1-1-2014

Hybrid intelligent model for software maintenance prediction

Abdulrahman Ahmed Bobakr Baqais

Mohammad Alshayeb

Zubair A. Baig
Edith Cowan University

Hybrid Intelligent Model for Software Maintenance Prediction

Abdulrahman Ahmed Bobakr Baqais, Mohammad Alshayeb, and Zubair A. Baig

***Abstract*— Maintenance is an important activity in the software life cycle. No software product can do without undergoing the process of maintenance. Estimating a software's maintainability effort and cost is not an easy task considering the various factors that influence the proposed measurement. Hence, Artificial Intelligence (AI) techniques have been used extensively to find optimized and more accurate maintenance estimations. In this paper, we propose an Evolutionary Neural Network (NN) model to predict software maintainability. The proposed model is based on a hybrid intelligent technique wherein a neural network is trained for prediction and a genetic algorithm (GA) implementation is used for evolving the neural network topology until an optimal topology is reached. The model was applied on a popular open source program, namely, Android. The results are very promising, where the correlation between actual and predicted points reaches 0.91.**

***Index Terms*— Maintenance Prediction, Genetic Algorithm, hyprid AI, Software Maintenance**

I. Introduction

NEURAL network is a computation algorithm resembling the brain. It is a schematic graph of a set of nodes called input, another set of nodes called output and a set of hidden unknown layers connecting both ends. Neural networks are ultimately used for training. The strength of neural networks lies in its capability to be a function approximation. Input and output nodes or neurons are connected with different values of weights by adjusting the weights. Artificial Neural Networks (ANN) is capable of minimizing the error of mapping input to output. Based on the number of neurons in each layer and the number of hidden layers between the input and the output, neural networks are believed to be capable of mapping any function theoretically. The purpose of ANN is used for classification based on test data and mapping function learning.

Evolutionary computing refers to a computer algorithm which has the ability to evolve through multiple runs to optimize a given problem. Evolution indicates that out of the population set, the algorithm can provide a solution space where optimized solutions are presented and inadequate solutions are removed in the process, to be replaced with better ones. Evolutionary computing is based on the fundamentals of the theory of evolution, where the survival is only for the best. Gradually, different terminologies have crept in to the field, such as evolutionary programming, genetic algorithms, and genetic programming. Genetic algorithms are an extension to the concept of evolutionary computing. The genetic algorithm is based on the following main steps:

- Initialization of population based on randomness.
- Computing the fitness function.
- Selection of a solution from the solution pool based on order of the fitness function.
- Applying primitive GA operations for updating the solutions, and
- Stopping at a termination criteria.

Benefits of genetic algorithms include robustness and ease of implementation, but suffer from the number of runs to produce the final solutions as it may stick into local optimum points. 30 different runs are an acceptable minimum [21]. Back propagation of neural network is a "typical delegate"[9]. Neural Network may suffer from local minimum and also from the slowness of convergence. Hence, a combination of GA and ANN may overcome these issues. ANN can be used for training and to build prediction models while GA can be used to speed up the process of ANN by tuning up the design parameters of the ANN [9]. It must be noted that a good design of the model with tuning some parameters can enhance the speed of convergence [2]. Combining neural networks and GA in one model has been investigated and built by different researchers [22]-[26].

The objective of this paper is to build a an artificial intelligence model, based on neural networks and evolutionary computing in a hybrid fashion, in order to obtain high prediction of the software maintenance.

II. Background

Maintenance is an important phase in the software life cycle. Software intrinsically preserves the property of being modified, altered, and improved or corrected [10]. It is inevitable for any software product to undergo the process of

Manuscript received March 18, 2031; revised April 3, 2013. This work was supported the Deanship of Scientific Research (DSR) at King Fahd University of Petroleum & Minerals (KFUPM).

Abdulrahman Ahmed Bobakr Baqais is a PhD at the Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, 31261, Saudi Arabia (corresponding author to provide e-mail: baqais@kfupm.edu.sa).

Moahammad Alshayeb is with the Department of Information and Computer Science, King Fahd University of Petroleum and Minerals, Dhahran, 31261, Saudi Arabia (e-mail: alshayeb@kfupm.edu.sa).

Zubair A. Baig is with the Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran, 31261, Saudi Arabia (e-mail: zbaig@kfupm.edu.sa).

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

maintenance. It has been reported that a high spike of the software project cost lies within the maintenance phase [15] [11]. Hence, there has been a tremendous effort by many researchers to provide means in the form of some measures to predict the maintainability of the software [10], [12], [13] [16], [17]. That is, researchers have been investigating the factors that classify software as maintainable and easily modified. To formalize our discussion of the software maintainability and set a common ground of the concept, the following definitions are presented [13]:

"Maintenance: The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment".

"Maintainability: The ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment".

There are different aspects of maintenance that must be dealt separately to ensure an overall measurement of the process:

"Corrective Maintenance: Maintenance performed to correct faults in hardware or software".

"Adaptive Maintenance: Software maintenance performed to make a computer program usable in a changed environment".

"Perfective Maintenance: Software maintenance performed to improve the performance, maintainability, or other attributes of a computer program".

Another aspect of maintainability that has been proposed in the past few years is prevention. Preventive software maintenance [18] "refers to all activities that are prepared and decided upon regularly, for example annually, in co-operation between the client and the maintainer organizations, and are based on the joint analyses of the present condition as well as the forecasted needs of the software".

III. Previous Work:

To provide a coherent overview of the previous work in predicting maintainability, we are going to present previous effort in measuring and predicting maintainability followed by papers that utilize AI techniques for prediction.

A. Maintainability Measurementand Prediction:

Estimating software's maintainability effort and cost is not an easy task due to the various factors that influence the proposed measurement. It is clear from the above definitions that maintenance is a wide concept that covers many aspects in a heterogeneous manner. That is, the optimality of one maintenance aspect is not orthogonally correlated to other aspects. Nguyen and Boehm [16] assessed the maintenance effort on different aspects of maintenance. The study was directed to answer two hypotheses relating to relationship between maintenance types and effort. The first hypothesis assumes that for different types of maintenance, the productivity required is almost similar. The second hypothesis is related to the effort conjecturing that performing any of maintenance type will demand an even division of effort. The result of the study negates both hypotheses and shows different proportions of effort and productivity to maintenance aspect. This may give a glimpse of the complexity involved in estimating the maintainability effort and cost.

Another reason behind the difficulty of predicting maintainability effort is the time where maintenance activity occurs in the software development life cycle. Maintenance is performed in the last phase of software life cycle and its duration and effort is dynamically affected by the effort and duration of previous phases. Oman and Hagemeister [17] provided a framework to include all of these factors and classified them into different categories. These categories are:

- The procedures and management approaches which have been employed during the course of the software project.
- Environmental factors associated with the hardware and software available in the intended system.
- The intended system itself which is the main concern in this paper.

One of the most popular maintainability metrics is called Maintainability Index (MI) [19]. It was proposed by Software Engineering Institute (SEI) in Carnegie Mellon University. MI is a maintainability metric that predicts the cost of maintainability based on the source code. This is an interesting feature that allows the project team to be able to predict the maintenance effort while developing the code and adjusting the costs accordingly. MI is calculated based on polynomial formula that can be simply calculated based on the code lines, comments and complexity of the code as illustrated below [19]:

$$\left[\begin{array}{l} 171 - 5.2 * \ln(aveV) - 0.23 * aveV(g') - 16.2 * \ln(aveLOC) \\ -50 * \sin(sqrt(2.4 * perCM)) \end{array}\right] \quad (1)$$

The terms are defined as follows:

aveV = average Halstead Volume V per module

aveV(g') = average extended cyclomatic complexity per module (aveLOC = the average count of lines of code (LOC) per module;)

perCM = average percent of lines of comments per module (optional).

In [16] a controlled experiment on the relationship between predicting the effort for maintenance and the maintenance tasks (enhancement, modification or correction). The authors asked subjects to perform different strategies with various maintenance tasks. The experiment was based on a source code for a painter program written in java. The result shows that the estimation effort is not consistent for different kinds of maintenance.

Maintainability can be measured and predicted based on the quality metrics since fewer bugs is an indicator of high quality [20]. The linkage to maintenance is obvious and interlinked. Fewer bugs lead to a reduced number of maintenance effort and then higher quality software. Though maintainability is not constrained to size and number of bugs, the size of bugs detected can indicate the level of maintainability of the software.

The effort spent on writing code is correlated with

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

maintainability [4]. Hayes and Zhao [4] built a prediction model named (MainPredMo). The model takes three metrics into consideration requirement collection effort, designing effort and coding effort. The experiment was conducted and coding effort correlation with maintainability is realized.

Though software development share common process activities and life cycle phases, they differ in the underlying paradigm on which they are going to be built. Many of the maintainability metrics in the literature are targeting functional or structural programming models where extending these metrics to include object oriented software may not be applicable. An effort to study different maintainability metrics and apply it on an object oriented paradigm was presented in [12]. In this study four groups of metrics were assessed to conclude their direct influence on maintainability. Size, cohesion, coupling and inheritance metric groups were empirically analyzed to understand their relationship with maintainability based on one software written in java acquired by the authors. The result shows that size of the code and the coupling metrics are strong candidate to provide an accurate prediction of the maintainability measurement.

B. AI for Maintainability Prediction:

The choice of an AI technique model for maintainability is not solely and necessarily depending on the strength of the model but on the ease of finding prediction and building accurate models [3]. There have been many attempts to build solutions based on GA but unfortunately they can't be applied in practical environments due to the assumption of infinity time or resources. Moreover, some authors justify the applicability of GA to their problems based on seemingly fruitful results without considering the reasons, limits or implicit assumptions in the target problem.

Different attempts have been made to relate GA solutions mechanism to the domain of SE problems [1] [5] [6]. In [1] many software engineering problems have been presented such as scheduling of software projects, testing and verification and risk optimization. Garcia et al. [5] acknowledged the importance of genetic algorithm in assisting software engineering researchers in conducting their experiments. In [6] the author argued that most of GA solutions presented in the SE literature are solely based on experimental data. The need of a solid theoretical proof to support the hypothesis of the applicability of GA is crucial, however, it was left out in most of the papers [1].

In [10] the authors presented a review of the papers that use source code metrics as successful predictors of maintainability index and maintainability predictor based on the size of the code which are very popular for measuring maintainability. In [7] a maintainability predictor based on TreeNet was proposed. TreeNet is commonly known as multiple additive regressions Tree (MART). They have run their experiment on two popular datasets in the maintainability domain known as UIMS and QUES. UIMS is a dataset of classes designed for a user interface system in which there are around 39 classes while QUES dataset contains 71 classes [7]. The paper provides competitive results to other prediction approaches such as Multivariate adaptive regression splines MARS [3], Multiple Linear Regression (MLR), Support Vector Regression (SVR), Artificial Neural Network (ANN) and Regression Tree (RT). In [14] Sharawat applied neural network solution to predict maintainability of OO Program using the common metric: Maintainability Index (MI).

Other researchers use neural Network to predict the quality of software by measuring change effort [8]. Quah and Thwin [8] presented some OO metrics that evidently have a direct impact on the maintainability. These metrics are:

Depth of Inheritance Tree (DIT):
Specifies the level at which the class is built in Inheritance hierarchy [27].
Response for a class (RFC):
Measures the degree at which the class responds to a message [27]
Weighted Method per Class (WMC):
This gives the complexity of a class by measuring the number of methods and properties it has [27].
Message Passing Coupling (MPC):
The numbers of messages are exchanged between objects of a class [8].
Lack of Cohesion in methods (LCOM)
The difference between pairs of methods that don't share any property and pair of methods that have properties in common [27].
Data Abstraction Coupling (DAC)
NOM (Number of Local Methods)
Size1 (Lines of Codes)
Size2 (Number of properties and methods).

IV. Research Methodology

The objective of this paper is to build a an artificial intelligence model, based on neural networks and evolutionary computing in a hybrid fashion, in order to obtain high prediction of the software maintenance.

In this paper we propose a new evolutionary neural network model to predict the maintainability of the software. The proposed model is based on hybrid intelligent technique where neural network is used for prediction and genetic algorithm is used for evolving the NN topology until the optimized topology is achieved. The model is based on Object-Oriented dataset and Maintainability index is used to measure maintainability. Since the dataset contains object-oriented programs, couples of object oriented metrics have been chosen carefully as predictors as explained in the next section.

V. Experiment Setup

Building a prediction model requires identifying the predictors, target output and good choice of a model. We selected an open source project namely Android as a dataset. To extract different object-oriented metrics from each class in this dataset, Metamata tool was used for that purpose. Another tool called JHawk was also utilized for calculating the Maintainability Index. After that, these data were fed to another tool called DTREG to build the predictive model based on AI techniques. In order for the data to be processed by DTREG, it must be preprocessed beforehand and saved

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

in CSV format. After that, four object-oriented predictors were chosen namely: LOC (Line of Codes), NOA (Number of Attributes), NLM (Number of Local Methods) and WMC (Weighted Methods per Class).

For this experiment, we selected 78 classes from Android version 2.3.1. Before feeding these data to our predictive model, several preprocessing steps are required:

A. Extracting Metrics:

All the Android classes have been stripped out from their respective folders and were placed together inside one folder to facilitate the calculation of metrics. Using Metamata 2.0 tool, we were able to obtain values of different object-oriented metrics for each Android Class. Some of these Metrics are not relevant to our model, so we only select 4 metrics as predictors as we described in the previous section.

B. Calculating Maintainability Index:

Maintainability Index is calculated based on the source code of the project. As explained above, the maintainability index is calculated based on the same classes that have been chosen for metrics extraction. In order to do that, a java tool called JHawk has been utilized to calculate MI. Unfortunately, due to the limitation of this tool; it could only produce results for three classes in one run. Considering that Android Version has 78 classes that means at least 26 runs must be performed. In addition, these three classes must be supplied to the tool manually and the experimenter must be careful not to duplicate any classes, omit some or choose different classes. Obviously, this is time consuming, not practical and error-prone. To speed up the process, we have created a small programming file that extracts the intended classes from Android projects in chunks of size 3. Then, the program will place these chunks in different files to be submitted to JHawk serially. All MI values calculated by JHawk are stored and organized in Excel sheet. It must be noted that JHawk provides indication of the level of maintainability by coloring the value as shown below:

- MI < 65 Bad
- 65 < = MI < 85 Good
- 85 >= MI Excellent

C. Data Format:

After getting all the data, we must save them in a CSV format to be readable by DTREG Program.

VI. Results

We ran our experiment several times with different configurations and different parameters to ensure obtaining the best possible result. Table I shows the last configuration of our model while table II shows the obtained result.

VII. Conclusion and Future Work

The result above shows that maintainability index can be predicted with the above specified predictors with high accuracy. It is evident that if we give the experiment more time to evolve the neural network, the result is getting more and more accurate. It would be interesting to see the effect of our model on other types of software programs. In addition, the model is not transparent now and only expert users are able to understand the different parameters. Hence, it would be highly interesting to apply a transparent approach of the model and transform it into a user-friendly tool that can be used easily by ordinary users.

TABLE I
EXPERIMENT CONFIGURATION

Model	Multilayer Perceptron
Node in hidden layer 1	7
Nodes in hidden layer 2	10
Hidden layer activation function	Logistic
Output layer activation function	linear

Model	Genetic Algorithms
Max Generation	100
Generations with no improvement	50
Mutation Scale Factor	0.75
Crossover probability	0.6

TABLE II
RESULTS

Factor	Value
Normalized mean square error	0.276
Correlation between actual and predicted	0.912

We are aiming to extend our work to include other various open source programs such as Eclipse and Net beans. An interesting improvement would be in the transparency of the approach. A new tool with a user-friendly interface should be built to facilitate the maintenance prediction for the users. Other significant enhancements should be in the evolutionary algorithm. It would be better if we can evolve the network for a few hundred generations, so an optimized topology of the network could be obtained.

VIII. Threats to Validity

There are two main threats that may have impact on the results of this study. The first threat is that we used the data of one system, however, we plan to use the data of more systems in future studies.

Another threat is in data collection process. The process of collecting and analysing the data was semi-automated. This may impact the results as human error may occur.

References

[1] H. Jiang, C. K. Chang, D. Zhu, and S. Cheng, "A foundational study on the applicability of genetic algorithm to software engineering problems," in Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, 2007, pp. 2210 –2219.

[2] A. Shukla, R. Tiwari, and R. Kala, Real Life Applications of Soft Computing, 1st ed. CRC Press, 2010.

[3] Y. Zhou and H. Leung, "Predicting object-oriented software maintainability using multivariate adaptive regression splines," Journal of Systems and Software, vol. 80, no. 8, pp. 1349–1361, Aug. 2007.

[4] J. H. Hayes and L. Zhao, "Maintainability prediction: a regression analysis of measures of evolving systems," in Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on, 2005, pp. 601 – 604.

[5] R. E. Garcia, M. C. F. de Oliveira, and J. C. Maldonado, "Genetic algorithms to support software engineering experimentation," in

Empirical Software Engineering, 2005. 2005 International Symposium on, 2005, p. 10 pp.

[6] H. Jiang, "Can the Genetic Algorithm Be a Good Tool for Software Engineering Searching Problems?," in Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International, 2006, vol. 2, pp. 362 –366.

[7] M. O. Elish and K. O. Elish, "Application of TreeNet in Predicting Object-Oriented Software Maintainability: A Comparative Study," in Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on, 2009, pp. 69 –78.

[8] T.-S. Quah and M. M. T. Thwin, "Application of neural networks for software quality prediction using object-oriented metrics," in Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, 2003, pp. 116 – 125.

[9] T. Yan, "An Improved Genetic Algorithm and Its Blending Application with Neural Network," in Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on, 2010, pp. 1 –4.

[10] M. Riaz, E. Mendes, and E. Tempero, "A systematic review of software maintainability prediction and metrics," in Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, Washington, DC, USA, 2009, pp. 367–377.

[11] K. Nishizono, S. Morisaki, R. Vivanco, and K. Matsumoto, "Source code comprehension strategies and metrics to predict comprehension effort in software maintenance and evolution tasks - an empirical study with industry practitioners," in Software Maintenance (ICSM), 2011 27th IEEE International Conference on, 2011, pp. 473 –481.

[12] M. Dagpinar and J. H. Jahnke, "Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison," in Proceedings of the 10th Working Conference on Reverse Engineering, Washington, DC, USA, 2003, p. 155–.

[13] T. Pearse and P. Oman, "Maintainability measurements on industrial source code maintenance activities," in Software Maintenance, 1995. Proceedings., International Conference on, 1995, pp. 295 –303.

[14] S.Sharawat, Software Maintainability prediction using Neural Networks,International Journal of Engineering Research and Applications (IJERA), v2, issue 2, pp 750-755, 2012.

[15] Z. Rana, M. Khan, and S. Shamail, "A comparative study of spatial complexity metrics and their impact on maintenance effort," in Emerging Technologies, 2006. ICET '06. International Conference on, 2006, pp. 714–718.

[16] V. Nguyen, B. Boehm, and P. Danphitsanuphan, "Assessing and Estimating Corrective, Enhancive, and Reductive Maintenance Tasks: A Controlled Experiment," in Software Engineering Conference, 2009. APSEC '09. Asia-Pacific, 2009, pp. 381 –388.

[17] P. Oman and J. Hagemeister, "Metrics for assessing a software system's maintainability," in Software Maintenance, 1992. Proceerdings., Conference on, 1992, pp. 337 –344

[18] R. Vehvilainen, "What is preventive software maintenance?," in Software Maintenance, 2000. Proceedings. International Conference on, 2000, pp. 18 –19.

[19] C4 Software Technology Reference Guide – A Prototype. Software Engineering Institute (SEI), Carnegie Mellon University, 1997. Accessed online via http://www.sei.cmu.edu/library/abstracts/reports/97hb001.cfm

[20] P.Sandhu, S.Khullar, S.Singh, S.Bains, M.Kaur andG.Singh. A study on Early Prediction of Fault Proneness in Software Modules using Genetic Algorithm. World Academy of Science, Engineering and Technology, 2010.

[21] R.Curry, Toward Efficient Training on Large Datasets for Genetic Programming, Master Thesis,Dalhousie University, 2004.

[22] HE Fangguo, Qi Huan. Back propagation neural network based onmodified genetic algorithm and its application. Journal of Huazhong Normal University, 2007

[23] Li Ying, Xu Tao, Xing Wei. Optimization for Neural Network Based on Evolved Genetic Algorithm. Journal of Changchun University of Technology, 2006

[24] Deng Zheng-Hong, Hu Qing, Zhen Yu-Shan.Two-time Training Algorithm of Neural Network Based on Genetic Algorithm. Microelectronics and Computer, 2005

[25] Dam M, Saraf D N. Design of neural networks using genetic algorithm for online property estimation of crude fractionator products.Computers and Chemical Engineering, 2006

[26] Mu A-Hua, Zhou Shao-Lei, Liu Qing-Zhi, Xu Jin. Using Genetic Algorithm to Improve BP Training algorithm. Computer emulation , 2005

[27] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476–493, Jun. 1994.

ISBN: 978-988-19251-0-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)