual fitness cases. GP normally works with a standardized fitness function in which lower non-negative values correspond to better values, usually with zero as the best value.

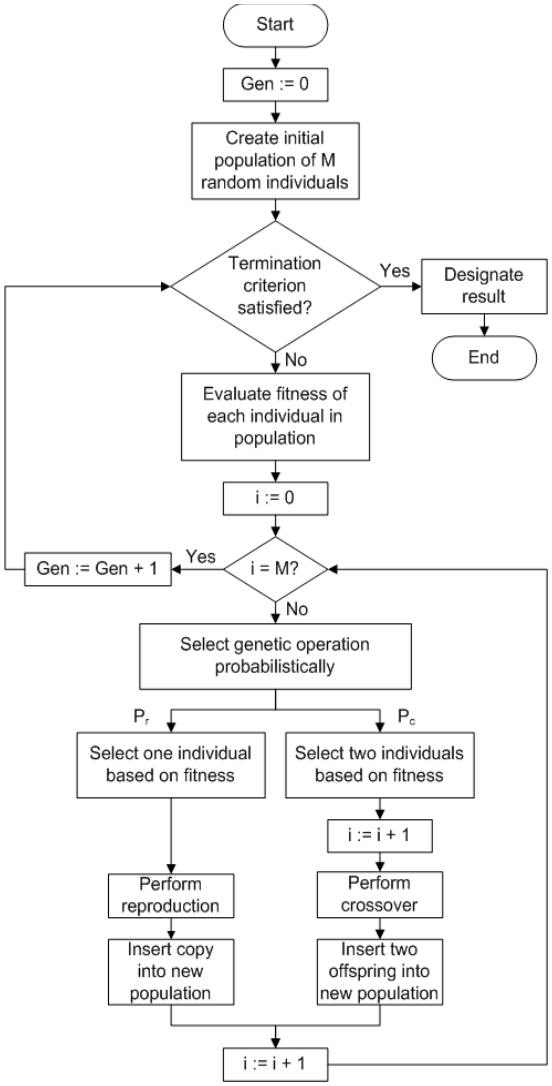

Fig. 1. Flowchart followed by a typical implementation of the GP algorithm.Symbols are as follows: Gen = Generation counter; i = Individual counter; M = Population size; Pr= Probability of reproduction; Pc = Probability of crossover.

After all programs in the population have been evaluated, a selection is made among the individuals in the population to produce the next generation. This selection is usually made proportionate to fitness so that programs with better fitness values have a higher probability of being selected. The Darwinian selection of the fittest individuals in the population is the biological basis on which the various evolutionary computation paradigms are inspired. A number of operations can be applied to selected individuals to provide for variability in the new generation. The reproduction operation consists of selecting a fixed percentage of individuals to pass unchanged to the next generation according to a certain probability of reproduction (Pr). In the crossover operation, two individuals are selected according to a probability of crossover (Pc) to function as parents to produce two offspring programs. In each of the parents a node in the corresponding trees is selected randomly to constitute a crossover point.

The subtrees that have the selected nodes as roots are then exchanged generating two new individuals that are usually different from their parents.

Fig. 2 shows an example of two parental trees before crossover, with the corresponding S-expression below each tree; arrows point at the root nodes of the subtrees chosen to be exchanged, with the corresponding subexpressions shown in boldface.

Fig. 3 presents the generated offspring trees resulting from the exchange of the subtrees in Fig. 2 whose root nodes are pointed at by the arrows. The exchange of subtrees corresponds to the exchange of the sublists shown in boldface below each tree.

A fixed portion of the next generation is produced using the crossover operation, having the possibility of forcing that a fixed percentage of the selected nodes correspond to functions, whereas the rest correspond to either functions or terminals. Unlike genetic algorithms, the mutation operation is normally not necessary in GP, as the crossover operation can provide for point mutation when two nodes corresponding to terminals in the parents are selected to be exchanged.

The process of evaluating, selecting and modifying individuals to produce a new generation is continued until a termination criterion is satisfied. The GP run usually terminates when either a predefined number of generations has been reached or a desired individual has been found.

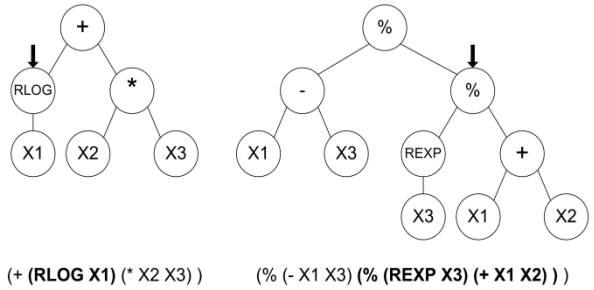

Fig. 2. Example of two parental trees before crossover and the corresponding S-expressions.

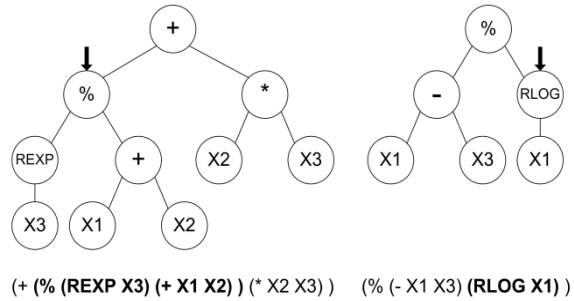

Fig. 3. Offspring trees after crossover and the corresponding S-expressions.

III. Related Work

Results from the application of neural networks and statistical regression have shown that the estimation accuracy of both techniques are competitive with models generated from data of large projects [16][17][18], and of small projects [19].

The accuracy of the genetic programming model used in the present work is compared against the accuracies obtained from the neural network and the multiple linear regression models described in [19]. These two models were generated using data from small-scale projects. The kind of neural network used was a feedforward multi-layer perceptron with a backpropagation learning algorithm (the most commonly used in the effort estimation field [20]). The feed-forward neural network used the Levenberg-Marquardt algorithm due to its reported efficiency [21].

Genetic programming has already been applied to large projects; however, we did not find any study related to its application for predicting the software development effort of small projects developed in laboratory learning environments [22]. Some of the methods reported in previous publications resemble the approach taken in the present work, in which a mathematical model that best fits the data is searched.

The main difference of the present work with most previous reports lies in the genetic programming parameters they used and the data on which they applied the genetic programming algorithm. In [10] a GP algorithm was implemented having a population size of 1000 individuals reproducing for 500 generations during only 10 runs. They used a dataset of 81 software projects that a Canadian software company developed in the late 1980s. They suggested that the genetic programming approach needed further study to fully exploit its advantages. On the other hand, in [23] GP was used with the goal of comparing the use of public datasets against company-specific ones. The techniques they used (GP, artificial neural networks and multiple linear regression) were slightly more accurate with the company-specific database than with publicly available datasets. They used the same GP parameters as in [10]. They concluded that companies should base effort estimates on in-house data rather than on public domain data. In [24] GP was compared against artificial neural networks and multiple linear regression using a number of publicly available datasets. Using less individuals in the GP population (from 25 to 50) than normally employed in the typical implementation of the algorithm (several hundred), they found that although GP was better at effort prediction than neural networks and multiple linear regression with some datasets, in general, none of the techniques they tested rendered a good effort estimation model. These authors concluded that the datasets used to build a prediction model had a great influence in the ability of the model to provide adequate effort estimation. In [25] a different approach was used with GP; instead of finding the mathematical model that best fitted the data, they developed a grammar-based technique they called Grammar Guided Genetic Programming (GGGP) and compared it against simple linear regression. They used the data of 423 software development projects from a public repository and randomly divided them into a training set of 211 projects and a test set of 212 projects. The results obtained using the GGGP technique were not very encouraging, as the effort prediction they found was not very accurate. In [26] GP was also applied for predicting the effort of large projects, and their results showed that GP was better than case-based reasoning and comparable with statistical regression. Finally, GP was applied in [27] using the same methodology as in the present work, but the model found had a slightly higher validation MMER than the model presented here.

IV. METHODS

In this study, the independent variables for all three models were New and Changed (N&C) as well as Reused code, and all of them were considered as physical lines of code (LOC). N&C is composed of added and modified code. The added code is the LOC written during the current programming process, whereas the modified code is the LOC changed in the base project when modifying a previously developed project. The base project is the total LOC of the previous projects, whereas the reused code is the LOC of previously developed projects that are used without any modification [13]. Source lines of code represent one of the two most common measures for estimating software size [28]. Finally, the dependent variable Effort was measured in minutes.

The accuracy criterion for evaluating models in this work was the Magnitude of Error Relative to the estimate for observation *i*, or MER*i*, defined as follows:

$$MER_i = \frac{\left|\text{Actual Effort}_i - \text{Estimated Effort}_i\right|}{\text{Estimated Effort}_i}. \quad (1)$$

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

Another criterion that has been used in the past for evaluating prediction models is the Magnitude of Relative Error (MRE), which is calculated for the *i*-th observation as follows:

$$MRE_i = \frac{\left|\text{Actual Effort}_i - \text{Estimated Effort}_i\right|}{\text{Actual Effort}_i}. \quad (2)$$

The mean of MRE over multiple observations is denoted as MMRE.

The accuracy of an estimation technique is inversely proportional to the MMER or the MMRE. It has been reported that an MMRE ≤ 0.25 is considered acceptable [29]; however, no studies or argumentations supporting this threshold value have been presented [30].Results of MMER in [31] showed better results when compared to other studies; this fact is the reason for choosing MMER as evaluation criterion in the present work.

Experiments for this study were done within a controlled environment having the following characteristics:

- All of the developers were working for a software development company. However, none of them had previously 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. Only the last four of the assignments of each developer were selected for this study. The first three projects were not considered because they had differences in their process phases and in their logs, whereas the last four projects were based on the same logs and had the following phases: plan, design, design review, code, code review, compile, testing, and postmortem.
- 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.
- Developers had already received at least a formal course on the object oriented programming language that they selected to be used through the assignments, and they had good programming experience in that language. The sample for this study only involved developers whose projects were coded in C++ or JAVA.
- Because this study was an experiment with the aim of reducing bias, we did not inform the developers about our experimental goal.
- Developers filled out a spreadsheet for each project and submitted it electronically for examination. This spreadsheet contained a template called "Project Plan Summary", which included the completed data by project. This summary had actual data related to size, effort (time spent in the development of the project) and defects. This document had to be completed after each project was finished.
- Each PSP course was given to no more than fifteen developers.
- Since a coding standard establishes a consistent set of coding practices that is used as a criterion for judging the quality of the produced code [13], the same coding and counting standards were used in all projects. The projects developed during this study followed these guidelines. All projects complied with the counting standard shown in Table I.
- Developers were constantly supervised and advised about the process.
- The code written in each project was designed by the developers to be reused in subsequent projects.
- The kind of the developed projects had a similar complexity as those suggested in [13], and all of them required a basic knowledge of statistics and programming topics learned in the first semesters of an undergraduate program. From a set of 18 individual projects, a subset of seven projects was randomly assigned to each of the programmers. Description of these 18 projects is presented in [19].
- Data used in this study are from those programmers whose data for all seven exercises were correct, complete, and consistent.

TABLE I. COUNTING STANDARD.

Count type	**Type**
Physical/logical	Physical
Statement type	**Included**
Executable	Yes
Non-executable	
Declarations	Yes (one by text line)
Compiler directives	Yes (one by text line)
Comments and Blank lines	No
Delimiters:	
{ and }	Yes

V. GENERATION OF MODELS

Data from 132 individual projects developed by 40 programmers from the year 2005 to the year 2008 were used in the three models: GP, neural network and multiple linear regression. The projects that contained reused code were selected for the sample.

A. Multiple Linear Regression

The following multiple linear regression equation was generated [19]:

$$Effort = 45.06 + (1.20 * N\&C) - (0.30 * Reused). \quad (3)$$

The intercept value of 45.06is the value of the line where the independent variables are equal to zero. On the other hand, the signs of the two parameters comply with the following assumptions related to software development:

- The larger the value of new and changed code (N&C), the greater the development effort.
- The larger the value of reused code, the lesser the development effort.

An acceptable value for the coefficient of determination is $r^2 \geq 0.5$ [13],with this equation having an r^2 equal to 0.58. The ANOVA for this equation had a statistically significant relationship between the variables at the 99% confidence level and the two independent variables were statistically significant at the 99% confidence level.

B. Neural Network

There is a variety of tasks that neural network can be trained to perform. The most common tasks are: pattern association, pattern recognition, function approximation, automatic control, filtering and beam-forming. In the present work, a feedforward neural network with one hidden layer was applied for function approximation. This network had already been trained to approximate a function [19]. The effort was considered as a function of two variables: N&C (number of new and changed lines of code) and Reused (number of reused lines of code).

It has been shown that a feedforward network with one layer of hidden neurons is sufficient to approximate any function with a finite number of discontinuities on any given interval [21]. This is the reason for using a fully-connected feedforward neural network with one hidden layer of neurons in this work. The fully-connected part of the description means that each neuron in a layer receives a signal from each of the neurons in the preceding layer. There were two neurons in the input layer of the network: one received the number of N&C lines of code and the other received the number of reused lines of code. The output layer consisted of only one neuron indicating an estimated effort. The number of neurons in the hidden layer was empirically optimized. A range from two to 40 neurons was explored and the best results were obtained with ten neurons in the hidden layer. The optimized Levenberg-Marquardt algorithm was used to train the network.

The network passed through two phases: training and application. The first group of 132 software projects was used to train the network. This group of projects was randomly separated into three subgroups: training, validation and testing. The training group contained 70% of the projects. The input-output pairs of data for these projects were used by the network to adjust its parameters. The next 20% of data was used to validate the results and identify the point at which the training should stop. The remaining 10% of data was randomly chosen to be used as testing data, to make sure that the network performed well with the data that was not presented during the parameter adjustment.

C. Genetic Programming

A LISP implementation of the GP algorithm was used for generating a model to predict software development effort. The following standard parameters were used on all runs [15]: the initial population consisted of 500 S-expressions randomly generated using the ramped half-and-half generative method. In this method, an equal number of trees are created with a depth that ranges from 2 to the maximum allowed depth (6 in this work) for new individuals. For each depth, half of the programs corresponded to full trees, and the other half consisted of growing trees of variable shape. The maximum depth for individuals after the application of the crossover operation was 17. The reproduction rate was 0.1, whereas the crossover rate was 0.7 for function nodes and 0.2 for any node. Finally, each GP run was allowed to evolve for 50 generations and the individual with the best fitness value was selected from the final generation.

The set of terminals was defined by the two independent variables X1 and X2 corresponding to the New & Changed and Reused lines of code, respectively. Additionally, terminals also consisted of floating-point constants randomly generated from the range [-5, 5).

The set of functions consisted of the arithmetic operators for addition (+), subtraction (–) and multiplication (*), along with the following protected functions shown in prefix notation. To avoid division by zero, the protected division % was defined as follows:

$$(\%\quad x\quad y)=\begin{cases}1 & y=0\\ x/y & y\neq 0\end{cases}. \tag{4}$$

To account for non-positive variable values, the protected logarithmic function RLOG was defined as

$$(RLOG\quad x)=\begin{cases}0 & x=0\\ \ln|x| & x\neq 0\end{cases}. \tag{5}$$

Finally, the protected exponential function REXP was defined as

$$(REXP\quad x)=\begin{cases}0 & |x|\geq 20\\ e^{x} & |x|<20\end{cases}, \tag{6}$$

where the boundary value 20 was arbitrarily chosen to avoid over- and underflows during evaluation.

Since the standardized fitness function f is required to consist of non-negative values, with zero as the best match, this function was defined as

$$f=\sum\left|\text{Actual Effort}_{i}\ -\ \text{Estimated Effort}_{i}\right|. \tag{7}$$

The MMER value was not considered an appropriate fitness measure, as the denominator in the MER formula can yield negative values if the estimated effort in the LISP model is negative itself.

Fifty experiments each consisting of 1,000 GP runs were made. From each experiment, the run with the highest fitness value (lowest f value) was selected and finally an individual program from all runs was selected according to how well it predicted software development effort on both the verification and validation data sets. The selected program from the 50,000 runs is presented next in LISP notation, where X1 is New and Changed code, and X2 is Reused code.

```
(- (- (+ (- X1 (* (- X2 X2) 3.7990248)) (REXP 3.7627742))
     (% (+ (* 2.2606792 X1) (- X2 -4.461488)) (+ (+ X1 X2) X1)))
   (% (+ (% X2 (+ X1 2.2606792)) (- 4.497994 X1))
     (+ (% (% X2 (% -1.1173002 X2)) (+ X1 X1)) (+ X1 2.2606792))))
```

After evaluation of constant subexpressions and simplification of additions involving subexpressions evaluating to zero, the next equivalent program was obtained.

```
(- (- (+ X1 43.06774)
     (% (+ (* 2.2606792 X1) (- X2 -4.461488)) (+ (+ X1 X2) X1)))
```

(% (+ (% X2 (+ X1 2.2606792)) (- 4.497994 X1))

(+ (% (% X2 (% -1.1173002 X2)) (+ X1 X1)) (+ X1 2.2606792))))

VI. VERIFICATION OF MODELS

The GP, the multiple linear regression equation, and the neural network models were applied to the original dataset of 132 projects for estimating effort; then their accuracy by project (MER), as well as by model (MMER), were calculated giving the following results for MMER:

- Genetic Programming = 0.25
- Multiple Linear Regression = 0.26
- Neural Network = 0.25

The following three assumptions of residuals for MER ANOVA were analyzed:

- Independent samples: in this study, groups of developers are made up of separate programmers and each of them developed their own projects, rendering the data independent of each other.
- Equal standard deviations: in a plot of this kind the residuals should fall roughly in a horizontal band centered and symmetrical about the horizontal axis (as shown in Fig. 4).
- Normal populations: a normal probability plot of the residuals should be roughly linear (as shown in Fig. 5).

Once these three residual assumptions had been met, the ANOVA for MER of the projects was calculated, which showed that there was not a statistically significant difference among the prediction accuracy for the three models (*p*-value of Table II is greater than 0.05).

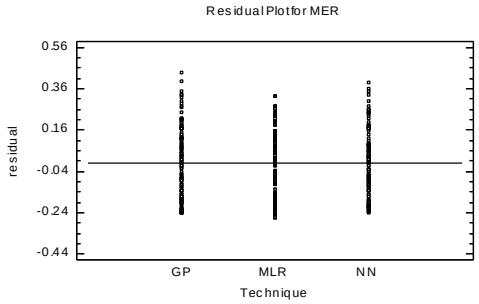

Fig. 4. Equal standard deviation plot of MER ANOVA – verification stage.

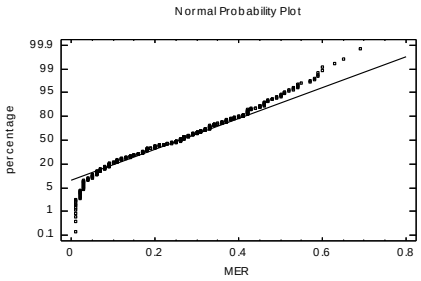

Fig. 5. Normality plot of MER ANOVA– verification stage.

TABLE II. ANOVA TABLE FOR MER BY MODEL (VERIFICATION)

Source	Sum of squares	Degrees of freedom	Mean square	F-ratio	p-value
Between groups	0.0317	2	0.0158	0.60	0.5488
Within groups	10.3644	392	0.0264		
Total	10.3962	394			

VII. VALIDATION OF MODELS

Another group of developers consisting of 24 programmers developed 77 projects through the year 2009. These projects were developed using the same standards, logs, and following the same processes as the 132 programs used for generating the models presented in Section V. Once the three models for predicting effort were applied to these data, the MER by project as well as the MMER by model were calculated yielding the following MMER values:

- Genetic Programming = 0.23
- Multiple Linear Regression = 0.24
- Neural Network = 0.22

An ANOVA for the MMER models (Table III) showed that there was not a statistically significant difference among the accuracy of prediction for the three models (*p*-value is greater than 0.05) at 95% of confidence. Fig. 6 and Fig. 7 show how ANOVA residuals assumptions as described in the previous section are met.

TABLE III. ANOVA TABLE FOR MER BY MODEL (VALIDATION).

Source	Sum of squares	Degrees of freedom	Mean square	F-ratio	p-value
Between groups	0.045	2	0.0226	1.00	0.3703
Within groups	5.0962	225	0.0226		
Total	5.1414	227			

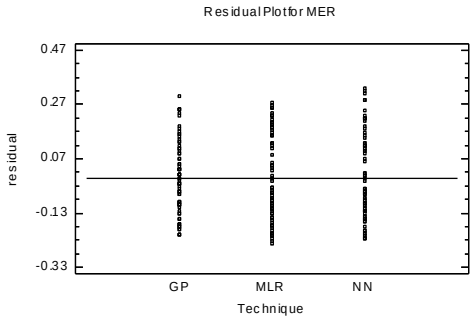

Fig. 6. Equal standard deviation plot of MER ANOVA – validation stage.

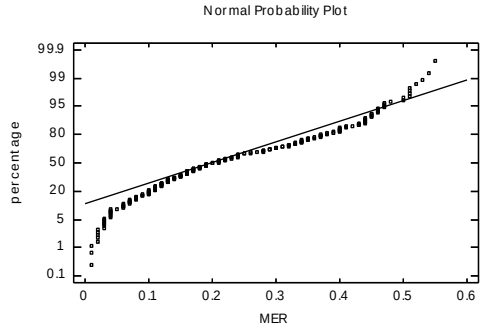

Fig. 7. Normality plot of MER ANOVA – validation stage.

VIII. Conclusion

Taking into account that no single estimation technique is best for all situations, this study compared a GP model with the results obtained from a neural network, as well as from statistical regression.

Data samples integrated by 132 and 77 individual projects for verifying and validating the three models were developed by 40 and 24 programmers, respectively. All the projects were developed following the same practices from the personal software process. The independent variables used in the models were New & Changed code as well as Reused code, whereas the dependent variable was the effort measured in minutes.

The accepted hypothesis in this study was the following: prediction accuracy of a genetic programming model is statistically equal to those obtained from a feedforward neural network, and from a statistical regression model when these three models are generated from two kinds of lines of code and they are applied for predicting software development effort of individual projects that have been developed with personal practices.

Even though we found that the three estimation techniques we tested had a similar power of prediction, GP can have an advantage over other techniques in those cases where specific non-linear functions are suspected to be part of the prediction function, as GP allows the use of any function desired, and the final solution can be a composition of the selected functions.

Future research involves the application of genetic programming for estimating the effort of individual projects involving more independent variables and larger datasets.

Acknowledgment

The authors of this paper would like to thank CUCEA of Guadalajara University, Jalisco, México, ConsejoNacional de Ciencia y Tecnología (Conacyt), as well as Programa de MejoramientodelProfesorado (PROMEP).

References

[1] Jørgensen, M., T. Halkjelsvik, T.: The effects of request formats on judgment-based effort estimation. The Journal of Systems and Software, 83, 29–36 (2010)

[2] Rombach, D., Münch, J., Ocampo, A., Humphrey, W.S., Burton, D.: Teaching disciplined software development. Journal of Systems and Software, 81, 747- 763 (2008)

[3] López-Martín, C., Abran, A.: Applying expert judgment to improve an individual's ability to predict software development effort. International Journal of Software Engineering and Knowledge Engineering 22(4): 467-484 (2012)

[4] Jørgensen, M.: Forecasting of software development work effort: Evidence on expert judgment and formal models. Journal of Forecasting, 23(3), 449-462 (2007)

[5] Boehm, B., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D., Steece, B.: COCOMO II. Prentice Hall (2000)

[6] Kok P., Kitchenhan, B.A., Kirakowski, J.: The MERMAID approach to software cost estimation. In: Proceedings ESPRIT (1990)

[7] Pedrycz, W.: Computational intelligence as an emerging paradigm of software engineering. In: ACM 14th International Conference on Software Engineering and Knowledge Engineering, pp 7-14 (2002)

[8] Lopez-Martin, C., Yañez-Marquez, C., Gutierrez-Tornes, A.: Predictive accuracy comparison of fuzzy models for software development effort of small programs. Journal of Systems and Software, 81(6), 949-960 (2008)

[9] Lopez-Martín, 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), 724-732 (2011)

[10] Wen, J., Li, S., Lin, Z., Hu, Y., Huang, C.: Systematic literature review of machine learning based software development effort estimation models. Information and Software Technology, 54, 41–59 (2012)

[11] Burguess, C.J., Lefley, M.: Can genetic programming improve software effort estimation? A comparative evaluation. Journal of Information and Software Technology, 43, 863-873 (2001)

[12] Boehm, B., Abts, C., Chulani, S.: Software development cost estimation approaches: A survey. Journal of Annals of Software Engineering, 10(1-4), 177-205 (2000)

[13] Humphrey, W.S.: A discipline for software engineering. Addison Wesley (1995)

[14] Hsu C.J., Huang C.Y.: Comparison of weighted gray relational analysis for software effort estimation. Software Quality Journal, 19(1) 165-200 (2011)

[15] Koza, J.R.: Genetic programming: On the programming of computers by means of natural selection. The MIT Press (1992)

[16] Lopez-Martin C., Isaza C., Chavoya A.: Software development effort prediction of industrial projects applying a general regression neural network. Journal of Empirical Software Engineering, 17(6) 738-756 (2012)

[17] De Barcelos Tronto, I.F., Simoes da Silva, J.D., and 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)

[18] 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)

[19] Lopez-Martin, 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)

[20] Park, S.: An empirical validation of a neural network model for software effort estimation. Journal of Expert Systems with Applications, 35, 929-937 (2008)

[21] Haykin, S.: Neural Networks: A comprehensive foundation. Prentice Hall (1998)

[22] Afzal, W. and Torkara, R.: On the application of genetic programming for software engineering predictive modeling: A systematic review. Journal of Expert Systems with Applications, 38(9), 11984-11997 (2011)

[23] Lefley, M., Shepperd, M.J.: Using genetic programming to improve software effort estimation based on general data sets. LNCS, 2724, 2477-2487 (2003)

[24] Dolado, J.J., Fernández, L.: Genetic programming, neural networks and linear regression in software project estimation. In: International Conference on Software Process Improvement, Research, Education and Training, pp 157-171 (1998)

[25] Shan, Y., McKay, R.I., Lokan, C.J., Essam, D.L.: Software project effort estimation using genetic programming. In: Proceedings of the IEEE 2002 International Conference on Communications, Circuits and Systems, 2, pp. 1108-1112 (2002)

[26] Ferrucci, F., Gravino, C., Oliveto, R., Sarro, F.: Genetic programming for effort estimation: An analysis of the impact of different fitness functions. In: The 2nd International Symposium on Search Based Software Engineering, pp. 89-98 (2010)

[27] Chavoya, A., Lopez-Martin, C., Meda-Campaña, M.E.: Applying genetic programming for estimating software development effort of short-scale projects. In: IEEE 2011 Eighth International Conference on Information Technology: New Generations (ITNG 2011), pp. 174-179 (2011)

[28] Sheetz, S.D., Henderson, D., Wallace, L.: Understanding developer and manager perceptions of function points and source lines of code. Journal of Systems and Software, 82, 1540–1549 (2009)

[29] Conte S.D., Dunsmore H.E, Shen V.Y.: Software engineering metrics and models. Benjamin/Cummings Pub Co. (1986)

[30] Jørgensen, M.: A critique of how we measure and interpret the accuracy of software development effort estimation. In: The First International Workshop on Software Productivity Analysis and Cost Estimation (SPACE'07). Information Processing Society of Japan, pp. 15-22 (2007)

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