Journal of Software Engineering and Applications, 2015, 8, 384-394
Published Online August 2015 in SciRes. http://www.scirp.org/journal/jsea
http://dx.doi.org/10.4236/jsea.2015.88038

A Framework for Software Defect Prediction Using Neural Networks

Vipul Vashisht[1], Manohar Lal[1], G. S. Sureshchandar[2]

[1]SOCIS, IGNOU, New Delhi, India
[2]ASQ India Pvt Ltd., Chennai, India
Email: vipulvashisht@gmail.com, prof.manohar.lal@gmail.com, suresh.gettala@gmail.com,

Received 10 July 2015; accepted 21 August 2015; published 24 August 2015

Copyright © 2015 by authors and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY).
http://creativecommons.org/licenses/by/4.0/
Open Access

Abstract

Despite the fact that a number of approaches have been proposed for effective and accurate prediction of software defects, yet most of these have not found widespread applicability. Our objective in this communication is to provide a framework which is expected to be more effective and acceptable for predicting the defects in multiple phases across software development lifecycle. The proposed framework is based on the use of neural networks for predicting defects in software development life cycle. Further, in order to facilitate the easy use of the framework by project managers, a software graphical user interface has been developed that allows input data (including effort and defect) to be fed easily for predicting defects. The proposed framework provides a probabilistic defect prediction approach where instead of a definite number, a defect range (minimum, maximum, and mean) is predicted. The claim of efficacy and superiority of proposed framework is established through results of a comparative study, involving the proposed framework and some well-known models for software defect prediction.

Keywords

Software Defect, Software Defect Prediction Model, Neural Network, Quality Management

1. Introduction

Software quality concerns require our gradually increasing attention, especially in view of our ever increasing dependency on software to conduct routine business of life. In particular, business goals of most organizations, being targeted towards customer satisfaction and profitable growth, are being met through increasing use of software. A minor defect, or even inefficiency, in the software may lead to not only loss of millions of dollars, but loss of customer base. In view of the potentially harmful consequences of defect leakage for the reputation

How to cite this paper: Vashisht, V., Lal, M. and Sureshchandar, G.S. (2015) A Framework for Software Defect Prediction Using Neural Networks. *Journal of Software Engineering and Applications*, **8**, 384-394.
http://dx.doi.org/10.4236/jsea.2015.88038

of the product and to the supplier, its reduction in the production environment is extremely vital to achieve such goals of business.

As per IEEE definition [1], fault is an incorrect step, process, or data definition in a computer program. Throughout this paper, the terms defect and fault are used interchangeably. These terms refer to the manifestation of an error in source code, where an error is an erroneous action made by a developer [2]. Faults can be the cause of failures which occur when users experience undesirable system behavior at any point in time.

In the software development life cycle (SDLC), early prediction of defects has always been, though highly desirable yet, a challenging task for the project managers. Developing fault-free reliable software is a daunting task in the current context, when software is being developed for problems with increasing difficulty with more and more complex problem domains that involve constantly increasing constraints like requirement ambiguity and complex development processes. In spite of meticulous planning and well documented processes, occurrences of certain defects are inevitable. These software defects may lead to degradation of the product quality which may lead to failures leading to customer dissatisfaction. In today's cutting edge competition, it is necessary to make conscious efforts to control and minimize defects in software engineering processes. However, these efforts require organizations not only to spend large amount of money, time and resources but also defect prediction software based on appropriate model [3]. These efforts could help the project manager to take preventive actions, thereby saving time and cost apart from delivering high quality software. Also, the development and use of a Process Performance Model (PPM) that includes defect prediction model has been identified as one of the high maturity practices in SEI CMMI model. This falls under the process areas of Organizational Process Performance (OPP) and Quantitative Project Management (QPM). Organizations attempting for CMMI L5 appraisal have to mandatorily showcase the process improvements by making use of such prediction models [4].

In respect of software, apart from the quality consideration, the time of delivery is also an important factor. Implementing Quality reviews are time-consuming and if unplanned, they could delay the delivery of a software system. Automated tools are used to accelerate the review process. The assessments, through reviews, check whether the software has reached the required quality threshold or not and also highlight areas which still need attention [5]. This is the task of software defect predictors that are used as tools for the purpose of 1) identifying parts of a software system requiring further examination before release and 2) finding relative priorities among these parts.

In most projects, information collected during testing and defect detection phases is analyzed to help predict defects for similar types of projects. However, since it is found that so far, each of the models of defect prediction has areas where it does not function satisfactorily; the search for one model that can predict defects satisfactorily in a wide range of projects is still on. This justifies the investigations reported through this communication.

The paper is organized as follows. Section II provides a brief overview of Neural Networks, Section III reviews the existing literature on the subject, Section IV describes the proposed framework, Section V discusses the results as obtained through use of the proposed defect prediction framework vis-à-vis some other relevant models/frameworks and finally Section VI concludes the presented work.

Our investigations are based on statistical pattern classification of defects. Recent advances in learning algorithms using artificial neural networks and parallel computation have led to renewed research in the area of statistical pattern classification. Artificial neural networks, simply called ANNs or neural networks, have been applied both to time varying patterns as well as static patterns. In this respect, in the next section we briefly discuss some basic concepts, the structure and functions of a Neural Network.

2. Neural Networks

Artificial Neural Network (ANN) is a sort of an approach to computation or is a model of processor/computer based on human brain. Prevalent view of human brain is that it is a sort of neural network: A networks of about 100 billion neurons, each neuron being connected, on the average, to about 1000 other neurons. A neuron is the basic constituent of brain, a sort of elementary processor having small local memory and capable of localized information processing. To a connection between a pair of neurons, called interconnection, is associated an adjustable weight indicating the strength of the connection. The terminology of neuron and interconnection etc is used in the similar sense in ANN. In **Figure 1**, a circle denotes a neuron, and a straight line denotes interconnection. Generally, group of neurons are arranged in layers—as shown in **Figure 1** in the form of vertically arranged set

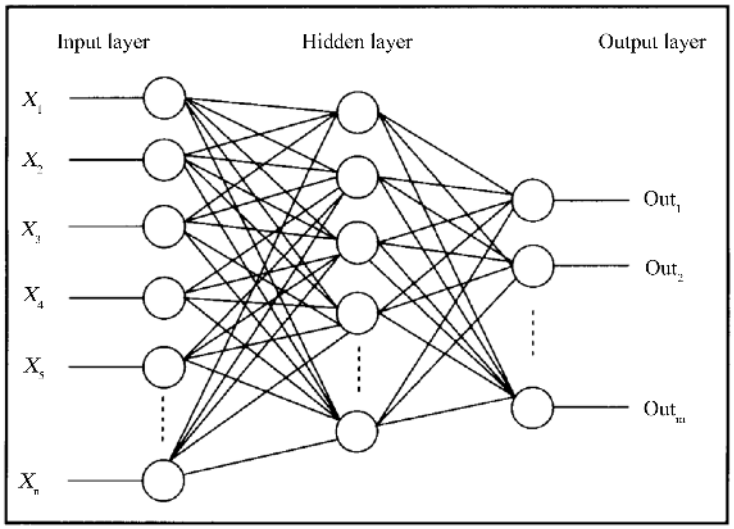


Figure 1. Neural network.

of neurons.

There is one layer for the input variables, and one for the output. There can be one or several layers between these two which are referred to as hidden layers. The signal or input given to one neuron is passed to all the neurons to which it is connected in fractions equivalent to the weight between these neurons. Each neuron calculates its output based on a function which can be sigmoid, step or some such suitable function. The strength of neural networks is their capability to learn from patterns. A neural network learns patterns by adjusting its weights. When the neural network is properly trained, it can give correct, or nearly correct, answers for not only the sample patterns, but also for new similar patterns [6] [7].

During the previous decades, neural network approach has emerged as a promising technology in applications which require generalization, abstraction adaptation and learning. The application of neural networks is to diverse fields range from autonomous vehicle control [8], financial risk analysis to handwriting recognition [9]. Therefore, it is only natural that the neural network technology can be exploited to solve different software engineering problems. For example, neural networks have been earlier applied in dynamic software reliability modeling [10].

The decision about using Neural network based framework for modeling defect prediction, was taken after diligent literature review of success of neural networks regarding software metrics models [11]-[15], and also, in view of the fact that neural networks have been found effective in situations where data relationships may not be known, as normally happens in the case of defect prediction. Also, there is sufficient evidence in support of applying neural networks in software effort estimation [16] [17]. Next, we give an over view of the relevant literature.

3. Review of Literature

In view of the fact that each model of defect prediction has its own set of advantages and disadvantages, it is hard to decide which model should be used for a particular type of project scenario, specially, as every project tends to be unique.

For the reasons mentioned earlier, software defect prediction is a very active research area in software engineering. Researchers have proposed new defect prediction algorithms and/or new metrics to effectively predict defects. The historical data of software systems is a valuable asset used for research ranging from software design to software development, software maintenance, software testing, etc.

Recently, there has been an increase in the use of computational intelligence in the field of software engineering. Computational Intelligence (CI) includes technologies of fuzzy logic, neural networks, genetic algorithms, genetic programming, and rough sets. Only a very small fraction of the activities involved in software design and development can be automated using software tools; most of the activities necessarily require human involvement. However, the human involvement, especially in view of the human judgment being imprecise and uncertain,

characterizes many of the challenges including those observed in software defect prediction. Incorporation of computational intelligence into the various phases of software development and analysis helps in addressing the problems arising due to imprecise measurement and uncertainty of information [18] [19].

General principles/approaches/steps which have been found useful so far in handling the difficult task of software defect prediction, along with the relevant literature, are summarized below:

A defect prediction model based on an enhanced Multilayer Perceptron Neural Network technique using data mining is proposed and explored in [20], in which comparative analysis of modeling of defect proneness predictions using dataset of different metrics from NASA MDP (Metrics Data Program) was performed. The proposed MLP neural network model gave better results, when compared with the existing techniques like Random Tree, classification and regression trees (CART) algorithm, and Bayesian logistic regression.

Levenberg-Marquardt (LM) algorithm based on neural network tool for the prediction of software defects at an early stage of the SDLC is described in [21]. This study uses data gathered from the PROMISE repository of empirical software engineering dataset. The dataset uses the CKOO (Chidamber and Kemerer Object-Oriented) metrics. The study concludes that for predicting the software defects, the Levenberg-Marquardt (LM) neural network based algorithm provides better accuracy (88.09%) as compared to each of polynomial function-based neural networks (pF-NNs), linear function-based neural network (lf-NN) and quadratic function-based neural network (qf-NN) respectively.

In [22], it is suggested that an under development module with same or similar metrics properties of a defective module developed in the same environment, would have the same level of defect proneness. For the purpose of defect-prediction in software programs, the authors have designed Adaptive Resonance Neural Network having 29 input nodes and two output nodes. The network is trained with data extracted from PROMISE dataset. The network improves the recall (true positive) rate in predicting whether a module is defective or not [23].

In [24] an approach is described for static reliability modeling, and for modeling of software reliability from software complexity in terms of the predictive quality and the quality of fit. The performance of a model based on the approach is compared with more traditional regression modeling techniques. For the purpose, the data have been taken from an Ada development environment for the command and control of a military data link communication system (CCCS), in which neural and regression analysis techniques were employed. It was found that the neural network model is superior to traditional regression based techniques and also had a smaller standard error.

This paper describes a framework, for applying neural networks, for formulating models for defect prediction early in the software life cycle. A series of empirical experiments are conducted based on input and output measures extracted from "real world" project subsystems. The experiments establish the efficacy and superiority of the approach. Next section describes the proposed framework.

4. The Proposed Framework

As mentioned earlier, the objective in our study is to develop prediction framework based on Neural Network for forecasting the defects. In this section, first we explain the assumptions made about the proposed framework. Then we describe the structure of the proposed framework and functions of major components of the framework. The results and other related issues are discussed in the next section.

For this purpose we use the actual runtime historical data set that is fed into the system during training of the proposed network as shown in **Figure 2**. The historical data of software systems is a valuable asset used for research in all phases of SDLC ranging from requirement gathering, analysis, software design, coding, to system testing and maintenance. The experiments reported here involve data set taken from 45 real projects from a software organization. The actual defect data is taken from completed projects based on Java technology and waterfall life cycle model.

This historical data has served as a training data to build the proposed framework and then the neural network so obtained is used to predict the defects for all new projects. As waterfall model is the oldest among the models used for software development and which is still widely practiced, it tends to be the standard against which other development approaches are compared.

We have considered along with the production effort, the prevention effort, review effort and rework effort as

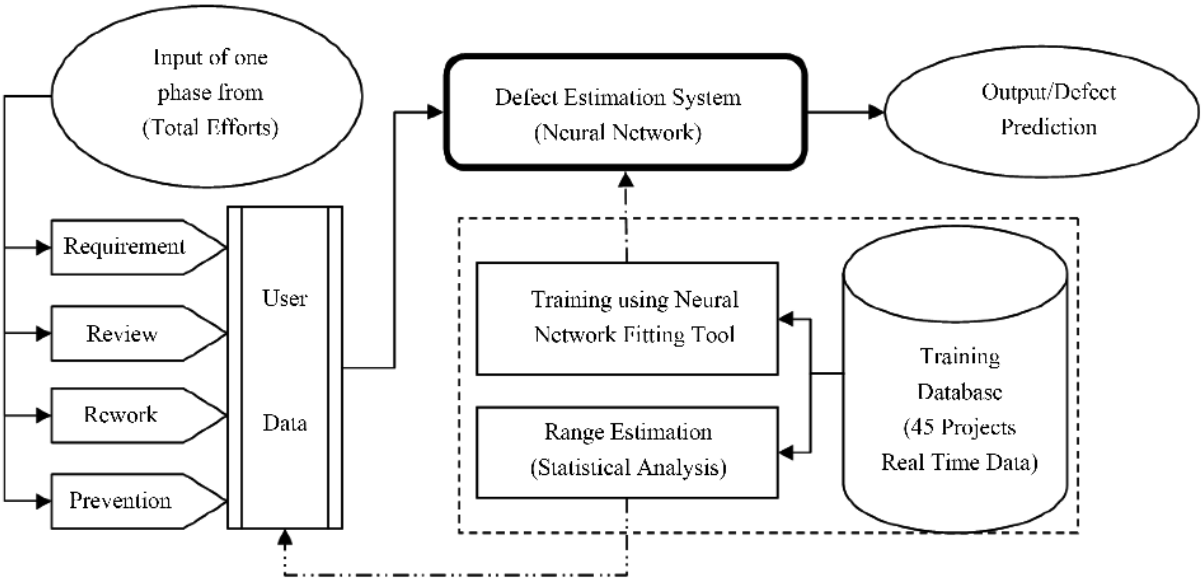


Figure 2. Model framework design.

components of input in our neural network based defect prediction framework in view of support for the fact that as voluntary costs of defect prevention are increased, the involuntary costs of rework decrease by much more than the increase in prevention costs [22] [25], thus leading to overall lower total costs, and better quality.

The proposed framework as shown in **Figure 3** requires user to enter the planned effort data of five SDLC phases, namely, Requirement Gathering, Analysis, Design, Coding and System integration testing (SIT). The data is provided with breakup of planned effort allocated to production effort, review effort, rework effort and prevention effort. If the data provided by the user matches with eligibility criteria of given range then defects are estimated using Defect Estimation System by Neural Network Technique.

4.1. Structure of the Proposed Neural Network

A feed-forward network with sigmoid (hidden and linear output neurons) is used to formulate the system. The network is trained with scaled conjugate gradient back-propagation (training).

Defect prediction system consists of five parallel neural networks with different configurations and parameters for each sub phase. Only first phase that is for Requirement Gathering is having 5 hidden layers & remaining phases have 10 hidden layers in their architecture. The architectural view of Neural Network for requirement gathering phase is shown in **Figure 4**. The selection of number of hidden layers is done in order to optimize the regression value for attaining the best performance. Although more neurons require more computation, and also have a tendency to over fit the data when the number is set too high, but at the same time, they allow the network to solve more complicated problems [26]. Input data of 45 real time projects is divided randomly in three parts before training with it is initiated: Training (70%), Validation (15%) and Testing (15%).

Levenberg Marquardt back-propagation optimization method is used for training the network. It is the fastest among methods available for the current scenario of Supervised Learning. It also requires less memory. Like the quasi-Newton methods, the Levenberg-Marquardt algorithm was designed to approach second-order training speed without having to compute the Hessian matrix. When the performance function has the form of a sum of squares (as is typical in training feedforward networks), then the Hessian matrix can be approximated as:

$$H = J^T J$$

and the gradient can be computed as: $g = J^T e$,

where J is the Jacobian matrix that contains first derivatives of the network errors with respect to the weights and biases, and e is a vector of network errors. Regression R Values measure the correlation between outputs and targets. **Figure 5** represents the relationship for all five SDLC phases.

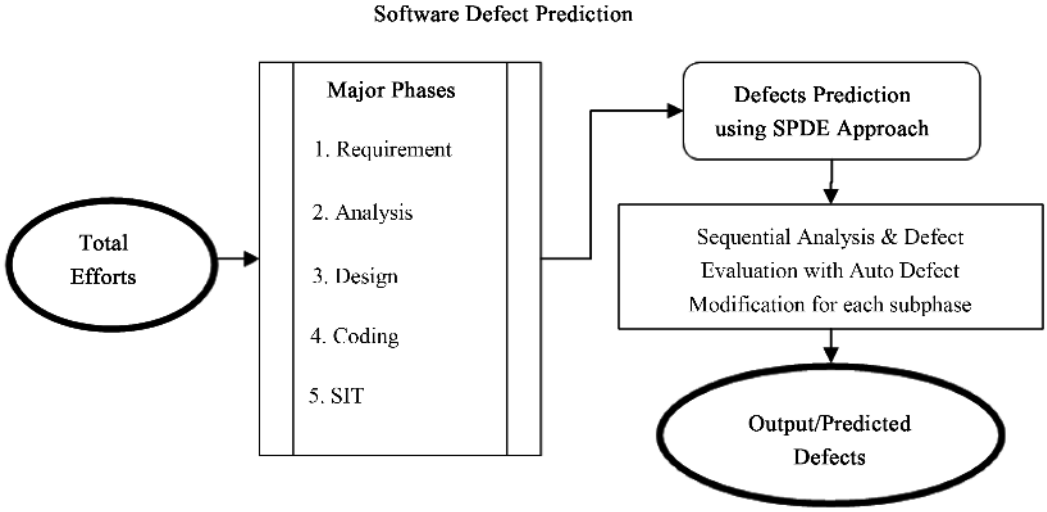


Figure 3. Software prediction model.

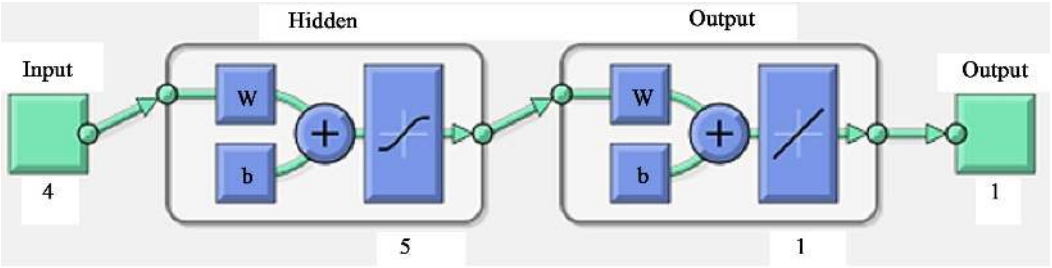


Figure 4. Architectural view of network for requirement gathering phase.

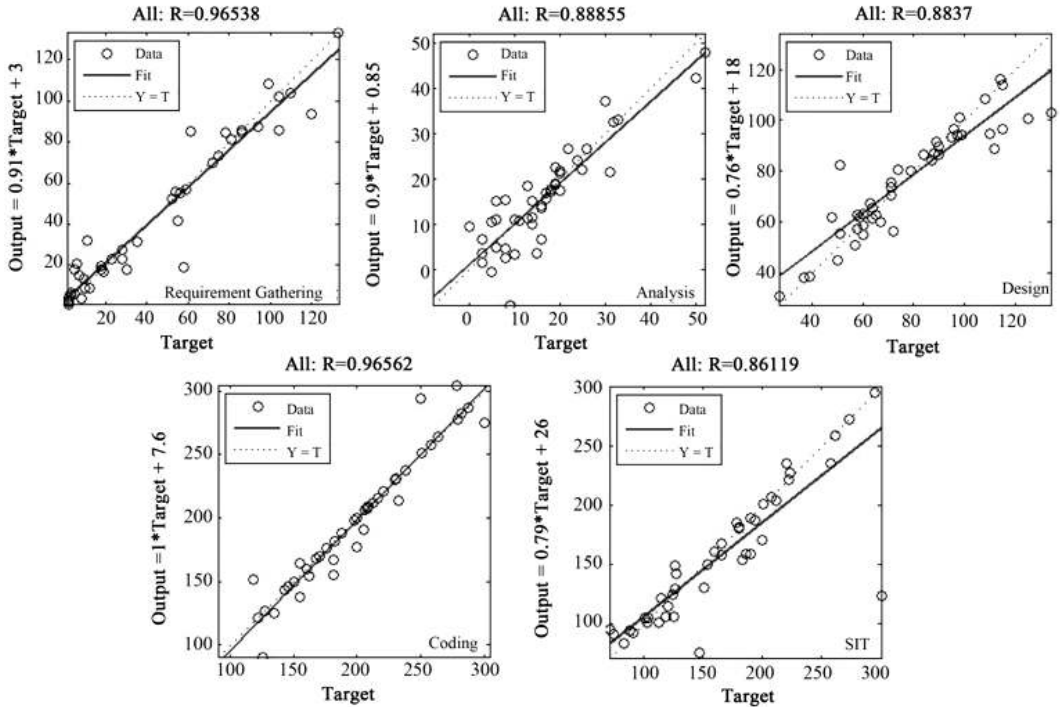
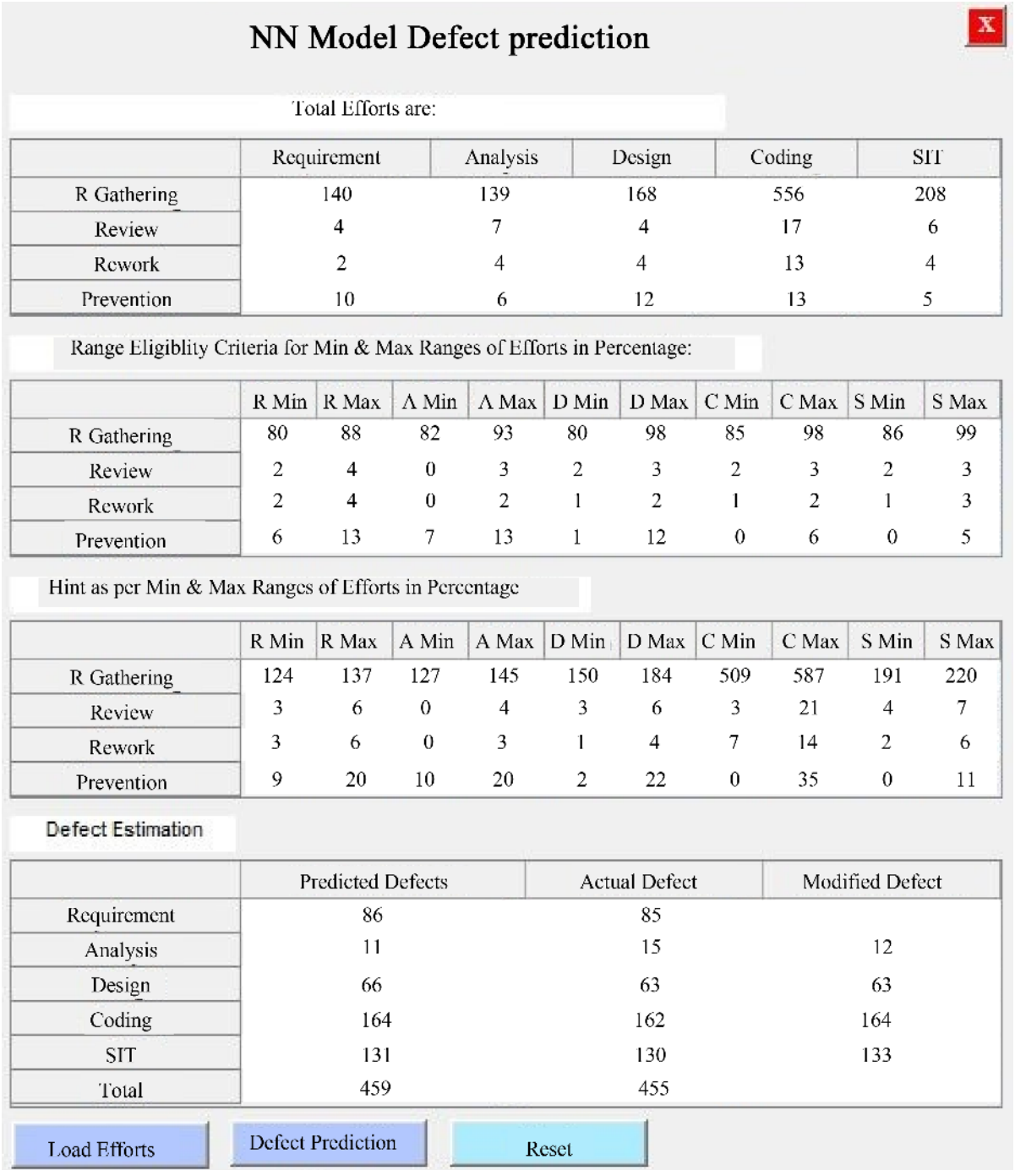


Figure 5. Representation of R value for SDLC phases.

4.2. User Interface for Testing the Framework with New Projects

One of the primary goals of our project is to present the prediction information to project managers in the most user friendly manner. Project Managers should not be required to understand the mathematics behind the prediction model. To make the predictions easily accessible to its users, a GUI based tool has been designed and implemented that asks for only the most basic information from users, and returns a straightforward output of the defect predictions.

A user interface tool (refer to **Figure 6**) has been developed using Matlab to help in testing the framework design.



	Requirement	Analysis	Design	Coding	SIT
R Gathering	140	139	168	556	208
Review	4	7	4	17	6
Rework	2	4	4	13	4
Prevention	10	6	12	13	5

	R Min	R Max	A Min	A Max	D Min	D Max	C Min	C Max	S Min	S Max
R Gathering	80	88	82	93	80	98	85	98	86	99
Review	2	4	0	3	2	3	2	3	2	3
Rework	2	4	0	2	1	2	1	2	1	3
Prevention	6	13	7	13	1	12	0	6	0	5

	R Min	R Max	A Min	A Max	D Min	D Max	C Min	C Max	S Min	S Max
R Gathering	124	137	127	145	150	184	509	587	191	220
Review	3	6	0	4	3	6	3	21	4	7
Rework	3	6	0	3	1	4	7	14	2	6
Prevention	9	20	10	20	2	22	0	35	0	11

	Predicted Defects	Actual Defect	Modified Defect
Requirement	86	85	
Analysis	11	15	12
Design	66	63	63
Coding	164	162	164
SIT	131	130	133
Total	459	455	

Figure 6. Defect prediction system UI.

The predictions thus achieved can be used to prioritize testing efforts, to plan code or design reviews, to allocate human and infrastructure resources, and to plan for risk mitigation strategy.

4.3. Working of Defect Prediction User Interface

- For any new project, the software project manager will provide the inputs required to the UI.
- The inputs would be the phase wise efforts planned for the project
- Apart from production effort, the planned review effort, planned prevention effort and the planned rework effort are also required as a feed to the framework.
- Framework provides graphical analysis for each sub phase to analyze the input eligibility (refer to **Figure 7**). Based on historical data the framework would also provide a warning message to the project manager if planned effort is less for review, rework or engineering activity for a particular phase. There could be specific scenarios where project manager might want to go ahead and ignore the warning. Example of such cases could be planning for a higher prevention effort for projects based on lesson learned from past projects or cases which might need a higher review effort since requirements from previous vendor could be incomplete or unavailable.
- Based on these inputs, the framework will forecast the number of defects that the project manager could expect to be discovered in various SDLC phases in the project.
- The defects are forecast in a range based manner. The framework would provide the minimum, maximum and the mean number of defects.
- The forecast would enable the project manager to plan prevention activities for the phase where the framework is projecting higher number of defects.
- The project manager can plan multiple preventive actions, like multiple review gates, usage of tools, increasing review effort etc to mitigate the higher probability of defect leakage.
- The framework is designed to autocorrect itself. After phase completion, user feeds the actual count of defect data. If actual defects are lower than predicted then defects leaked to the subsequent sub phase are auto corrected.
- For example, it may happen that there is higher amount of defects predicted in design phase but due to multiple preventive actions taken by the project manager, the actual defects count identified is less. As soon as the project manager enters the actual defects into the proposed framework based prediction model, the model corrects the predictions of future phases accordingly.

5. Result and Discussions

In this paper, proposed defect prediction framework has been validated on 15 real time projects of the same kind (based on Java technology and waterfall life cycle) and found that actual defects lie inside the range of predicted defects. Although there is a small deviation in some projects but that is well within the tolerance band of 2% to 5%. It is being evident from **Figure 8** that the actual defects are in line with the predicted defects. For the test results of pilot conducted on 15 projects, our proposed framework has accuracy of close to 90% .The regression value from **Figure 5** depicts a closer relationship between the predicted and actual defects.

In our experiments, the data sets with different network architectures have been used. The actual defects data from 45 completed projects was taken and used as a training data. Later, the framework was also tested for prediction of defects for newly started projects.

The quality of fit and the predictive quality found for each of the data sets have given very optimistic results. The prediction results indicate that the net (based on the proposed framework) tries to track the behavior of the full data set and sometimes its predicted value is more than the actual and sometimes less. The data set itself is an instance of random behavior. But what the net predicts is the number of faults that should occur, given the input variables based on the pattern it recognizes in the training data. The training data is the only data for the net to base its conclusions on, since that is the only information the net receives from the outer world.

Framework Comparison

The results of accuracy of close to 90% obtained by the proposed framework are comparable with, and even

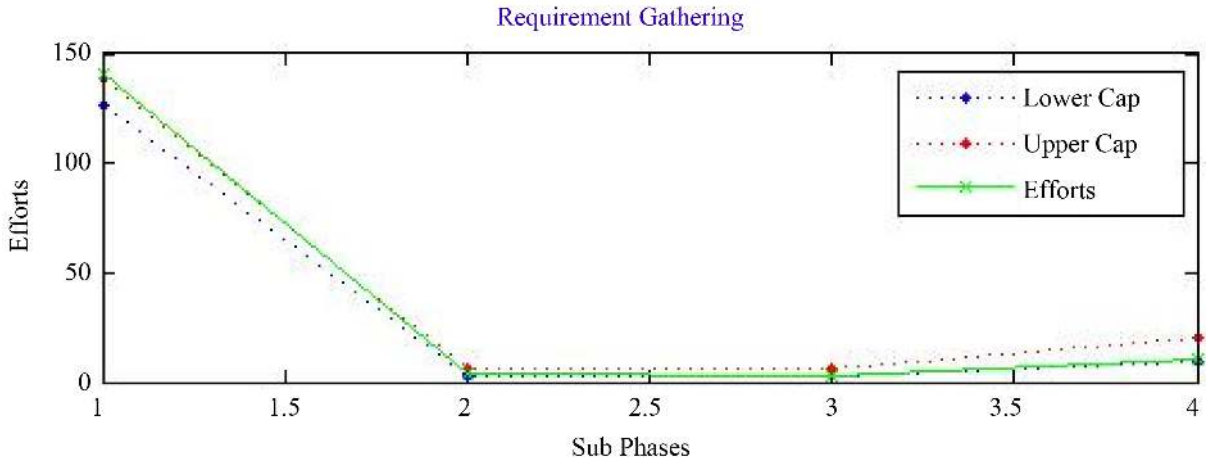


Figure 7. Input eligibility of requirement gathering phase.

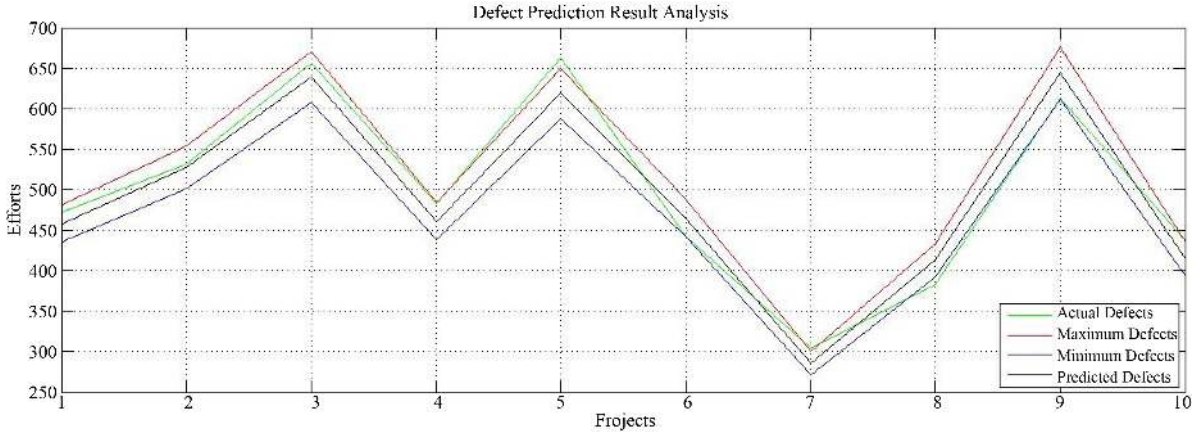


Figure 8. Defect prediction result analysis.

better than, the results with accuracy of 88.09% as obtained in [21], using public PROMISE library and Levenberg-Marquardt (LM) algorithm based neural network for predicting the software defects. Also, the data set used in our proposed framework is quite recent (year 2015) and the chosen technology (Java) is the widely used one in most software development scenarios.

Results were also compared with the ones obtained in [27], where evolutionary neural network (ENN) has been used to predict fault prone modules taken from a large software development project for which uniform crossover was used. The results even from best ENN Classifications were found to be close to only 75% of accuracy.

The same data set has also been compared with the linear regression based prediction model used in one of the large software development organization (refer to **Figure 9**). Linear regression based defect prediction model revealed an accuracy of 66% and only 10 projects from the sample of 15 were able to meet the predictions, where, as mentioned earlier, the proposed neural based framework has an accuracy of close to 90%.

6. Conclusion

Results from our experiments suggest that the proposed framework based on neural network approach possesses good properties from the standpoint of model quality of fit and predictive capability. Our conclusions are based on investigations of software development projects using java programming language and following waterfall life cycle model. It is expected that the proposed framework can be customized to suit other software development life cycle models like iterative and incremental development, agile etc. The investigation in those respects

DEFECTS	Project	Requirment	Analysis	Design	Coding	SIT	Total Defects	Variation from Predicted
Predicted Defects	P1	65	24	54	181	172	496	4.84%
Actual Defects		62	20	52	174	164	472	
Predicted Defects	P2	75	26	63	230	200	594	13.13%
Actual Defects		88	15	28	180	205	516	
Predicted Defects	P3	62	15	54	146	173	450	-18.22%
Actual Defects		74	11	71	172	204	532	
Predicted Defects	P4	68	16	59	180	189	512	-5.47%
Actual Defects		81	20	74	165	200	540	
Predicted Defects	P5	124	28	50	199	222	623	-5.30%
Actual Defects		137	26	60	204	229	656	
Predicted Defects	P6	48	21	60	198	150	477	-1.26%
Actual Defects		56	18	52	187	170	483	
Predicted Defects	P7	88	33	90	198	225	634	-4.57%
Actual Defects		108	20	95	225	215	663	
Predicted Defects	P8	16	5	85	150	175	431	-2.55%
Actual Defects		20	8	93	162	159	442	
Predicted Defects	P9	8	0	42	122	89	261	-16.86%
Actual Defects		13	3	54	130	105	305	
Predicted Defects	P10	2	2	88	199	79	370	-3.51%
Actual Defects		4	0	67	216	96	383	
Predicted Defects	P11	25	5	138	202	230	600	-2.17%
Actual Defects		30	11	120	233	219	613	
Predicted Defects	P12	15	11	43	132	184	384	-13.80%
Actual Defects		19	7	60	158	193	437	
Predicted Defects	P13	11	19	79	250	122	481	-5.20%
Actual Defects		7	15	95	295	94	506	
Predicted Defects	P14	51	4	148	198	278	679	-11.05%
Actual Defects		62	7	122	245	318	745	
Predicted Defects	P15	1	16	111	213	73	414	-5.80%
Actual Defects		2	17	119	221	79	438	

Figure 9. Defect prediction results from linear regression based model.

will be reported in subsequent communications.

References

[1] (1990) IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, 1, 84.

[2] Fenton, N.E. and Neil, M. (1999) A Critique of Software Defect Prediction Models. *IEEE Transactions on Software Engineering*, **25**, 675-689. http://dx.doi.org/10.1109/32.815326

[3] Levinson, M. (2001) Let's Stop Wasting $78 Billion per Year. CIO Magazine.

[4] Tamura, S. (2009) Integrating CMMI and TSP/PSP: Using TSP Data to Create Process Performance Models. Carnegie Mellon University, Pittsburgh.

[5] Sommerville, I. (2006) Software Engineering. Addison-Wesley, Harlow, England.

[6] Sivanandam, S.N. and Deepa, S.N. (2009) Principles of Soft Computing. 2nd Edition, John Wiley & Sons, Inc., Hoboken.

[7] Munakata, T., Ed. (2007) Fundamentals of the New Artificial Intelligence. Texts in Computer Science. Springer, London. http://dx.doi.org/10.1007/978-1-84628-839-5

[8] Narula, S.C. and Wellington, J.F. (1977) Prediction, Linear Regression and the Minimum Sum of Relative Errors. *Technometrics*, **19**, 185-190. http://dx.doi.org/10.1080/00401706.1977.10489526

[9] Gafhey Jr., J.E. (1984) Estimating the Number of Faults in Code. *IEEE Transactions on Software Engineering*, **SE10**, 459-464.

[10] Karunanithi, N., Malaiya, Y.K. and Whitley, D. (1991) Prediction of Software Reliability Using Neural Networks. *Proceedings of the International Symposium on Software Reliability Engineering*, Austin, 17-18 May 1991, 124-130. http://dx.doi.org/10.1109/issre.1991.145366

[11] Boetticher, G., Srinivas, K. and Eichmann, D. (1993) A Neural Net-Based Approach to Software Metrics. *Proceedings*

of the 5th International Conference on Software Engineering and Knowledge Engineering, San Francisco, 16-18 June 1993, 271-274.

[12] Boetticher, G. and Eichmann, D. (1993) A Neural Net Paradigm for Characterizing Reusable Software. *Proceedings of the 1st Australian Conference on Software Metrics*, University of Houston, 18-19 November 1993, 41-49.

[13] Boetticher, G. (1995) Characterizing Object-Oriented Software for Reusability in a Commercial Environment. Reuse' 95 Making Reuse Happen—Factors for Success, Morgantown, August 1995.

[14] Boetticher, G. (2001) An Assessment of Metric Contribution in the Construction of a Neural Network-Based Effort Estimator. *2nd International Workshop on Soft Computing Applied to Software Engineering*, Enschede, 8-9 February 2001, 234-235.

[15] Boetticher, G. (2001) Using Machine Learning to Predict Project Effort: Empirical Case Studies in Data-Starved Domains. *Workshop on Model-Based Requirements Engineering*, San Diego, 30 November 2001, 17-24.

[16] Kumar, S., Krishna, B.A. and Satsangi, P.J. (1994) Fuzzy Systems and Neural Networks in Software Engineering Project Management. *Journal of Applied Intelligence*, **4**, 31-52. http://dx.doi.org/10.1007/BF00872054

[17] Srinivasan, K. and Fisher, D. (1995) Machine Learning Approaches to Estimating Software Development Effort. *IEEE Transactions on Software Engineering*, **21**, 126-137. http://dx.doi.org/10.1109/32.345828

[18] Boetticher, G.D. (2003) Applying Machine Learners to GUI Specifications in Formulating Early Life Cycle Project Estimations. In: Khoshgoftaar, T.M., Ed., *Software Engineering with Computational Intelligence*, Springer, New York, 1-16. http://dx.doi.org/10.1007/978-1-4615-0429-0_1

[19] Khoshgoftaar, T.M., Ed. (2003) Software Engineering with Computational Intelligence. Springer, New York. http://dx.doi.org/10.1007/978-1-4615-0429-0

[20] Gayathri, M. and Sudha, A. (2014) Software Defect Prediction System using Multilayer Perceptron Neural Network with Data Mining. *International Journal of Recent Technology and Engineering*, **3**, 54-59.

[21] Singh, M. and Salaria, D.S. (2013) Software Defect Prediction Tool based on Neural Network. *International Journal of Computer Applications*, **70**, 22-28. http://dx.doi.org/10.5120/12200-8368

[22] Crosby, P. (1979) Quality Is Free: The Art of Making Quality Certain. McGraw-Hill, New York.

[23] Singh, S. and Singh, M. (2012) Software Defect Prediction using Adaptive Neural Networks. *International Journal of Applied Information Systems*, **4**, 29-33. http://dx.doi.org/10.5120/ijais12-450612

[24] Katiyar, N. and Singh, R. (2011) Prediction of Software Development Faults Using Neural Network. *VSRD-IJCSIT*, **1**, 556-566.

[25] Juran, J. and Gryna, F. (1988) Quality Control Handbook. 4th Edition, McGraw-Hill, New York.

[26] Neural Network Toolbox™ User's Guide, 2015.

[27] Hochman, R., Khoshgoftaar, T.M., Allen, E.B. and Hudepohl, J.P. (2003) Improved Fault-Prone Detection Analysis of Software Modules Using an Evolutionary Neural Network Approach. In: Khoshgoftaar, T.M., Ed., *Software Engineering with Computational Intelligence*, Springer, New York, 69-100. http://dx.doi.org/10.1007/978-1-4615-0429-0_4