Simplified Neural Model for the Software Development Team Optimization

Madhu S. Nair and Jaya Vijayan
Rajagiri School of Computer Science, Rajagiri College of SocialSciences, India.

Abstract: *A simplified neural model to optimize a project team and to attain maximum throughput as well as to obtain high quality software has been proposed here. A novel approach, which uses the concept of Artificial Neural Networks, to train the software professionals and make them perform at high level of standards, is adopted. In this approach, a high level of communication among the professionals is achieved which will lead to good team work and finally produce quality software that meets the required level of standard.*

Keywords: *Neural network, optimization, training, weights, acceptance.*

Received August 8, 2006; accepted February 15, 2007

1. Introduction

The optimization of the project team in a software development organization can be effectively performed using an optimization process by applying Artificial Neural Network (ANN) concept. The *Supervised* learning method of ANN has been adopted in this model to improve the performance of the project team members. The following are the two important phases involved in the training procedure adopted in the model:

1. Team formation and project preparation phase.
2. Team optimization phase.

Out of the above two phases, the second phase uses the proposed *Project Team Optimization Model*.

1.1. Team Formation and Project Preparation Phase

This phase mainly consists of two major activities, team formation, and project preparation and scheduling.

1.1.1. Team Formation

Initial activity is categorization of recruited candidates into different groups and formation of the team. Each team will have a project leader to guide and control the team. Team formation is an important activity since the success of the project depends on the teamwork. Following are the aspects that are taken into consideration while the team formation takes place:

a) Level of skills of each trainee.
b) Level of background knowledge.
c) Area of interest (like analysis, design, coding, testing etc.).
d) Soft skills.

The skill levels can again be categorized into high-level, medium-level and low-level. This approach will ensure that there is a mix and match of all the three levels of skills in the project teams. Another objective behind this approach is to make the total average skill level of every team approximately the same.

1.1.2. Project Preparation and Scheduling

A project training set representing all categories of projects that a particular organization is concentrating on is prepared. Each project specified in the training set should be a miniature of the real time projects. The training set should also contain the description of all projects & the minimum time period required for completion as well as the excepted output value for each project.

Once the training set and teams are formed the schedule for training the teams are to be prepared. The aspects that are to be considered while scheduling the projects are:

- Duration of the training.
- Tools or resources that are to be provided for training.

1.2. Team Optimization Phase

The proposed project team optimization model is applied in this phase which will lead to improved performance of the team members by making them will versed in all categorized of projects. This phase is initiated by assigning the projects specified in the train set one by one and continued until the team gets optimized. Once the team gets optimized by applying the proposed model the team members can work effectively and efficiently in all categorizes of real time projects to obtain maximum throughput.

2. Project Team Optimization Model

The proposed model is a supervised learning model in which different projects selected from the training set is assigned to the corresponding team one by one, after which the actual output of the project will be compared with the expected output specified in the training set. After the comparison if there is any deviation from the expected output the team members have to rework on the project by identifying the flaws in the development activities, until the expected performance is attained.

After attaining the desired performance in a particular type of project the team will be assigned the next category of project from the training set. This process is continued until the team gets optimized and shows good performance in all the different categories of projects specified in the training set.

2.1. Related Work

A similar neural network approach has been efficiently used for software risk analysis [10]. After identifying the key software risk factors responsible in achieving successful outcome, a neural network approach has been used to establish a model for minimizing the risks attributed to failed projects. Inputs of the model are software risk factors that were obtained through interview, and output of the model describes the final outcome of the project. The experimental result indicates that the software risk analysis can be improved through these methods and that the risk analysis model is effective.

2.2. Optimization Model Architecture

The proposed optimization model uses single layer feed forward perceptron topology. The architecture consists of one input layer and one output layer. The learning algorithm uses the training set prepared in the project preparation phase. Based on the training set, the weights have to be modified to make the model stable. Once the model becomes stable, it can be utilized to optimize the software project team using the acceptance factor of the output layer, which indicates the success rate of the project. The detailed architecture is shown in Figure 1. The input layer of the model consists of eight input nodes. Let X = $\{x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8\}$ be the input vector and W = $\{w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8\}$ be the weight vector respectively.

The activation function used in the model is non-polynomial continuous bounded function (tauber-wiener) particularly non-linear hyperbolic tangent (sigmoid) function. Let *O* be the activation value of the output scalar and let *b* be the bias value of the model. The activation function for the above architecture using forward computing is given in Equation 1.

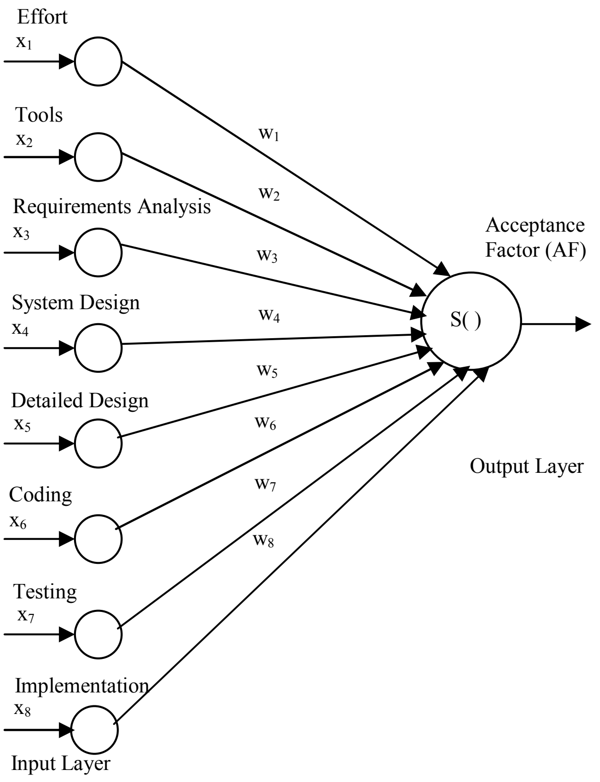

Figure 1. Feed forward perceptron model.

$$O = S\left(\sum_{i=1}^{8} w_i x_i + b\right) \tag{1}$$

The training sample for the p^{th} project in the training set is given by (x_p, d_p), where x_p is the input pattern and dp is the desired output pattern. Let O_p be the actual output pattern obtained using the activation function given above. A supervised, sequential, error driven, linearly separable and generalized delta rule for back propagation learning is used in this model. The objective of learning is to obtain the incremental values so as to minimize the sum square error.

$$E = (O_p - d_p)^2 \tag{2}$$

If η is the learning rate then

$$\Delta w_i = \eta(d_p - O_p)x_i \tag{3}$$

The weights can then be modified using the following equation:

$$w_i^{improved} = w_i^{old} + \Delta w_i \tag{4}$$

2.2.1. Input Layer

The input layers of the proposed model consist of 8 input nodes which accept values from different phases of software development as inputs. These values are inturn used to compute the acceptance factor of output layer, which gives an indication to the success rate of the project.

a. Effort

The first input parameter to the model is the *effort*. Effort is estimated using the formula, $E = a \times (KDLOC)^b$, where a and b depends on the type of the project.

The projects are categorized into three types organic, semidetached and embedded [2]. Organic projects are used in an area in which the organization has considerable experience and requirements are less stringent. Semidetached systems include developing a new Operating System (OS), Database Management System (DBMS) etc.,. Embedded projects are used in an area in which the organization has little experience and stringent requirements for aspects such as interface and reliability. The constants a and b for different systems are given in Table 1.

Table 1. Effort metric.

System	***a***	***b***
Organic	3.2	1.05
Semidetached	3.0	1.12
Embedded	2.8	1.20

b. Tools

The second input parameter to the model is the type of the *tool* used for the development. The use of software tools is categorized as*very low* , *low*, *nominal, high* and *very high*. The effort multipliers for tools are given in Table 2.

Table 2. Effort multipliers for tools.

	Rating				
	Very Low	**Low**	**Nominal**	**High**	**Very High**
Tools	1.24	1.10	1.00	0.91	0.83

c. Requirements Analysis

Optimization model uses *requirements analysis* as the third input parameter. The most commonly used metric in the requirements analysis phase is the *Function Point metric* (FP) [3].

FP are derived using an empirical relationship based on countable measures of software information domain and assessments of software complexity. The original formulation for computing the function point used the count of five different parameters namely *number of user inputs*, *number of user outputs*, *number of user inquiries*, *number of files* and *number of external interfaces*. The five parameters are determined and counts are provided in the appropriate table location. The count total is determined by summing up the count for each parameter. The weighting factor for each parameter is shown in the Table 3.

Table 3. Weight factor for FP metric.

Parameter	**Weighting Factor**		
	Simple	**Average**	**Complex**
No. of User Inputs	3	4	6
No. of User Outputs	4	5	7
No. of Inquiries	3	4	6
No. of Files	7	10	15
No. of External Interfaces	5	7	10

d. System Design

The fourth input to the optimization model is the *system design* parameter. The metric *total number of modules* [5] is the simplest and the commonly used metric in the system design phase.

The metric value can be easily obtained from the design by using an average size of a module. From this metric the final size in Lines Of Code (LOC) can be estimated [4]. Alternatively, the size of each module can be estimated, and then the total size of the system will be estimated as the sum of all the estimates. As a module is a small and clearly specified programming unit, estimating the size of a module is relatively easy.

e. Detailed Design

The proposed model accepts *detailed design* parameter as the fifth input. The metric widely used in the detailed design phase is the *information flow metric* [8].

In the information flow metric, the complexity of a module is considered as depending on the intra-module complexity and inter-module complexity. The intra-module complexity is approximated by the size of the module in LOC. The inter-module complexity of a module depends on the total information flowing in the module (inflow) and the total information flowing out of the module (outflow). The inflow of a module is the total number of abstract elements flowing in the module and outflow is the total number of abstract data elements that are flowing out of the module. Module design complexity:

$$Dc = Size \times (inflow \times outflow)^2 \qquad (5)$$

To identify modules that are extra complex, what complexity number is normal has to be defined. The complexity of modules in the design and highlight modules that are relatively speaking more complex has to be evaluated. One of the method used for highlighting the modules are as follows: Let average complexity be the average complexity of the modules in the design being evaluated and let standard deviation be the standard deviation in the design complexity of the modules of the system. This method classifies the modules into three categories: *error prone, complex* and *normal*. If Dc is the complexity of a module, it is classified as follows:

- *Error Prone*, if *Dc* > average complexity + standard deviation.
- *Complex*, if average complexity < *Dc* < average complexity + standard deviation.
- *Normal*, Otherwise.

f. Coding

The proposed model accepts c*oding* parameter as the sixth input, and the metric used for coding phase is halstead measure [7]. A number of variables are defined in the software science. These are n_1 (number of unique operators), n_2 (number of unique operands), N_1 (total frequency of operators) and N_2 (total frequency of operands). As any programs must have at least two operators, one for function call and one for end of statements, the ratio $n_1/2$ can be considered the relative level of difficulty due to the larger number of operators in the program. The ratio N_2/n_2 represents the average number of times an operand is used. In a program in which variables are changed more frequently, this ratio will be larger. As such programs are harder to understand, ease of reading or writing is defined as

$$D = (n_1 \times N_2) / (2 \times n_2) \tag{6}$$

Halstead's complexity measure focuses on the internal complexity of a module. A module's connection with its environment is reflected in terms of operands and operators. A call to another module is considered an operator and all the parameters are considered operands of this operator.

g. Testing

The seventh input to the optimization model is the *testing* parameter. The metric *Defects per Thousand Delivered Lines of Code (KDLOC)* or *defects per function point* [2] can be applied to this node. This is a rough measure of the reliability of the software as the defect density directly impacts the reliability of the software.

h. Implementation

The optimization model accepts *implementation* parameter as the last input. The metric that can be applied to this node is *number of modules successfully implemented*. The metric is represented as follows:

$$M = \frac{No.\ of\ modules\ successfully\ implemented}{Total\ no.\ of\ modules\ to\ be\ implemented} \tag{7}$$

where *M* represents the success factor of the modules.

2.2.2. Output Layer

The output layer of the optimization mode architecture computes the *Acceptance Factor (AF)* using the eight inputs of the input layer. The AF value represents the level of acceptance of the software developed by the software project team. The different levels of acceptance are shown in T able 4.

Table 4. Different levels of acceptance.

Range of AF Values	Level of Acceptance
0.67– 1.00	Excellent (Can be Accepted)
0.33-0.66	Good (Can be Accepted)
0.01-0.32	Average (May be Accepted)
≤ 0	Poor (Cannot be Accepted)

Based on the AF values, the quality of the software as well as the ability of the software project team members can be determined. If the quality does not meet the required standard, the team members can be given proper guidance and a new project of similar category can be given. Again at the end, the quality will be tested using AF value and if succeeded, then the team can be assigned the next type of project. This process is continued until the team members get trained in all different types of project, which the organization concentrates on. The outcome of the process will be an optimized software project team, which in turn optimizes the project team members.

3. Case Study

To demonstrate the use of the model, a case study of the students' project evaluation has been considered. In this study, only organic type projects have been considered for the time being. No other metric information has been considered in the case study. The model takes eight input values from the input layer and then evaluates the AF.

A sample training set has been formed to train the model by using previous project evaluation data. The accuracy of the model can be improved by adding more data into the training set. In the study, organic type projects, in the range 10 KDLOC to 50 KDLOC are considered. After the completion of each project, the different input parameter values are fed into the input layer and the corresponding acceptance factor will be evaluated. If the AF value indicates that the project was successful, then the team will proceed to next type of project in the training set. If the project was not successful, as indicated by AF value, then the problem occurred during the development phases has to be thoroughly analyzed and a similar type project has to be undertaken again from the training set. This process is continued until the team gets optimized in different types of projects.

For the case study a sample training set has been formulated, consisting of input parameters and their corresponding desired output value, of organic type projects with a code size of 50 KDLOC maximum. For simplicity, only four inputs among the eight inputs in the input layer are considered. All other desired input information like number of user inputs, number of user outputs, number of inquires, number of files, no number of external files, number of modules to be implemented, coding method to be adopted to reduce the complexity etc., has been given to the students in advance. The four inputs considered in the case study are effort, tools, testing and implementation. For the testing input, the *defect rate* is used as the metric. If the defect rate metric has value 0, then it indicates that no defects have been identified. On the other hand, if the metric has value 1 then it indicates that all lines of code in the software are defective. For implementation input, the metric *number of modules successfully implemented* is used.

Using the training set consisting of the above-mentioned inputs and desired output value, an optimization model has been formulated here. In the actual model, all the eight input values will be considered to prepare the optimization model.

The graph representing the initial state of the model (before training the model) is given in Figure 2.

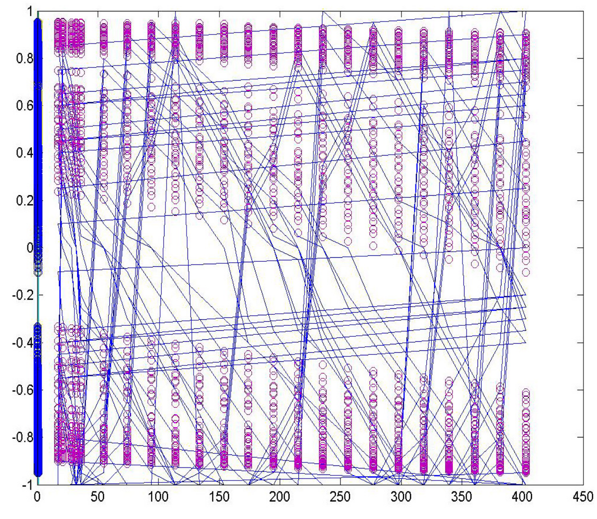

Figure 2. Initial state of optimization model.

The graph showing the performance of the training model is given in Figure 3 which shows that the model has been optimized and it can be used for assessing the acceptance of the project. Figure 4 represents the model after optimization.

The weights associated with four inputs such as effort, tools, testing and implementation, after training the model are as follows:

w_1 (Effort)	= -0.0003
w_2 (Tools)	= 0.7836
w_3 (Testing)	= -0.3213
w_4 (Implementation)	= 0.0800.

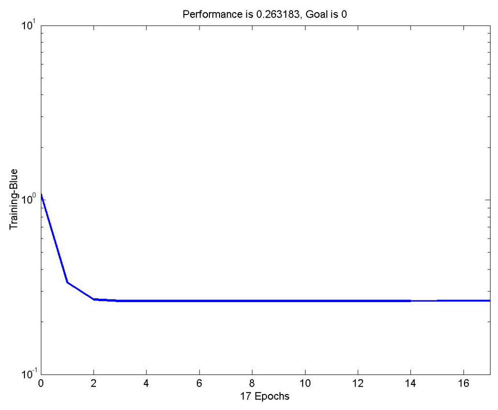

Figure 3. Performance graph of the optimization model.

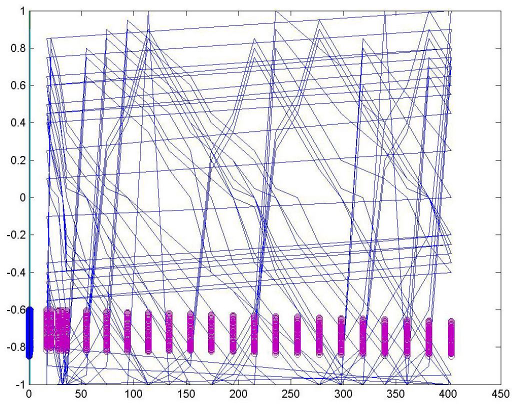

Figure 4. Graph representing the optimized model.

Using this model we can optimize the project team of students by assessing the AF value. Based on the AF value, the project leader or the concerned staff-in-charge can decide whether to give the same type of project again to improve their performance or to go to the next type of organic type project. Again the model is used to assess the AF value of the newly assigned project. Based on the AF value, the necessary action will be taken by the project leader or staff-in-charge. This process is repeated until the team gets optimized in all required categories of organic type projects.

4. Conclusion

This paper gives an insight into how neural model can be used for software project team optimization. This architecture uses basic metrics for accessing the quality of the software developed by the team. This model can be used by software organizations for effectively train their newly recruited software trainees. The advantage of this model over conventional training methods is that the team members get well versed in all types of projects, the organizations deals with.

The proposed neural model can be further improved by incorporating advanced quality metrics in each layer of the proposed neural model to enhance the optimization of team members.

References

[1] Fairley R., *Software Engineering Concepts*, McGraw Hill, 2000.

[2] Hyatt L. and Rosenberg L., "A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality," *8th Annual Software Technology Conference*, Utah, 1996.

[3] Jalote P., *An Integrated Approach to Software Engineering*, Narosa Publishing, 1998.

[4] Kaner C. and Bond W., "Software Engineering Metrics: What Do They Measure and How Do We Know?," *10th International Software Metrics Symposium*, 2004.

[5] Mehrotra K., Ranka S., and Chilukuri M., *Elements of Artificial Neural Networks*, MIT Press, 1997.

[6] Perampalam S., "Software Metrics," http://www.cs.ucl.ac.uk/staff/W.Emmerich/lectures/3C05-01-02/aswe12.pdf

[7] Roger S., *Software Engineering: A Practitioner's Approach*, McGraw Hill International Edition, 2001.

[8] VanDoren E., "Halstead Complexity Measures," http://www.sei.cmu.edu/str/descriptions/halstead_body.html.

[9] Watson A. and McCabe T., "Code Complexity Metrics,"http://www.softwaresafetynet/Metrics/.

[10] Yong H., Juhua C., Zhenbang R., Liu M., and Kang X., "A Neural Networks Approach for Software Risk Analysis," *Sixth IEEE International Conference on Data Mining-Workshops (ICDMW'06)*, pp. 722-725, 2006.

Madhu S. Nair is currently working as lecturer at Rajagiri School of Computer Science, Kochi. He received his BSc in computer applications (BCA) from Mahatma Gandhi University with first rank and his MS in computer applications (MCA) from Mahatma Gandhi University with first rank. He holds a post graduate diploma in client server computing (PGDCSC) from Amrita Institute of Technology. He had also qualified National Eligibility Test (NET) for lectureship conducted by University Grants Commission (UGC) and Graduate Aptitude Test in Engineering (GATE) conducted by Indian Institute of Technology (IIT). He had published papers in national and international journals. He is also a life member of Computer Society of India (CSI).

Jaya Vijayan is currently working as lecturer at Rajagiri School of Computer Science, Kochi. She received her BSc degree in physics from University of Kerala and master degree in computer applications (MCA) from Manonmaniam Sundaranar University. She has around eight years academic experience at the postgraduate level. Her areas of interests are software engineering and neural networks.