# *Predicting Software Reliability using Computational Intelligence Techniques: A Review*

Kulamala Vinod Kumar, A. Sarath Chandra Teja, Abha Maru, Yogesh Singla, Durga Prasad Mohapatra
Dept. of Computer Science and Engineering
National Institute of Technology, Rourkela, India
email: 517cs1007@nitrkl.ac.in, email: 714cs1044@nitrkl.ac.in, email: 217cs3313@nitrkl.ac.in,
email: 115cs0243@nitrkl.ac.in, email: durga@nitrkl.ac.in.

***Abstract*— Software measurement is yet in an infant stage. There is hardly any efficient quantitative method to represent software reliability. The existing methods are not generic and have many limitations. Various techniques could be used to enhance software reliability. However, one has to not only balance time but also cater to budget constraints. Computational Intelligence (CI) techniques that have been explored for software reliability prediction have shown remarkable results. In this paper, the applications of CI techniques for software reliability prediction are surveyed and an evaluation based on some selected performance criteria is presented.**

***Keywords-Software reliability; Assessment; fault prediction; Computational intelligence techniques.***

## I. INTRODUCTION

### *A. Software Reliability*

Software reliability is often defined as the probability of achieving a software operation that is failure free in a specific environment over a specified duration of time. Software reliability indeed is an important factor to be considered when designing a system and predicting its reliability. It is unlike hardware reliability that reflects manufacturing perfection, software reliability on the other hand manifests design perfection. The factor that affects software reliability problems profoundly is rather the high complex nature of software itself.

A few researchers have come up with models that consider software reliability as a function of time. CI based modeling of software reliability is being widely explored now. However, care has to be taken when selecting an appropriate model for ensuring it best suits for a given case.

### *B. Software Failure Mechanisms*

This section lists some of the reasons that lead to software failures. Failures often occur due to ambiguities, oversights and human errors. Misinterpretations of specifications also lead to unachievable expectations. Sloppiness and incompetence in writing of code, insufficient testing and unforeseen usage of given software, also lead to unexpected complications. Techniques employed for enhancing hardware reliability cannot be used for enhancing software reliability. The inherent difference in the nature of these entities leads to failures that are different, because of the failure mechanisms itself. Hardware faults are typically physical faults. On the other hand, software faults are actually design faults. Design faults are difficult to visualize and classify. Hence, detecting and correcting software faults are often not easy tasks. These design faults are associated with fuzzy human factors and with the design process as well. Often, a clear understanding of these attributes is not perceivable. Design faults in hardware are also like to exist, but these are physical faults that are comparatively easily detectable. Finding software faults when designing software, is not easily perceivable, unlike in hardware manufacturing. Here, how software modules are uploaded affects the whole process. Quality of software cannot be enhanced or reduced once the software is uploaded and is in running state. Therefore, we cannot achieve higher reliability by duplicating the same software module.

The remaining part of the paper is organized as follows: Section 2 presents motivation and objectives. Section 3 describes various computational intelligence techniques. Section 4 presents literature survey. Section 5 discusses the evaluation criteria and Section 6 concludes the paper.

## II. MOTIVATION AND OBJECTIVE

### *A. Motivation*

When developing software for critical applications, it is utmost necessary to achieve higher degree of reliability. Today, this is a very challenging task for the software Industry. Today's software is expected to be self-adaptable and therefore often is more complex. Achieving reliability for a complex system, calls for optimizing a multi-objective problem in many domains. Over the past four decades, numerous Software Reliability Growth Models (SRGM) for better reliability prediction, have been proposed. Though, now there are a range of available reliability models, but none of the models work optimally across various projects.

### *B. Objective*

Computational Intelligence techniques yield better results in predicting than statistical methods and therefore, can be employed for predicting software failures more accurately [1]. In this paper, we essentially attempt to survey and assess

the use of Computational Intelligence techniques in software reliability prediction.

## III. COMPUTATIONAL INTELLIGENCE

Computational Intelligence is a term used to refer the ability of a computing device to learn specified tasks from data or by observing experimental outcomes. In some cases, it employs a combination of techniques such as Artificial Neural Networks (ANN), Evolutionary Computing, Learning Theory, and Fuzzy Logic. It may also use probabilistic methods that help in dealing with uncertainty and imprecision. Fuzzy logic empowers computers to understand natural language while, Artificial Neural Networks enables systems to learn experimental data by operating like a biotic neuron. Evolutionary computing techniques are inspired by biological processes that are based on natural selection.

## IV. LITERATURE SURVEY

In this section, we discuss the available work on software reliability prediction.

Wang et al. [2] have explored Non-Homogeneous Poisson process (NHPP) to optimize and enhance the effectiveness of the software reliability model. Their approach is based on a function that fits the logarithmic difference between the observed and estimated values using exponential distribution. They repeatedly apply this function on an available historical faulty software data set. Their results show that the logarithmic difference gradually tends to zero, as the number of fittings is increased. The trend, they observed that the logarithmic difference approaches to a stable value over a period of time, helps in building a prediction model that can predict the number of remaining faults in the software testing process. Their solution is far more optimized and is indeed achieved in an optimized manner. Their prediction model fits the historical faulty data set more accurately. It predicts the remaining amount of faults more successfully than other customary NHPP methods that have been used in software testing so far.

Owhadi-Kareshk et al. [3] have proposed a shallow Artificial Neural Network (ANN) that employs a pre-training technique. Their results suggest that though a shallow ANN has few hidden layers, it prevents over-fitting as in deep ANNs. But at the same time, shallow ANN enhances the accuracy as it helps in escaping from local minima. They test their proposed approach with almost seven different datasets taken from NASA codes. These data sets are available in the PROMISE repository. To evaluate their approach, they further compare with four SVM-based classifiers and also with a regular ANN that is not pre-trained. Their results show that pre-training helps in improving accuracy as it achieves the best overall ranking of 1.43 among seven datasets considered for testing. Their proposed approach achieved higher accuracy in at least four of them. ANN and SVM based approaches were found to be best only for two and one datasets, respectively.

Yohannese et al. [4] have explored and examined Ensemble Learning Algorithm (ELA) to enhance the prediction possibilities of fault proneness in software modules. They proposed an ELA based on Feature Selection (FS) and Data Balancing (DB). Their proposed framework with FS and DB combined techniques proved to be more robust and yielded higher performance. Figure 1 [4] depicts the Ensemble based learning framework as proposed in [4].

Li and Pham [5] have proposed a NHPP that also considers imperfect debugging (ID) along with operating environment uncertainty. Software is often tested in a given controlled environment. When used to other different operating environment by different user, it is likely to give different outputs than as expected. Developers may not be aware of such outcomes. Many SRGM models have been developed that are based on NHPP. But, the assumption of all these kind of models is that, both the operating environment and developing environment are similar. The unpredictability of the uncertainty of the operating environments, in which the software will run, influences the performance and affects the reliability in an unpredictable way. They have proposed a novel method for detecting software faults, based on testing coverage. They essentially consider the effect uncertainty of operating environments. They compared the performance of their proposed approach with several other NHPP based SRGMs and implemented it on three sets of real software failure data. They considered seven criteria. One of the criteria they considered was Improved Normalized Criteria Distance (NCD). This method was used to rank and select the best model with a goodness-of-fit criterion taken as a major perspective.

Zhu et al. [6] have considered software fault dependence and imperfect fault removal in their proposed reliability model. Their model is basically a Non-Homogeneous Poisson Process (NHPP). There are two types of faults – Type I fault which is an independent fault and type II fault which is dependent. They proposed a two-phase debugging process comprising of Phase I and Phase II. A small section of software faults that is usually unavoidable by software testers is considered in these two phases of the proposed model. They proved the effectiveness of their proposed model by testing with three datasets collected from various industries.

Kaur and Kaur [7] have developed a method to detect faults by taking object-oriented metrics. They also consider code smells as the predictors. They probed the impact on fault detection by not sampling, and further by using sampling methods. They tested their model with Apache POI 3.0 and Apache ANT 1.5 applications. They have considered various metrics and code smells as predictors and incidence of bug as response variable while applying various machine learning algorithms. Their observations are as follows. 1) By not considering any sampling method, the combination of software metrics and software code smells, performed better than only object-oriented software metrics for predicting occurrence of bugs in a class. 2) Resampling technique with metrics as predictors yielded better results than when no sampling was done with metrics as predictors. 3) K-star was found to have better performance than other machine learning algorithms when RESAMPLE technique was applied with metrics taken as predictors.

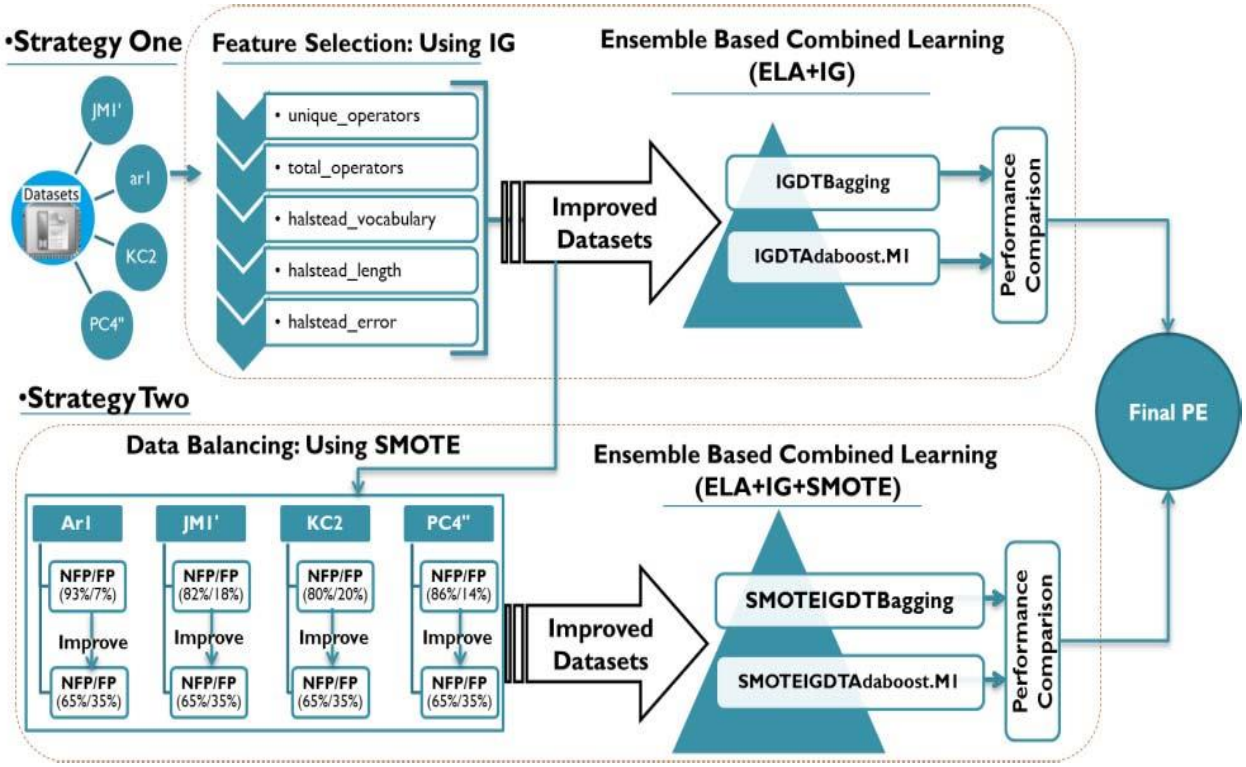


**Figure 1. Ensemble based combined learning [4]**

Tian and Noore [8] have attempted to predict software cumulative failure time. They have proposed a model based on evolutionary neural networks. They achieved better performance than existing neural network models. They have shown the impact of the neural network architecture as a deciding factor while measuring the performance of the network. Figure 2 [8] depicts the Evolutionary Neural Network Framework model that is employed for software reliability prediction as proposed in [8].

Kamei et al. [9] presented a performance analysis of various CI techniques when used in software fault prediction. They compare the prediction performance of an SVM based prediction method with other conventional methods such as logistic regression, linear discriminant analysis, neural network and classification trees. Their results show that SVM based software fault prediction model is far better than other conventional models.

Bhuyan et al. [14] explored machine learning techniques for software reliability prediction. They applied enhanced version of fuzzy min-max algorithm combined with recurrent neural network (FMMRNN). They applied their developed model on some of the software systems data sets. These data sets are collected when the system was in system testing phase combined with fault elimination. They tested the performance of their developed model by applying to a failure dataset collected from distributed system application. The main advantage of their FMMRNN is, the models complexity is automatically adjusted to the complexity of the failure history. When applied to different software systems, their model is robust and shown better performance according to next-step-predictability. So their model is a generic model. Figure 3 [14], depicts the architecture of FMMRNN model as proposed in [14].

Bhuyan et al. [15] explored prediction of software reliability by applying machine learning technique. They developed a software reliability prediction model using Recurrent Neural Network model combined with Back-propagation Through Time (RNNBPTT). They applied their developed model on software systems data sets. These data sets are collected when the software system was in system testing phase. Their procedure is little bit complicated, but the results of their work showed that Recurrent Neural Network produces better and stable performance in prediction of software reliability.

Bhuyan et al. [16] applied non-parametric neural network for predicting software reliability. The failure history datasets considered by them have parameters called failure time interval, number of failures etc. They also applied the feed-forward neural network with back-propagation for predicting the reliability of software systems. They compared their results with traditional parametric software reliability growth models. Their model produces better and consistent behavior in prediction of software reliability with respect to accuracy.

Ray and Mohapatra [17] proposed a technique based on hoe individual classes are contributing to the overall reliability of the system. According to their approach, they prioritized the classes at design phase. They calculated the

priority of a class using operational profile, use case diagram and sequence diagrams.

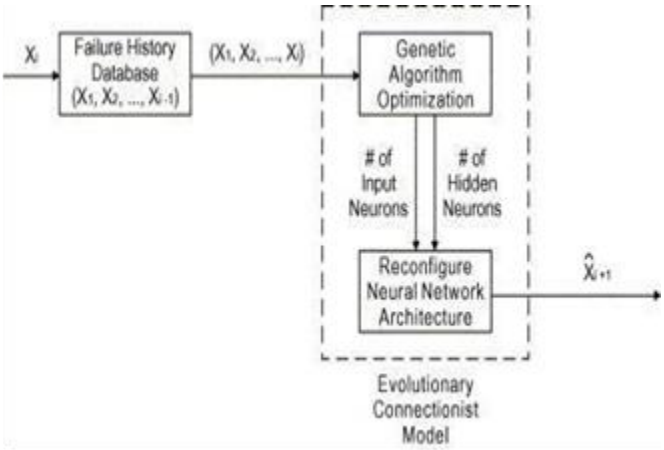


**Figure 2. Evolutionary neural network framework for software reliability prediction [8]**

The class will be tested according to its priority value. They developed an algorithm for event prioritization which takes sequence diagram of each use case and probability of each scenario of the use case and produces the output as the probability of each event activated in the use case. Then they generated an optimal test suite using a genetic algorithm-based technique. This technique selects test cases for a test suite out of a large pool of test cases.

Ray et al. [18] Small bugs always remain even after thorough testing of a program. Bugs and errors can be seen randomly scattered in the code. Not every bug in the code gives equally severe failure; different bugs in different parts of the program may cause failures with different frequency and severity than the rest part of the codes. Therefore, the ones giving frequent and higher severity failures can be prioritized. A metric is proposed to calculate the influence of a particular object in an object oriented program. The kind of effect a element causes shows the potential of that element to cause failures. The influence value determines the intensity with which each element is tested. Higher the influence value, higher is the intensity with which each element is tested. Comparison of the given scheme with other related schemes was done experimentally. Their approach is useful in applications of coding, debugging, test case design and maintenance, as well. The failure rate is reduced by their approach when the software is executed for some duration just after the testing phase is completed.

Kumar et al. [11] proposed a fault prediction model based on Least Square Support Vector Machine (LSSVM) with Linear, Polynomial and Radial Basis Function Kernels assisted with feature selection method. They essentially identified and investigated the predictive power of metrics used in fault prediction. Their experiments were conducted on 30 Open Source Java projects. Their results successfully revealed the faulty classes with a median fault identification efficiency of 46.206 percent. It was established that by eliminating almost 70 percent of available source code metrics after identifying a small subset of metrics, did not adversely affect the prediction accuracy. Metrics such as Measure of aggregation (MOA), Cohesion among methods of class (CAM), Average method complexity (AMC), Coupling between methods (CBM), Lines of code (LOC), Number of children (NOC) were found to have a significant impact. However, at the moment their prediction model can only identify a faulty class. Identifying a possible number of bugs in a class could be explored with the help of enhanced soft computing methods. Also, their model is only tested with Object Oriented Software. The model needs to be verified with software paradigms in order to generalize.

### *A Summary of Literature Survey*

Park and Baik [10] described a dynamic technique which selects and combines multiple software reliability models based on the decision trees learning of multi-criteria. [11-13] have explored various variants of Support Vector Machines (SVM) as well. A number of modelling techniques used to predict software reliability were surveyed by us. Table 1, presents a summary of the various computational intelligence techniques that have been explored for software reliability prediction

## V. EVALUATION CRITERIA

Precision, Recall and F1-measure are the most commonly used criteria while evaluating the performance of various software reliability prediction models.

Precision denotes the ratio of actual fault prone modules to the modules predicted as fault prone. Precision helps in knowing what proportion of positive identifications was actually correct. It is defined as:

$$\text{Precision} = TP / (TP+FP) \quad (1)$$

Where TP means True Positive and FP means False Positive. Recall is the ratio that suggests the number of correctly predicted fault-prone modules to the actual number of existing fault prone models. Recall attempts in knowing what proportion of actual positives were identified correctly. It is mathematically defined as follows:

$$\text{Recall} = TP / (TP+FN) \quad (2)$$

Where TP means True Positive, FP means False Negative. F1-measure is a term that signifies a combined value of recall and precision, and it is defined as shown in equation (3).

$$F1 = (2*\text{Recall}*\text{Precision})/(\text{Recall}+\text{Precision}) \quad (3)$$

Accuracy is another important metric for measuring the success of fault prediction. It is the ratio of the number of correctly predicted samples in the test set to the total number of samples actually taken in the test set. It is mathematically expressed as follows:

Where TP means True Positive, FN means False Negative, TN means True Negative, and FP is False Positive.

Accuracy = (TP+TN) / (TP+FN+TN+FP) (4)

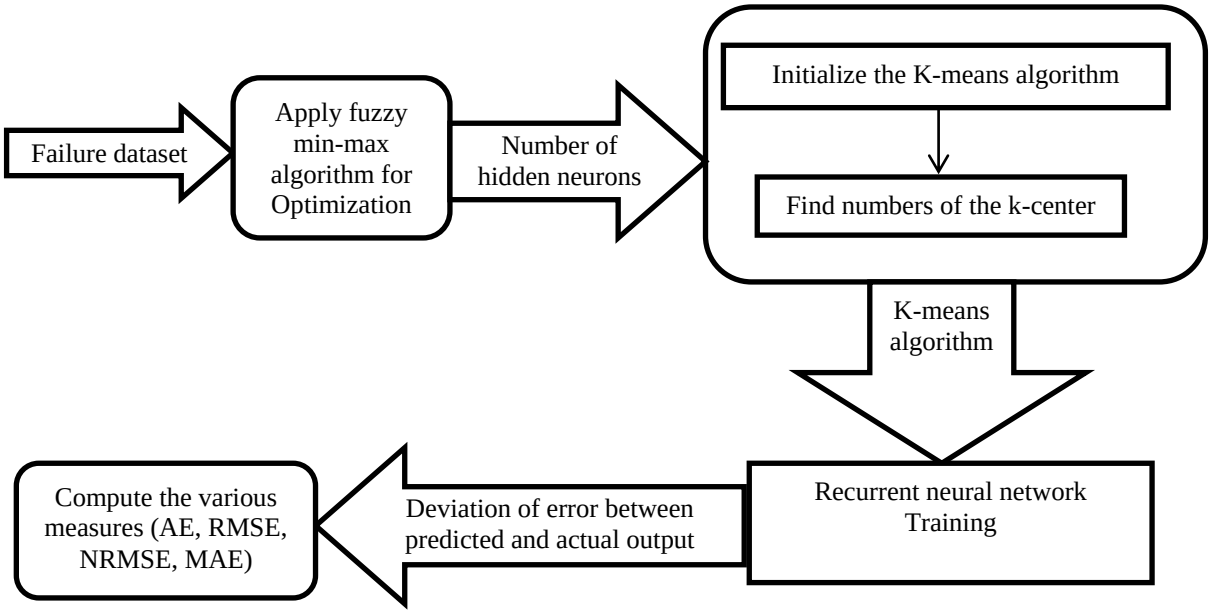


**Figure 3 Architecture of FMMRNN model for software reliability prediction [14]**

TABLE I. SUMMARY OF DIFFERENT COMPUTATIONAL INTELLIGENCE TECHNIQUES FOR SOFTWARE RELIABILITY PREDICTION

| Author | Datasets used | Algorithm | Evaluation Measure |
|---|---|---|---|
| Q. Li and H. Pham [5] | DS1:38 faults identified in 14 weeks, DS2:144 faults identified in 17 weeks, DS3: 146 faults identified in 60 months | NHPP | MSE, correlation index of regression curve, Predictive ratio risk, Predictive power |
| Owhadi-Kareshk et al. [3] | Seven datasets from PROMISE repository, NASA. | Pre training technique for shallow ANNs (ANNs with less number of hidden layers) | Accuracy |
| Yohannese et al. [4] | Eight datasets from PROMISE repository NASA | Ensemble Learning Algorithms (ELA) | AUC and Accuracy |
| Zhu and Pham [6] | DS1(failure data from real-time control system), DS2(failure data from wireless network switching center)and DS3(failure data from online bug tracking system) | NHPP | MSE and predictive-ratio risk (PRR) |
| Tian and Noore [8] | DATA-2: application having of 21,700 assembly instructions and 136 failures, DATA-11: flight application having of 10,000 LOC and 118 failures, DATA-12: flight application having of 22,500 LOC and 180 failures, DATA-13: flight application having of 38,500 LOC and 213 failures. | Genetic algorithm optimization and reconfiguration with Neural Network Architecture | Relative Error(RE), Average Relative Error (ARE) |
| Wang et al. [2] | DS1 was composed of data from 1500 telephone subscribers, DS2 was from a wireless network product, .DS3 was taken from a bug tracking system | NHPP | MSE |
| Kamei et al. [9] | The target software size is 300,000 lines of code (SLOC) and it has 514 modules, 277 modules are not-fault-prone and 237 modules are fault-prone | SVM with various kernels | F1 score |
| Bhuyan et al. [14] | Data set DBS-1 with 136 failures Dataset DBS-2 with 191 failures Dataset DBS-3 with 397 failures | Fuzzy min-max algorithm and Neural networks | AE, RMSE, NRMSE, MAE. |
| Ray and Mohapatra [17] | Sequence diagram | Genetic algorithm | No. of test cases |
| Kumar et al. [11] | 30 Open Source Java projects | LSSVM with Linear, Polynomial, RBF Kernels assisted with feature selection method | Accuracy, F-measure |

## VI. CONCLUSION

Software reliability is a very important aspect of software quality. Ensuring that the occurrences of software faults/failures is eliminated or by far minimized can only make the software reliable. Software reliability is not only dynamic but also stochastic in nature. It may be accounted as a probabilistic measure that considers the occurrences of failures of software as a random phenomenon. Many researchers are working on finding an optimal SRGM model for employing in their specific case of study. This is an open problem and needs more avenues to be explored. Existing software reliability prediction tools and techniques are inadequate and cannot be applied confidently. Existing approaches consider only a limited number of criteria when designing reliability prediction models. In this paper, various computational intelligence techniques such as SVM, MLP, Bagging, FFBPNN, CFBPNN, Regression, Neural Network ensembles, Radial-Bias Neural Networks, GRNN, ANFIS, NHPP, M5P, Instance based learning etc. that have been applied in software reliability prediction are surveyed and evaluated based on some selected criteria .

## REFERENCES


[1] R. Malhotra, A. Negi (2013) “Reliability modeling using particle swarm optimization,” The society for reliability engineering, quality and operations management (SREQOM), India and The Division of Operation and Maintenance, Lulea University of Technology, Sweden, Int J Syst Assur Eng Manag, , Vol. 4, Issue 3, September 2013, pp. 275–283, doi: 10.1007/s13198-012-0139-0.

[2] J. Wang, Z. Wu, Y. Shu, Z. Zhang, "An optimized method for software reliability model based on nonhomogeneous Poisson process," Applied Mathematical Mlling, Vol. 40, Issues 13–14, 2016, pp. 6324-6339, doi:10.1016/j.apm.2016.01.016.

[3] M. Owhadi-Kareshk, Y. Sedaghat and M. Akbarzadeh-T., "Pre-training of an artificial neural network for software fault prediction," 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, 2017, pp. 223-228, doi:10.1109/ICCKE.2017.8167880.

[4] C. W. Yohannese, T. Li, M. Simfukwe and F. Khurshid, "Ensembles based combined learning for improved software fault prediction: A comparative study," 2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE), Nanjing, 2017, pp. 1-6, doi:10.1109/ISKE.2017.8258836.

[5] Q. Li and H. Pham, "NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage," Applied Mathematical Modelling, Vol. 51, 2017, pp. 68-85, doi:10.1016/j.apm.2017.06.034.

[6] M. Zhu and H. Pham, "A two-phase software reliability modeling involving with software fault dependency and imperfect fault removal," Computer Languages, Systems & Structures, Vol. 53, 2018, pp. 27-42, doi:10.1016/j.cl.2017.12.002.

[7] K. Kaur and P. Kaur, "Evaluation of sampling techniques in software fault prediction using metrics and code smells," 2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Udupi, 2017, pp. 1377-1387, doi:10.1109/ICACCI.2017.8126033.

[8] L. Tian and A. Noore, "On-line prediction of software reliability using an evolutionary connectionist model," Journal of Systems and Software, Vol. 77, Issue 2, 2005, pp. 173-180, doi:10.1016/j.jss.2004.08.023.

[9] Y. Kamei, A. Monden, and K. Matsumoto "Empirical Evaluation of SVM-Based Software Reliability Model," Proc. Fifth ACM/IEEE Int'l Symp. Empirical Software Eng. vol. 2 pp. 39-41 2006.

[10] J. Park and J. Baik, "Improving software reliability prediction through multi-criteria based dynamic model selection and combination," Journal of Systems and Software, Vol. 101, 2015, pp. 236-244, doi:10.1016/j.jss.2014.12.029.

[11] L. Kumar, S. K. Sripada, A. Sureka, S. K. Rath, "Effective fault prediction model developed using Least Square Support Vector Machine (LSSVM)," Journal of Systems and Software, Vol. 137, 2018, pp. 686-712, doi:10.1016/j.jss.2017.04.016.

[12] F. Xing, P. Guo and M. R. Lyu, "A novel method for early software quality prediction based on support vector machine," 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), Chicago, IL, 2005, pp. 213-222, doi: 10.1109/ISSRE.2005.6.

[13] J. Cai and Y. Li, "Classification of Nuclear Receptor Subfamilies with RBF Kernel in Support Vector Machine," Proc. Advances in Neural Networks, Springer Berlin Heidelberg, 2005, pp. 680-685, doi:10.1007/11427469_108.

[14] M. K. Bhuyan, D. P. Mohapatra, and S. Sethi, “Software Reliability Prediction using Fuzzy Min-Max Algorithm and Recurrent Neural Network Approach,” International Journal of Electrical and Computer Engineering (IJECE), Vol. 6, 2016, pp. 1929-1938, doi:10.11591/ijece.v6i4.9991.

[15] M. K. Bhuyan, D. P. Mohapatra, and S. Sethi, “Prediction Strategy for Software Reliability Based on Recurrent Neural Network,” Proc. Computational Intelligence in Data Mining – volume 2, Springer India, 2016, pp. 295-303, doi:10.1007/978-81-322-2731-1_27.

[16] M. K. Bhuyan, D. P. Mohapatra, S. Sethi, and S. Kar, "An Empirical Analysis of Software Reliability Prediction Through Reliability Growth Model Using Computational Intelligence," Proc. Computational Intelligence in Data Mining - Volume 2, Springer India, 2015, pp. 513-524, doi: 10.1007/978-81-322-2208-8_47.

[17] M. Ray and D. P. Mohapatra, “A Scheme to Prioritize Classes at the Early Stage for Improving Observable Reliability,” Proc: 3rd India Software Engineering Conference (ISEC 10), ACM, 2010, pp. 69-72, doi: 10.1145/1730874.1730889.

[18] M. Ray, D.P. Mohapatra “Reliability Improvement Based on Prioritization of Source Code,” in Distributed Computing and Internet Technology (ICDCIT 2010), Lecture Notes in Computer Science, vol 5966, T. Janowski, H. Mohanty, Eds. Springer, Berlin, Heidelberg, 2010, pp. 212-223.