Available online at www.elixirpublishers.com (Elixir International Journal)

Computer Engineering

Elixir Comp. Engg. 73 (2014) 26347-26352

ISSN: 2229-712X

Improved prediction of software defects through perceptrons

V.Jayaraj* and N.Saravana Raman
Bharathidasan University, Trichy.

ARTICLE INFO

Article history:
Received: 21 June 2014;
Received in revised form:
25 July 2014;
Accepted: 8 August 2014;

Keywords
Software Defect Prediction (SDP),
Multi-Layer Perceptron Neural Network (MLPNN),
Back-propagation algorithm.

ABSTRACT

Occurrences of certain defects are inevitable despite meticulous planning, proper process control, and good documentation during software development. Such defects lead to quality degradation which can be the underlying reason for failure. Conscious efforts are required to control and minimize software engineering defects, but this costs money, time and resources. Predicting defective software module facilitates maintenance and corrective measures. In Software Defect Prediction (SDP), predictive model estimation is based on code attributes to assess software modules containing errors likelihood. This study proposes improved activation function based on cubic spline for Multi-Layer Perceptron Neural Networks (MLPNN) to classify SDP. The proposed method is evaluated using KC1 dataset.

Introduction

Software defects detected by customers ensure a major cost penalty for software companies. So, knowledge of how many defects to anticipate in a software product at any stage during development is a valuable asset. The ability to estimate defects will improve decision processes about releasing a software product considerably. Also, it improves software products production process through employing a prediction model that considers software production processes dynamic nature and reliably predicts defects [1].

A software defect is a mistake, error, bug, flaw, failure, or fault in a computer program/system that can generate an inaccurate/unexpected outcome, or preclude software from behaving as intended. A project team plans to create a quality software product with zero/little defects. High risk components in a software project are caught quickly to enhance software quality. Software defects incur cost regarding quality and time. Also, identifying/rectifying defects is time consuming and expensive. It is practically impossible to eliminate every defect but reducing defects magnitude and their adverse effect on projects is possible [2].

Software metrics describe program complexity and estimate software development time. "How to predict software quality through software metrics, before being deployed" is a key question, triggering much research to uncover an answer. There are many papers supporting statistical models and metrics which profess to answer quality questions. Usually, software metrics elucidate software product's quantitative measurements or its specifications. Defects are defined in disparate ways but are generally aberrations from specifications or ardent expectations which lead to procedure failures. Defect data analysis is of two types; Classification and prediction that can extract models to describe significant defect data classes or predict future defect trends.

Software metrics are a broad range of computer software measurement applicable to software processes with the aim of improving it continuously. Measurement is used to assist quality control, estimation, productivity assessment and project control. Measurement helps software engineers to assess technical work products quality and to assist in tactical decision making as project proceeds. There are many metrics based on source code analysis developed during the last decades for different programming paradigms like structural programming and Object-Oriented Programming (OOP). An important step in establishing a measurement program is measures selection for use [3]. Metrics selected should fit the process used and directly impact delivered software quality. Different metrics are appropriate for different products/processes within the same organization. Metrics validation is another topic in software measures as their acceptance depends on whether they can predict important qualities.

Defect prevention is performed at all Software Development Life Cycle (SDLC) stages. Major activities related to software defect prevention include: creation of a software defect prevention plan, defect inspection, defect causal analysis, classification of defects, defect prioritization and detection. Suma and Nair [4] consider defect pertains to an inaccuracy or blemish in a software product/process. The term usually refers to errors, bugs, faults or failures [5]. The terms error, fault and failure can be elaborated as 'action that leads to incorrect result', 'erroneous decision caused by wrong interpretation of available information' and 'lack of ability to meet expected performance' respectively. Defect prevention aims at prohibiting defects occurring beforehand [4]. Defect prevention can also be defined as quality improvement process through defects identification and root causes analysis and formulating effective corrective/preventive measures to prevent such defects from recurring [5,8].

Casual analysis is used for Root Cause Analysis (RCA) [9, 5] a technique for risks identification associated with distinctive defects. IBM devised Orthogonal Defects Classification (ODC), a common technique to unearth and classify defects in software products [10-12]. Bean [12] recommends Performance and Continuous Re-Commissioning Analysis Tool (PACRAT) to create, revise and archive during defect prevention. Due to software products diversified nature, there are no perfect/ideal solutions to preclude defects [14].

Tele:
E-mail addresses: jaya_v2000@yahoo.com

MLPNN consists of units arranged in layers [14]. Each layer has nodes and in a connected network, as considered here, each node connects every node in subsequent layers. Each MLP has a minimum of 3 layers including the input layer, one/more hidden layers and the output layer. The input layer distributes inputs to subsequent layers. Linear activation functions are present in input layers but no thresholds. In addition to weights, each hidden unit node and output node have thresholds associated with them. Hidden unit nodes have nonlinear activation functions and outputs linear activation functions. So, each signal feeding into a node in subsequent layers has original input multiplied by a weight with threshold added and passed through activation function that is either linear or nonlinear (hidden units).

This study proposes to investigate classification accuracy of MLPNN for a Software Defect Prediction (SDP) and proposes improved activation function based on cubic spline for MLPNN. The rest of the paper is organized as follows: section 2: literature survey, section 3: Methodology, Section 4: results and discussion and section 5: conclusion.

Related Work

Application of novel resampling strategies to SDP was suggested by Pelayo and Dick [15] which focused on Synthetic Minority Oversampling Technique (SMOTE) technique, a method of over-sampling minority-class examples. The goal was determining if SMOTE improved defect-prone modules recognition, and cost involved. Experiments proved that after SMOTE resampling, it was more balanced classification which found 23% improvement in average geometric mean classification accuracy on 4 benchmark datasets. The resulting classifiers would never predict faulty minority class. This machine learning problem was referred to as learning from unbalanced datasets. It examined stratification, a technique to learn unbalanced data that received little attention in software defect prediction

A SDP models critique was presented by Fenton and Neil [16]. Careful analysis of past/new results showed the conjecture to lack support and that some models were misleading. Holistic models were recommended for SDP, using Bayesian belief networks was discussed. It also argued for research into a software decomposition theory to test hypotheses about defect introduction and help construct better software engineering science.

A rough set model for SDP was proposed by Yang and Li [17], where defect detectors focused on SDP datasets. A rough set model to deal with the data sets of SDP attributes was presented. Applying this model to a famous public domain data set created by NASA's metrics data program showed its great performance.

A hybrid approach to coping with high dimensionality and class imbalance for SDP was proposed by Gao, et al., [18] where the author applied it to datasets group in a SDP context which used two classification learners and six feature selection techniques. It compared technique to an approach where feature selection and data sampling were used as also the case where feature selection was used singly. Experimental results showed that select Random Under-Sampling (RUS) Boost technique to be more effective compared to the other approaches in improving classification performance.

A novel evaluation method for defect prediction in software systems was proposed by Wang, et al., [19], where the author proposed evaluation metrics. It was applied to a dependency graph from the target software system, and got a list of classes ordered by their predicted defect degree under that metric. By using actual defect data from a subversion database and evaluating each metric’s quality through a weighted reciprocal ranking mechanism. This method revealed evaluated metrics overall quality and also quality of prediction result for each class, especially costly ones. Evaluation results and analysis showed the method’s efficiency and rationality.

A Neural Network (NN) based approach to model defects severity in function based software systems was proposed by Jianhong, et al., [20], where five NN based techniques were explored. Comparative analysis was performed to model faults severity present in function based software systems. NASA's public domain defect dataset was used for modelling. Comparison of various algorithms was made based on Mean Absolute Error, Root Mean Square Error and Accuracy Values. It concluded that of the five NN based techniques Resilient Back propagation algorithm based NN was best to model software components to varied fault severity levels. So, the proposed algorithm can identify modules with major faults needing immediate attention.

A constructive Radial Basis Function (RBF) NN to estimate defects probability in software modules was proposed by Bezerra, et al., [21].A new constructive RBF neural network based algorithm was introduced aimed at predicting error probability in fault-prone modules called RBF-Dynamic Decay Adjustment (DDA) with probabilistic outputs. This method’s advantage was that it informs the test team of defect probability in a module, instead of indicating whether the module was fault-prone. Experimental results showed the new method outperformed Pairwise Opposite Class-Nearest Neighbor (POC-NN) algorithms and was equal to KNN regarding performance and also produced less complex classifiers.

Efficient prediction of software fault proneness in modules using Support Vector Machines (SVM) and probabilistic neural networks was proposed by Al-Jamimi and Ahmed [22], that focused on evaluating high-performance SVMs and Probabilistic Neural Networks (PNNs) based fault predictors. Five public NASA datasets from PROMISE repository were used to make these models repeatable, refutable, and verifiable. According to results, PNNs provided best prediction performance for most datasets regarding accuracy rate.

Application of NN to predict software development faults using object-oriented design metrics was proposed by Thwin and Quah [23]. Object-oriented metrics are used to estimate quality. Quality estimation in practice means estimating reliability/maintainability. In an object-oriented metrics work context, reliability was measured as number of defects. The study was conducted on 3 industrial real-time systems having many natural faults reported for three methods.

A survey ON SDP using software metrics was done by Punitha and Chitra [24], which focused on identifying defective modules and so the scope of software needing examination for defects can be prioritized. This allowed developer to run test cases in predicted modules using test cases. The new methodology helps identify modules needing immediate attention and so software reliability can be improved quicker as higher priority defects are handled first. The aim of this investigation is to increase classification accuracy of Data mining algorithms. To start, it was initially planned to evaluate existing classification algorithms based on its weakness.

Cost-sensitive boosting NN for software defect prediction was proposed by Zheng [25], where three cost-sensitive boosting algorithms were studied to boost NN for software defect prediction. The first algorithm based on threshold-moving attempts to shift classification threshold to non-fault prone modules so that more fault prone modules are correctly classified. The other two weight updating based algorithms incorporated misclassification costs in a weight update rule to boost procedure so that algorithms boost more weights on samples associated with misclassified defect-prone modules. Performances of the three algorithms were evaluated using four NASA projects datasets regarding a singular measure, the Normalized Expected Cost of Misclassification (NECM). Experimental results revealed that threshold-moving was best to build cost sensitive software defect prediction models with boosted NN among the three algorithms studied, especially for datasets from projects developed by object-oriented language.

A robust recurrent NN modelling for software fault detection and correction prediction was proposed by Hu, et al., [26] in which the author proposed the following approach. First, recurrent NN were applied to model both processes. A systematic networks configuration approach was developed within the framework, with Genetic Algorithm (GA) according to prediction performance. To ensure robust predictions, an extra factor characterizing dispersion of prediction repetitions was incorporated into performance function. Comparisons with feed forward and NN analytical models were developed with regard to a real data set.

An effort prediction framework for software defect correction was proposed by Hassouna and Tahvildari [27] which has four enhancements: Data Enrichment, Majority Voting, Adaptive Threshold and Binary Clustering. This uses issues of common properties to form clusters which produced predictions. Numerical results showed a noticeable improvement over other methods proposed.

A study of subgroup discovery approaches for defect prediction was proposed by Rodriguez, et al., [28], where the author used a descriptive approach for defect prediction instead of precise classification techniques usually used. This allowed characterisation of defective modules with simple rules that could be easily applied by practitioners and deliver a practical result instead of highly accurate results. Results showed that generated rules can guide testing effort to improve software development projects quality. Such rules indicate metrics, threshold values and relationships between defective modules metrics.

Methodology

Identification/removal of software defects is tedious and time consuming. Improperly planned projects have defects and time to spot and fix them is more than code development time. Identification of modules necessitating re-engineering is a reverse engineering sub-area focusing on faulty modules prediction based on present information sources like documentation/source code. Predicting defective module is important in maintenance and reuse by simplifying system working with information/reusable parts localization. Predictive models estimation in software defect prediction is based on code attributes to assess software modules with a likelihood of errors. This study- investigates classification accuracy of Boosting techniques for software defect prediction based on KC1 dataset.

KC1 Dataset

KC1 dataset is a public, NASA Metrics Data Program [29]. KC1 dataset is used to verify and improve predictive software engineering models. KC1 is a C++ system implementing storage management for ground data receipt and processing. The dataset includes McCabe and Halstead features code extractors. The measures are module based.

Defect detectors are assessed as follows:

a = When Classifier predicts no defects, module has no error.

b= When Classifier predicts no defects, module actually has error.

c = When Classifier predicts some defects, module actually has no error.

d = When Classifier predicts some defects, module actually has error.

The accuracy, detection probability (pd) or recall, probability of false alarm (pf), precision (prec) and effort are calculated as

$$accuracy = \frac{(a+b)}{(a+b+c+d)} \quad (1)$$

$$recall = \frac{d}{(b+d)} \quad (2)$$

$$pf = \frac{c}{a+c} \quad (3)$$

$$prec = \frac{cd}{b+d} \quad (4)$$

$$effort = \frac{(c.LOC+d.Loc)}{TotalLOC} \quad (5)$$

The KC1 dataset includes 2109 instances, 22 different attributes which are 5 different LOC, 3 McCabe metrics, 12 Halstead metrics, a branch count and 1 goal-field. Dataset attribute information is as follows: McCabe's line count of code, design complexity, cyclomatic complexity, effort, program length, Halstead, total operands, class and others.

Examples from dataset:

Example 1 - 1.1, 1.4, 1.4, 1.4, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 2, 2, 2, 2, 1.2, 1.2, 1.2, 1.2, 1.4, false

Example 2 - 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, true

Example 3 - 83, 11, 1, 11, 171, 927.89, 0.04, 23.04, 40.27, 21378.61, 0.31, 1187.7,65, 10,6, 0,18, 25, 107, 64, 21, true.

Multilayer Perceptron Neural Network (MLPNN)

The MLPNN Model has three layers. The first is input layer, the middle layer is a hidden layer and last layer is an output layer. All layers consist of "n" neurons. All input layer neurons are connected to middle layer neurons. The hidden layer neurons outputs are connected to output layer neurons. The weights by which they are connected are adjustable and adapt according to inputs fed and trained [2]. Input layer consists of vector(x1...xp) which is fed as input. Each variable's value is standardized between '-1' to '1'. A constant input called Bias of value 1.0 is fed to each hidden layer; bias is multiplied by a weight and added to sum going into neuron. At hidden layer, value from each input neuron is multiplied by a weight (w_{ji}), and resulting weighted values are added producing a combined value u_j. The weighted sum (u_j), is fed into a transfer function, σ, which outputs a value hj. Outputs from hidden layer are fed to neurons of output layer. At output layer, each value from hidden layer is multiplied by weight (w_{kj}), and resulting weighted values are added producing a combined value vj. Weighted sum (v_j) is fed into a transfer function, σ, which outputs value y_k. The output of model is "y". The weights between input and hidden units determine when each hidden unit is active, and by modifying weights, a hidden unit chooses what it represents.

The purpose of training NN is to find set of weights that approximates output of NN very close to target values. So it is very important to decide on the hidden layers required, on number of neurons to be used in each layer etc. In most NN, only one hidden layer is used. Two/more hidden layers are used to model data with discontinuities. Using two/more hidden layers have no effect on results; instead they increased risk of converging to local minima. The next issue is deciding how many neurons are needed in network's hidden layer. When inadequate neurons are used, it may not fully train network and also not yield good results. If too many neurons are used, time needed to train may be too long and model may begin producing noise [30].

Back propagation is a widely used algorithm for supervised learning with multi-layered feed-forward networks. The idea of back propagation learning algorithm [3l] is repeated application of chain rule to compute each weight's influence in network regarding an arbitrary error function E as shown in figure 1.

The back-propagation algorithm consists of the following steps:

Each Input is multiplied by a weight that would inhibit/excite input. The weighted sum of inputs is calculated first, it computes total weighted input Xj, using formula:

$$X_j = \sum_i y_i W_{ij} \tag{6}$$

where yi is activity level of jth unit in previous layer and Wij is weight of connection between ith and jth unit.

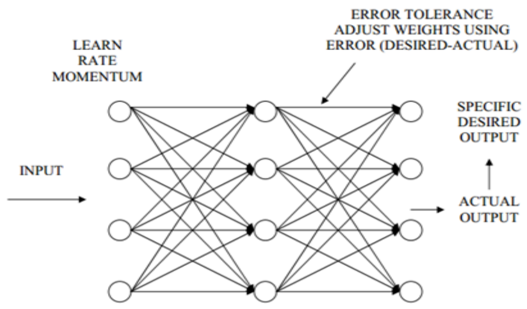

Figure 1: Back Propagation Network

Then weighed Xj is passed through a sigmoid function that scales output in between a 0 and 1. Next, unit calculates activity yj using some function of total weighted input. Sigmoid function is used:

$$y_i = \frac{1}{1+e^{-x_j}} \tag{7}$$

Once output is calculated, it is compared to required output and total Error E computed.

Once activities of output units are determined, the network computes error E, defined by expression:

$$E = \frac{1}{2}\sum_j (y_j - d_j)^2 \tag{8}$$

where yj is activity level of ith unit in top layer and dj is desired output of ith unit. Now error is propagated backwards.

1. Compute how fast error changes as activity of output unit changes. This Error Derivative (EA) is difference between actual and desired activity.

$$EA_j = \frac{\partial E}{\partial y_i} = y_j - d_i \tag{9}$$

2. Compute how fast error changes as total input received by output unit changes. This (EI) is the answer from step 1 multiplied by rate at which output of a unit changes as its total input changes.

$$EI_j = \frac{\partial E}{\partial x_j} = \frac{\partial E}{\partial y_j} \times \frac{dy_j}{dx_j} = EA_j Y_j (1 - y_j) \tag{10}$$

3. Compute how fast error changes as weight on connection into an output unit changes. This (EW) is the answer from step 2 multiplied by activity level of unit from which connection emanates.

$$EW_{ij} = \frac{\partial E}{dW_{ij}} = \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{dw_{ij}} = EI_j Y_i \tag{11}$$

4. Compute how fast error changes as a unit's activity in previous layer changes. This step allows application of back propagation to multi-layer networks. When a unit's activity in a previous layer changes, it affects activities of all output units connected to it. So to compute overall effect on error, all separate effects on output units are added. Each effect is simple to calculate. It is the answer in step 2 multiplied by weight on connection to output unit.

$$EA = \frac{\partial e}{\partial y_1} = \sum_j \frac{\partial E}{\partial x_j} \times \frac{\partial x_j}{\partial y_1} = \sum_j EI_j W_{ij} \tag{12}$$

By using steps 2 and 4, convert the EA's of one layer of units into EA's for previous layer. This procedure is repeated to get EA's for as many previous layers as needed. Once a unit's EA is known, steps 2 and 3 are used to compute EW's on incoming connections [32].

It is proposed to implement an improved activation function based on cubic spline.

The proposed spline activation function reproduces shape of whole cubic spline along directions specified by weight wj, j=1,..,n and written as:

$$\varphi(w_j x) \sum_{i=1}^{N} c_i \left| w_j x - \alpha_{ij} \right|^3 \tag{13}$$

The new activation function can be written as:

$$f(x) = \sum_{j=1}^{n} \mu_j \varphi_j (w_j x) \tag{14}$$

μ_j and w_j are found using back propagation, thus optimal set of parameters and coordinates are located.

The tracts in spline are described by a coefficients combination. Local spline basis functions controlled by only 4 coefficients represent activation function. Catmull-Rom cubic spline is used and its ith tract expressed as:

$$F_i(u) = \begin{bmatrix} F_{x,i}(u) \\ F_{y,i}(u) \end{bmatrix} = \frac{1}{2}\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \tag{15}$$

Results and Discussion

The KC1 Dataset is used for the performance evaluation of the proposed technique, 2107 samples was used of which 1391 samples are used as training set and 716 samples are used for testing. The software complexity measures such as LOC measure, Cyclomatic complexity, Base Halstead measures and Derived Halstead measures are used to classify the software modules. The proposed spline activation function reproduces the shape of whole cubic spline along the directions specified by weight wj, j=1,..,n. The Neural Network is made up of 20 input neurons and two hidden layers. The results obtained for classification accuracy, precision and recall are shown from figure 2-4.

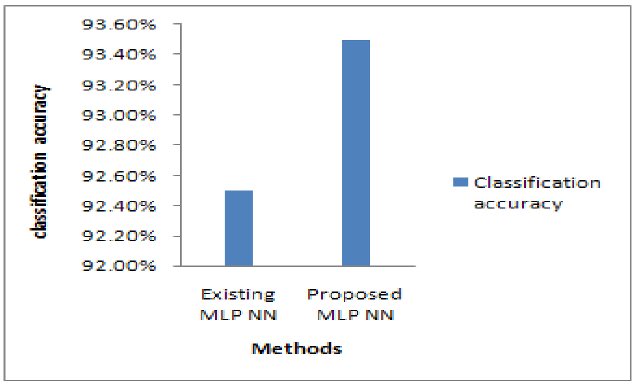

Figure 2. Classification accuracy

Figure 2 show that the proposed MLPNN achieves an increased classification accuracy of 1.08% than the existing MLPNN.

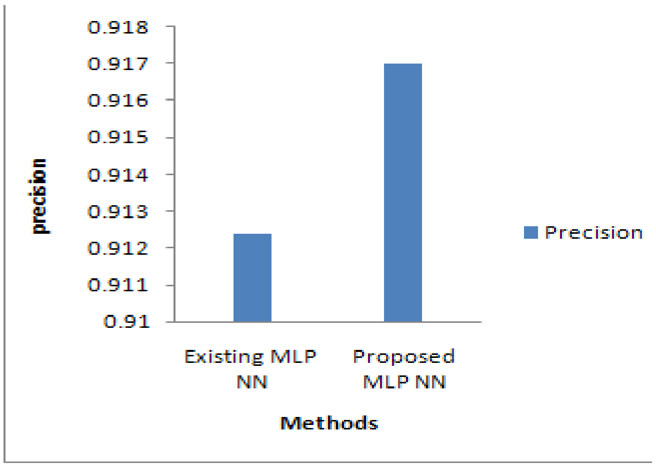

Figure 3. Precision

Figure 3 shows that the proposed MLPNN achieves increased precision of 0.5%than the existing MLPNN.

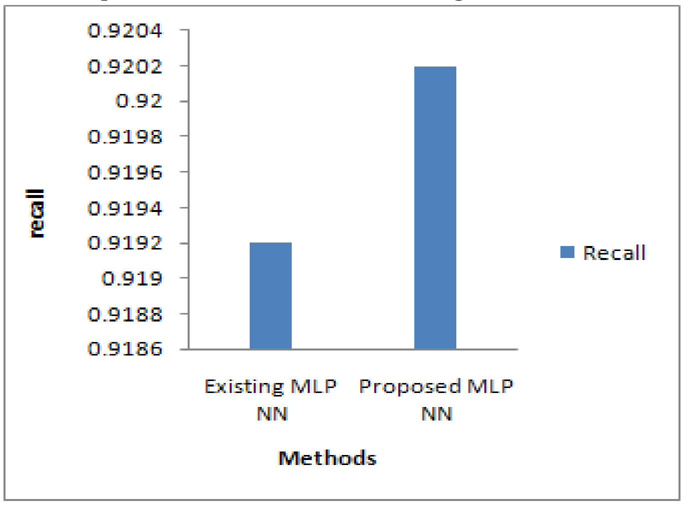

Figure 4. Rrecall

From figure 4 it is shown that the proposed MLPNN achieves a high recall percentage of 00.9202.

Conclusion

Software Defect Prediction (SDP) helps developers identify defects based on present software metrics through data mining techniques. It is a major requirement to enhance software quality. It helps reduce software development cost in development/maintenance phases. In this paper proposes to investigate classification accuracy of MLPNN for a SDP and proposes improved activation function based on cubic spline for MLPNN. The proposed MLPNN shows 1.08% increased classification accuracy, 0.5% precision and 0.11% recall than the existing MLPNN. Further investigation can be carried out in the direction of probability recurrences of neural network.

References

[1] Bergander, T., Luo, Y., & Ben Hamza, A. (2007, August). Software defects prediction using operating characteristic curves. In Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on (pp. 713-718). IEEE.

[2] Rawat, M. S., &Dubey, S. K. (2012). Software Defect Prediction Models for Quality Improvement: A Literature Study. International Journal of Computer Science Issues(IJCSI), 9(5).

[3] Scotto, M., Sillitti, A., Succi, G., &Vernazza, T. (2004). Dealing with Software Metrics Collection and Analysis: a Relational Approach. Stud. Inform. Univ.,3(3), 343-366.

[4] V. Suma and T. R. G. K. Nair, “Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels,” World Academy of Science, Engineering and Technology (WASET), 42, pp. 258_262, 2008.

[5] B. Clark and D. Zubrow, How Good Is the Software: A review of Defect Prediction Techniques, sponsored by the U.S. department of Defense by Carnegie Mellon University, version 1.0, page 5, 2001.

[6] C. P. Chang and C. P. Chu., Defect prevention in software processes: An action based approach, The Journal of Systems and Software 80, 559–570, 2007.

[7] S. Kumaresh and R. Baskaran, Defect Analysis and Prevention for Software Process Quality Improvement, International Journal of Computer Applications, Volume 8– No.7, pp. 42_47, 2010.

[8] M. Kalinoski, G. H. Travassos and D. N. Card., “Towards a Defect Prevention Based Process Improvement Approach,” 34th Euromicro Conference Software Engineering and Advanced Applications (SEAA), IEEE Computer Society, pp. 199_206, 2010.

[9] P. Trivedi. & S. Pachori, Modeling and Analysis of Software Defect Prevention Using ODC, International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 1, No. 3, pp. 75_77, 2010.

[10] A. Shenvi, “Defect Prevention with Orthogonal Defect Classification,” International Software Engineering Conference (ISEC’09), ACM, pp. 83_87, 2009.

[11] P. Tiejun, Z. Leina and F. C. bin., “Defect Tracing System Based on Orthogonal Defect Classification,” International Conference on Computer Science and Software Engineering (CSSE), IEEE, pp. 574_577, 2008.

[12] E. Bean, Defect Prevention and Detection in Software for Automated Test Equipment,” Instrumentation & Measurement Magazine, IEEE, Volume: 11 Issue: 4, pp. 16_23, 2008.

[13] Faizan, M., Khan, M. N. A., &Ulhaq, S. (2012). Contemporary Trends in Defect Prevention: A Survey Report. International Journal of Modern Education and Computer Science (IJMECS), 4(3), 14.

[14] Nazzal, J. M., El-Emary, I. M., &Najim, S. A. (1818). Multilayer perceptron neural network (MLPs) For analyzing the properties of Jordan oil shale. World Applied Sciences Journal. ISSN, 4952, 546-552.

[15] Pelayo, L., & Dick, S. (2007, June). Applying novel resampling strategies to software defect prediction. In Fuzzy Information Processing Society, 2007. NAFIPS'07. Annual Meeting of the North American (pp. 69-72). IEEE.

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

[17] Yang, W., & Li, L. (2008, October). A Rough Set Model for Software Defect Prediction. In Intelligent Computation Technology and Automation (ICICTA), 2008 International Conference on (Vol. 1, pp. 747-751). IEEE.

[18] Gao, K., Khoshgoftaar, T. M., & Napolitano, A. (2012, December). A Hybrid Approach to Coping with High Dimensionality and Class Imbalance for Software Defect Prediction. In Machine Learning and Applications (ICMLA), 2012 11th International Conference on (Vol. 2, pp. 281-288). IEEE.

[19] Wang, W., Ding, X., Li, C., & Wang, H. (2010, December). A Novel Evaluation Method for Defect Prediction in Software Systems. In Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on (pp. 1-5). IEEE.

[20] Jianhong, Z., Sandhu, P. S., & Rani, S. (2010, August). A Neural network based approach for modeling of severity of defects in function based software systems. In Electronics and Information Engineering (ICEIE), 2010 International Conference On (Vol. 2, pp. V2-568). IEEE.
[21] Bezerra, M. E., Oliveira, A. L., &Meira, S. R. (2007, August). A constructive rbf neural network for estimating the probability of defects in software modules. In Neural Networks, 2007. IJCNN 2007. International Joint Conference on (pp. 2869-2874). IEEE.
[22] Al-Jamimi, H. A., & Ahmed, M. (2013, June). Machine learning-based software quality prediction models: state of the art. In Information Science and Applications (ICISA), 2013 International Conference on (pp. 1-4). IEEE.
[23] Thwin, M. M. T., &Quah, T. S. (2005). Application of neural networks for software quality prediction using object-oriented metrics. Journal of systems and software, 76(2), 147-156.
[24] Punitha, K., &Chitra, S. (2013, February). Software defect prediction using software metrics-A survey. In Information Communication and Embedded Systems (ICICES), 2013 International Conference on (pp. 555-558). IEEE.
[25] Zheng, J. (2010). Cost-sensitive boosting neural networks for software efect prediction. Expert Systems with Applications, 37(6), 4537-4543.
[26] Hu, Q. P., Xie, M., Ng, S. H., &Levitin, G. (2007). Robust recurrent neural network modeling for software fault detection and correction prediction. Reliability Engineering & System Safety, 92(3), 332-340.
[27] Hassouna, A., &Tahvildari, L. (2010). An effort prediction framework for software defect correction. Information and Software Technology, 52(2), 197-209.
[28] Rodriguez, D., Ruiz, R., Riquelme, J. C., & Harrison, R. (2013). A Study of Subgroup Discovery Approaches for Defect Prediction. Information and Software Technology.
[29] Jayaraj, V., & Raman, N. S. (2013). Software Defect Prediction using Boosting Techniques. International Journal of Computer Applications, 65(13).
[30] Tamboli, Z. J., &Nikam, P. B. (2013). Study of Multilayer Perceptron Neural Network for Antenna Characteristics Analysis. International Journal, 3(8).
[31] Riedmiller, M., & Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In Neural Networks, 1993., IEEE International Conference on (pp. 586-591). IEEE.
[32] Gupta, C. (2006). Implementation of Back Propagation Algorithm (of neural networks) in VHDL. Master of Engineering Thesis, Thapar Institute of Engineering & Technology (Deemed University), Patiala, India.