World Academy of Science, Engineering and Technology
International Journal of Computer and Systems Engineering
Vol:11, No:9, 2017

Multilayer Neural Network and Fuzzy Logic Based Software Quality Prediction

Sadaf Sahar, Usman Qamar, Sadaf Ayaz

***Abstract*—**In the software development lifecycle, the quality prediction techniques hold a prime importance in order to minimize future design errors and expensive maintenance. There are many techniques proposed by various researchers, but with the increasing complexity of the software lifecycle model, it is crucial to develop a flexible system which can cater for the factors which in result have an impact on the quality of the end product. These factors include properties of the software development process and the product along with its operation conditions. In this paper, a neural network (perceptron) based software quality prediction technique is proposed. Using this technique, the stakeholders can predict the quality of the resulting software during the early phases of the lifecycle saving time and resources on future elimination of design errors and costly maintenance. This technique can be brought into practical use using successful training.

***Keywords*—**Software quality, fuzzy logic, perceptron, prediction.

I. Introduction

OF all the non-functional requirements of any software system, the quality of that system and its reliability has acted as a magnet for research. If the management and the stakeholders have some knowledge of the quality of the system being developed beforehand, it will help them to save the cost and resources in the later stages of the software development lifecycle. These expenses may include fixing the design errors, elimination of architectural loopholes, etc.

In order to get this motive, a software quality prediction technique comes in handy. This technique will help us to develop a system which is flexible enough to cater for any changes made during the lifecycle of the software development. These changes may include the properties of the development process the characteristics of the target product as well as the environmental changes for correct operation.

Some related work already done in the related field has been discussed in Section II. Section III gives the background information about the proposed model. The next section explains the proposed methodology in detail. After that, some results are discussed after applying our proposed methodology on small scale. Finally, the research has been concluded, and the future work aimed is given in the last section of this research paper.

Sadaf Sahar, Usman Qamar and Sadaf Ayaz are with the Department of Computer Engineering College of Electrical & Mechanical Engineering, National University of Sciences and Technology, Pakistan (e-mail: Sadaf.sahar@ceme.nust.edu.pk, usmanq@ceme.nust.edu.pk, Sadaf.ayaz@ceme.nust.edu.pk).

II. Related Literature Evaluation

In order to make the software profitable, the entire process of software development should revolve around the fact that a good quality software should hit the market in a reasonable amount of time. To achieve that, the developers must have sound knowledge of the modules or the areas in the software which are likely to have faults or become the reason of failure of the system. There is a lot of literature present in which the authors have tried to categorize as to which modules should be put under the fault-prone heading. There are numerous techniques which include classification using fuzzy logic, classification trees and logical regression, etc. [1]-[3]. These authors have proposed various methods including the use of metrics in order to identify the fault prone areas. This helps the developers and testers to focus their attention not only on the development of the modules but also on the validation and verification of these modules as well. Once the module is identified as fault-prone, major attention should be provided to this module along with allocation of resource and time to the fault fixing. Apart from this, there are other methods devised by authors for the prediction of the quality of the target system. These methods make use of artificial neural networks (ANNs) using fuzzy logic [4], [5] and support vector machines (SVMs) [6]. There is still a lot of research going on to test the accuracy of these techniques. This is because the entire software development process is itself so complex that checking the performance of a technique becomes a tedious job. Along with this during the development lifecycle, the management may encounter changes in the working environment or changes in the requirements from the clients.

Fenton and Neil thought that there is insufficiency of algorithms which can efficiently predict the quality of the software hence they proposed a methodology based on Bayesian Belief Network. They modelled their approach in a way that it would take more than one factors which are likely to affect the quality of the software and output them on multiple streams showing the quality attribute of the target software [7].

Since it is almost impossible to take all the project related attributes into consideration, Khoshgoftaar and Munson proposed an attribute analysis technique to identify the important attributes in their research [12]. Also, much effort has been done by many other researchers in defining an effective product metrics. Chidamber and Kemerer proposed a complete suite for the development of product metrics for Object Oriented Designs [13]. Similar work has been done by Henderson-Sellers which calculates the complexity attribute of products [14]. Likewise, Li and Henry proposed the

maintenance metrics for the similar purpose [15].

With the development of many new proposed models related to quality predictions, much work has been done for the validation purpose of these models. As Bandi et al. predicted the performance of maintenance of an Object-Oriented Design by using complexity metrics [16]. Similar work has been done by Basili et. al. who performed the validation of Design Metrics for the quality prediction [17]. Yu et al. also did an industrial case study research for the prediction of fault proneness [18]. Gyim´othy et al. also tried to validate the metrics by doing fault prediction on an open source software [19]. Similarly, Subramanyam and Krishnan tried to validate the metrics by performing some analysis [20].

Although there is much work done for the development and validation of quality prediction models, but there is not a single model which can completely and accurately tell the quality of the product before time. We have tried in this paper to provide with a new quality prediction which can alone be used for the prediction of product quality with maximum probability and accuracy.

III. Perceptron & Fuzzy Logic

A. Learning Sequence of Perceptrons

A learning sequence is an adaptive algorithm by the help of which a web of inputs and other computing units arrange themselves in an organized fashion in order to achieve a requisite output. This behavior can be achieved by training the system for the scenarios to which we know the output to. By this way, the system will start to learn how to get the functionality that is required from the system. There is an embedded corrective step which works iteratively throughout the working of the system to make sure that the actual output of the system is the expected output as well. After performing the corrective step, the parameters are fed back to the system, hence forming a closed loop.

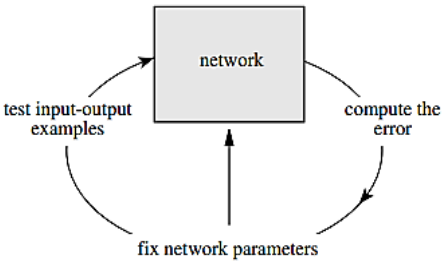

Fig. 1 Closed loop learning and correction of parameters

In most of the cases of perceptron, learning the computing units are associated with different weights. These weights are liable to an update based on sequential stochastic tests. It is to be noted here that learning is a process which achieves the desired results by continuous update of weights and parameters. Any algorithm which jumps to conclusions blindly cannot be termed as learning. It can only be called a learning algorithm if the parameters of the network are updated within a closed loop and the results are obtained with the help of previous experience.

Fig. 2 shows a single layer neural network in the form of a diagram. Inputs **p1** and **p2** have their corresponding weights. At **b,** a weighted sum **Wp** of these inputs is calculated and the sum is fed for thresholding. After the limiting is done, the result **a** is classified into one of the resultant classes available. There could be more than two inputs and more than one output classes for multilayer Neural Network.

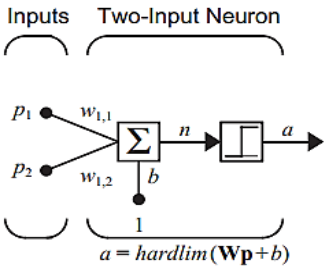

Fig. 2 Neural Network with two inputs and one output

B. Why Use Fusion of Fuzzy Logic and Perceptron

Perceptron (Artificial Neural Networks) and fuzzy logic (FL) are both vastly emerging field in the category of artificial intelligence as they help in solving complex real life problems in a self-explanatory and organized manner. The learning efficiency and reasoning quality of the neural networks makes it possible for them to solve large complex problems. For very large and architecturally complex systems, it is nearly impossible to obtain a single mathematical model. In such cases, neural networks and fuzzy logic come in handy which do not require any tedious mathematical model for system computation [8], [9]. It is because of these advantages that ANNs and fuzzy logic are widely used for almost all domains implementing their self-organizing properties.

C. Limitations of This Technology

With the immense amount of help that ANNs and fuzzy logic provide, unfortunately there are some limitations which have to be catered for. For example:

- ANNs are also termed as “black boxes”. This is due to the fact that the original model of the entire system is termed as weights and inputs, making it hard to comprehend and block out the functions of the system.
- It is difficult, almost impossible to find out the size and scale of the neural network before starting the computation and modelling.
- The perceptron and neural networks have a problem of scalability. For each increase in the input, the entire training and testing have to be carried out again and weights get modified in order to adapt to the change in the environment.
- When the size of the system increases, it is very difficult to identify the useful inputs and functions to be implemented and which of them are redundant and will have no impact on the output of the system.
- For every new system that has to be developed the algorithm for prediction and classification has to be defined. There are no predefined standard functions that can be embedded for all. The accuracy is maximized by

rigorous training and thorough testing of every input and weight and the corresponding output that they produce.

Apart from all the limitations of the fuzzy logic, the biggest and one of the key advantages of it is that it can be easily implemented in a linguistic fashion, i.e. it is easy to program, hence reducing the computations by hand.

IV. Proposed Method for Software Quality Prediction

In this section, we have tried to present a novel technique for the prediction of the quality of the software product as it matters a lot in the success of a product. Software development is a vast process, and to get all the software and environment attributes under one umbrella, it is a very difficult process as there exist many attributes which are almost impossible to be considered and also it is very difficult to specify a reasonable value for many of the attributes. Hence in this section, we have tried to explain the working of our software model in a way that is easier to comprehend. We have used only three inputs and only two outputs to reduce the complexity and to make it easy to understand.

A. Preparing the System

We are all familiar that there are countless quality attributes that can be associated with the software product. These quality attributes can also be termed as non-functional requirements and can include reliability, efficiency, portability, usability, supportability & maintainability, etc. If all these attributes were to be taken for the demonstration of our model, it would have increased the complexity manifold as it will increase the complexity.

When we come across most of the software products the major qualities, what we look for in them is the reliability and efficiency. In order to make our model understandable, we have taken two quality attributes, i.e. reliability and efficiency. These attributes will be the decision factors of the quality of the software product.

Every perceptron has to have a set of inputs to evaluate the output. Each of these inputs is associated with a weight which tells us how much is this input contributing to the output of the fuzzy neural network [10], [11]. Inputs to our model are selected on the basis of those attributes which will be available to us during the high level and low level design of our software product. For the inputs, we have chosen complexity (both structural and functional), reuse and depth of inheritance tree (also known as DIT). Complexity of the design can be structural or functional. It is the measure of the amount of information that a particular software design represents. The best software is termed to be the one which can achieve has the probability to achieve more goals in a specified amount of time. Code reuse is the measure of how much the architecture of the system or design of it can be used to base other releases. Depth of inheritance tree tells us how far a class is down in the hierarchy of the system and how many other classes or functions it is dependent on.

B. Training of the System

After having the knowledge of our inputs to the system and the expected outputs, the next step is to train our models with some sample data that can be acquired for systems which already exist is the market or by the experience of the developers in the organization. Each of the inputs is assigned weights and can be categorized into three types, i.e. low, medium, and high. The weights of the inputs range normally from 0 to 1, closer to 0 being very low and closer to 1 being high. The range 0.4 to 0.6 is considered to be medium.

Each input can hold one of the three states for a given software system. For example, if the reuse value for a system is 0.9000 then the value of this input will stand at the high branch of the input layer. Similarly, if the value of reuse is 0.2, then this input belongs to the Low Branch of Input Layer. The structure of the system is given in Fig. 3.

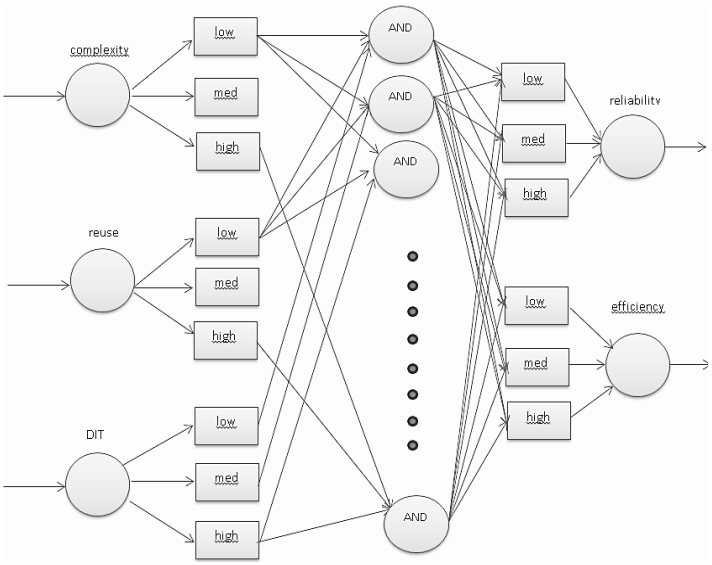

Fig. 3 Structure of the model

As there are three input nodes and each has three possible values, the possible number of fuzzy logic values will be 3x9=27. All the permutations of these inputs are shown in the table below.

TABLE I
POSSIBLE PERMUTATIONS FOR INPUT VALUES

	COMPLEXITY	REUSE	DIT
1.	Low	Low	Low
2	Low	Low	Medium
3	Low	Low	High
4	Low	Medium	Low
5	Low	Medium	Medium
6	Low	Medium	High
7	Low	High	Low
8	Low	High	Medium
9	Low	High	High
10	Medium	Low	Low
11	Medium	Low	Medium
12	Medium	Low	High
13	Medium	Medium	Low
14	Medium	Medium	Medium
15	Medium	Medium	High
16	Medium	High	Low
17	Medium	High	Medium
18	Medium	High	High
19	High	Low	Low
20	High	Low	Medium
21	High	Low	High
22	High	Medium	Low
23	High	Medium	Medium
24	High	Medium	High
25	High	High	Low
26	High	High	Medium
27	High	High	High

Note: the values of all the inputs used in this model are normalized between 0 & 1.

For training of the data, we have two samples for which we know the output of. This means that after running these samples from the system we can compare the actual output to the output generated by our model. Sample 1 says that when Complexity & DIT are high and Reuse is of medium level, then the resulting efficiency should be medium and reliability should be high. Sample 2 says that if all the input variables are of low level, then the system will be highly efficient and moderately reliable. These inputs along with the computation with their weights are fed to layer number 3 where the ruling nodes are present. The higher the weight of the input the greater will be the fire power of the ruling node. After the multiplication of the ruling node with their weights, these products are fed into the final layer/ decision layer.

Each output node is also an attribute and can have three possible values, i.e. high, medium, and low. Among these three branches, the one which has the highest product will be the resulting quality attribute. For example, if the highest product for reliability is at the medium branch, we will say that the reliability of the software product will be medium. If the actual quality attribute will be the same as the one predicted, no action will be taken. But if the predicted results differ from the actual results, the values of the weights for the ruling nodes will be updated using the equation.

$$W_{AB(new)} = W_{AB(old)} - \eta \frac{\partial E^2}{\partial W_{AB}}$$

where η is the learning rate of the system. The smaller the value of it the slower will be the learning, and hence, the model will be thorough. $\frac{\partial E^2}{\partial W_{AB}}$ is the differential of error to the old weights.

Error is described as the difference of actual output and predicted output. Two of the samples for which the system was trained are as follows.

TABLE II
EXAMPLE OF SAMPLES FROM THE TRAINING DATA

Quality attributes			**Expected output**		**Actual output**	
complexity	reuse	DIT	reliability	efficiency	reliability	efficiency
0.9951	0.5714	0.9600	0.9000	0.4000	0.9071	0.4415
0.1667	0.1429	0.0011	0.6000	0.9000	0.5812	0.8600

Table II makes it quite clear the proximity of the results which were found by training the system and the actual results. Now, when the system is done training itself, all the weights are updated according to it. We can say that the state of the system is now in a fixed mode and we can test the system with new sample data.

C. Testing of the System

Once all the training is done and its result is less than 0.1% of error, we can then begin to test the system for new samples. Testing for the system is done in a similar manner as training. The only difference is that, in training, the weights which rendered the output incorrect had to be updated. In case of testing, the system is in a fixed state and, so are the weights with rigorous training. Testing only involves giving the values of the inputs and checking the output.

For testing, two samples were fed to the system. The values of complexity, DIT, and reuse were given as inputs. The system was run with these inputs, and the consequent output was tested for the quality attributes, i.e. reliability and efficiency.

Test sample 1 was considered to be software which has low complexity, low reuse of code and moderate depth of inheritance tree. Test sample 2 was software which had low complexity and low reuse and DIT. Statistically if we observe, the test sample 1should yield moderately efficient and reliable system. As for test sample 2, the reliability should be moderate and efficiency should be high.

V. RESULTS AND DISCUSSION

Making the test samples run through the system, we came up with the following results

Looking at the results of the model that is proposed, we find the need to elaborate the features of this model as well. There are a number of reasons why this technique can be rendered

useful in prediction of the quality of the target software under development.

TABLE III
TEST RESULTS

Quality attributes			Actual output	
complexity	reuse	DIT	reliability	efficiency
0.2500	0.2857	0.5900	0.5177	0.4632
0.0833	0.2500	0.0073	0.5168	0.8334

- This system can take multiple inputs for computation and can yield multiple outputs making a strong connection between the factors which affect the quality of the software and the actual quality attributes of the system. Multiple input-Multiple output system shows that it can take into account countless factors which contribute to change in the quality of the software, and on the other hand, it enables us to take as many quality attributes for evaluation as we want.
- The prediction done by this model can be counter-validated using knowledge based experience. It is a common experience that, when the number of classes and modules for the software system is big, the efficiency of the system is low and it will make the maintenance of the system very tedious. Hence using the knowledge of previously developed software and those which are deployed in the market, we can validate the quality of those systems using this model.
- The model which is proposed is flexible in a way that, in the early stages of the software development, it is difficult to know all the factors which will affect the quality attributes of the target system. This system will adapt to all the changes in the system and can take is as many inputs during any phase of the software development life cycle.

VI. CONCLUSION AND FUTURE WORK

In this paper, we have proposed a novel approach for the early prediction of software quality. This technique is developed with the amalgamation of concepts of artificial intelligence and software quality engineering. The proposed system uses multilayer neural networks to correctly predict the quality of the target software system. The factors which affect the quality of that software system are fed to the model as inputs, and doing the necessary computations, it gives us the level of the quality attributes of the system. The key benefit of this proposed methodology is its ease of use and flexibility as it can take any number of inputs and any number of decision making quality attributes. In future, we will try to validate the proposed model by applying it on a real time product development and will try to make it useful for the types of projects for which the training data could not be available already, e.g. for the development of a new product.

REFERENCES

[1] T. M. Khoshgoftaar and N. Seliya "Analogy-based practical classification rules for software quality estimation", Empirical Software Engineering, 8(4): 325-350, 2003.

[2] T. M. Khoshgoftaar, Y. Liu and N. Seliya "A multiobjective module-order model for software quality enhancement", *IEEE Transactions on Evolutionary Computation*, 8(6): 593-608, 2004.

[3] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones and J. P. Hudepohl "Classification-tree models of software-quality over multiple releases", *IEEE Transactions on Reliability*.

[4] T. M. Khoshgoftaar, R. M. Szabo and P. J. Guasti "Exploringthe behaviour of neural network software quality models", Software Engineering Journal, 10(3): 89-96, 1995.

[5] M. M. T. Thwin and T. S. Quah "Application of neural networks for software quality prediction using object-oriented metrics", *Journal of Systems and Software*, 76(2): 147-156, 2005.

[6] S. S. So, S. D. Cha and Y. R. Kwon "Empirical evaluation of a fuzzy logic-based software quality prediction model", *Fuzzy Sets and Systems*, 127(2): 199-208, 2002.

[7] N. E. Fenton and M. Neil "A critique of software defect prediction models", *IEEE Transactions on Software Engineering*, 25(5): 675-689, 1999.

[8] E. Khan, *Neural Fuzzy Based Intelligent Systems and Applications*, in *Fusion of Neural Networks, Fuzzy Systems and Genetic Algorithms: Industrial Applications*, by L.C. Jain and N.M. Martin, Chapter 5, CRC Press, 1998.

[9] M. B. Ghalia and A. T. Alouani "Artificial neural networks and fuzzy logic for system modeling and control: a comparative study", in *Proceedings of the 27th Southeastern Symposium on System Theory*, pp. 258-262, March 1995.

[10] C. T. Lin and C. S. G. Lee "Neural-network-based fuzzy logic control and decision system", *IEEE Transactions on Computers*, 40(12): 1320-1336, 1991.

[11] C. J. Lin and C.T. Lin "An ART-based fuzzy adaptive learning control network" *IEEE Transactions on Fuzzy Systems*, 5(4): 477-496, 1997.

[12] J. C. Munson and T. M. Khoshgoftaar, "The dimesionality of program complexity," in Proceedings of the 11th International Conference on Software Engineering, pp. 245-253, Pittsburgh, PA, May 1989.

[13] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," IEEE Transactions on Software Engineering, vol. 20, no. 6, pp. 476–493, 1994.

[14] B. Henderson-Sellers, Object-oriented metrics: measures of complexity.Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1996.

[15] W. Li and S. Henry, "Maintenance metrics for the object oriented paradigm," in IEEE Proceedings of the First International Software Metrics Symposium, May 1993, pp. 52–60.

[16] R. Bandi, V. Vaishnavi, and D. Turk, "Predicting maintenance per-formance using object-oriented design complexity metrics," Software Engineering, IEEE Transactions on, vol. 29, no. 1, pp. 77–87, Jan. 2003.

[17] V. R. Basili, L. C. Briand, and W. L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators," IEEE Trans. Softw. Eng., vol. 22, no. 10, pp. 751–761, 1996.

[18] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using oo met-rics. an industrial case study," Software Maintenance and Reengineering, 2002. Proceedings. Sixth European Conference on, pp. 99–107, 2002.

[19] T. Gyim ´ othy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," IEEE Trans. on Software Engineering, vol. 31, no. 10, pp. 897–910, 2005.

[20] R. Subramanyam and M. S. Krishnan, "Empirical Analysis of CK Met-rics for Object-Oriented Design Complexity: Implications for Software Defects," IEEE Trans. Softw. Eng., vol. 29, no. 4, pp. 297–310, 2003.