Section A-Research paper
ISSN 2063-5346

A Hybrid SVR based non-linear parametric estimation model for software reliability estimation

Anusha Merugu[1],Research Scholar, Department of Computer Science and Engineering, JNTUH
Dr. M. Chandra Mohan[2], Professor , Department of Computer Science and Engineering, JNTUH

Article History: Received: 25.03.2023 Revised: 06.05.2023 Accepted: 26.06.2023

DOI: 10.31838/ecb/2023.12.6.89

Abstract
Detection and analysis of software defects at a very early stage is very much essential in the domain of software engineering. It also influences the decision-making process related to allocation of resources for evaluation or verification. Software quality assurance can be defined as a significant phenomenon for the implementation of various machine learning techniques in defect detection. These techniques basically emphasize on single product-based software defects rather than the multi-product-based defects. Software reliability prediction models are used to predict the fault rate of the software systems using machine learning models. A large number of traditional reliability measures are used to test the software faults in the debugging and testing process. Most of the traditional machine learning based fault prediction models are integrated with standard software reliability growth measures for reliability severity classification. However, these models are used to predict the reliability level of binary class with less standard error. In this paper, a hybrid support vector regression-based non-linear parametric growth measure is implemented on the training fault datasets. Experimental results are simulated on various reliability datasets with different configuration parameters for fault prediction.
Keywords: Software fault detection, reliability prediction ,support vector machine.

1.Introduction

Detection and analysis of software defects at a very early stage is very much
essential in the domain of software engineering. It also influences the decision-making process related to allocation of resources for evaluation or verification.
Software quality assurance can be defined as a significant phenomenon for the implementation of various machine learning techniques in defect detection. These techniques basically emphasize on single product-based software defects rather than the multi-product-based defects.

Reliability in its simplest form means that a failure cannot occur within a certain period of time. The reliability concept thus stresses the probability, expected function(s), time and operating conditions of four components. Reliability also depends on the conditions of the system that may or may not change over time. Software systems have increased significantly in size and complexity in recent decades, and the trend is expected to continue in the future[1]. Computer reliability and accessibility, usability, performance, serviceability, capabilities and documentation are important attributes of software quality. Software reliability is difficult to achieve, since software complexity seems to be high. While it is difficult to achieve a certain degree of reliability of any highly complex system, including software, system developers tend to upgrade the software layer with complexity and rapidly developing system sizes.The Software Reliability Growth Model (SRGMs) is a software reliability model (SRMs) design recognition class which is converted into a mathematical model. The

reliability assessment of recent system updates is an important challenge in IT software management[2].

The probabilistic models are based on dynamic models and represented as time-based statistical distributions. All these models are used to predict current trends and to predict future trends in reliability. Probabilistic software reliability prediction models use statistical methods to estimate variables such as system error numbers, failure rates, software complexity and program failure, etc. There exists a number of software models in the literature, but none of them is ideal. The selection of an appropriate estimate model based on a specific application is a major research problem[3]. A data set that includes instances of defined classes and a test data set for which the class must be decided must therefore be entered. The quality of the data provided for learning, and also the type of algorithm used in machine learning, depends greatly on the ability to classify successfully. Categorical labels (discrete, unorderly) estimate classification results of continuously valued function models. It implies that numerical data values are expected instead of class marks to be incomplete or inaccessible. Regression analysis is the most widely used statistical method for numerical forecasting. Although other methods are available, the prediction also consists in identifying distribution trends based on available data. Genetic algorithms are also implemented to maximize the number of delayed input neurons and the number of neurons in the neural network's hidden architectural layer. We have demonstrated, using the software model for online adaptation, that good-fitness and next-step predictability is better than traditional methods when cumulative software failure times are forecast. Because those variables ' meanings are certainly not known. Many potential values can equate to the likelihood of occurrence. Therefore, we really don't know when the next loss will happen. We know only a few possible failure times and their likelihood. Two types of fault data, namely time-domain data and interval-domain data, were widely used in software reliability modeling. The time-domain form is determined by the time the failure occurred. Learning supervised is a methodology for machine learning to build a data structure for preparation. Maximum Likelihood Assessment (MLE) is a common statistical method for the determination of the probability distribution parameters underlying a given dataset. Throughout literature there are many predictive models of the reliability of software based neural networks, which are better known than certain statistical models[4-6]. Computer reliability is one of the key factors taken into account in maintaining the accuracy of the computer. Simply put, software reliability is about system failure or failure[7]. Success and success are two distinct variables commonly included in our software development. Fault could be identified as a fault or error during the development phase. In recent decades, the size of the object-oriented defects increases, the prediction of multi-level defects also increasing exponentially. The main objective of the software defect prediction models is to improve the true positive rate of the defects with minimum time and cost. Traditional software prediction classifiers are developed to assess the metrics in the application level. Bayesian network (BN), Naïve Bayes, SVM, linear regression approaches as well as bagging approaches are used to assess the software defects with limited feature space. Most of the traditional software defect prediction models are focused on limited defect features in a single application. A software test is a study to inform stakeholders about the quality of the tested product or service. A series of software error detection activities. Testing is a process that is used to detect computer software correctness, integrity and quality. A Software Defect / Bug is a condition of a software product which does not meet the expectations or requirements of the user (not specified but reasonable). In other words, a malfunctioning program or incorrect coding or logic error produces wrong, unintentional findings. The current forecasting work concentrates on estimating the number of faults in software systems; (ii) the discovery of fault associations and (iii) classification of fault-pronounced software components, which are typically faulted rather than fault-pronounced, in two

classes. The second type of work is carried out by the community association of data mining to disclose software defects that can be used for three purposes. This technique first finds a candidate concurrency bugs through code patterns. And then, it inserts noise injections at the candidate bug site in order to detect concurrency bugs with high probability in testing. Upon ConTest, this technique contributes to active testing of concurrent Java program. They adopted bug patterns for assisting code review process[8]. The authors extend the regular expression in Perl language for bug specifications and bug detections. As a preprocessing to code review by experts, this technique automatically attach the comments on a code which is corresponding to a given bug specification[9]. As the software industry evolves, the monitoring and enhancement of software quality is increasingly engaged in software businesses. In 1992 IBM conceived the Orthogonal Default Classification (ODC) in quantitative and qualitative assessment to satisfy these criteria. ODC utilizes a range of orthogonal characteristics, including the operation, cause, effect, destination, type of defect, defect status, origin and age, in order to categorize the defect[10]. By using those characteristics and their values, a defect may be categorized as a point in the Cartesian coordinates area. Once we obtain the ODC information from a project, software designers and testers will provide us with feedback when using some analysis characteristics. We can also obtain test effectiveness by examining defect kinds or skills by evaluating the distribution of the triggers and operations, identify weaknesses in design and code, and assess client use with impact analyses, triggers, defect types and quality assurance. Now it is commonly used in IBM, Motorola, Nortel, Lucent, etc. ODCs are commonly used. The ODC investigations today focus on the implementation in real job of defect management. People are interested in using ODC documents, and the author has suggested a method called the' Harmony Matrix' for better information collection. The author developed a matrix for harmonies, showed associations between combinations of kinds of triggers, recognized three categories of low, medium and high and utilized relations between faulty kinds and triggers in consultation with domain expert views. The matrix indicates that defect triggers are the most probable kinds of defects. For example, if a defect trigger is backward compatibility, the type of defect is high in the matrix, whereas if the type of defect is present, the type of defect is probably the same as the type of defect. It takes only two characteristics into consideration, the trigger and the error type, as other characteristics also influence each other and the technology does not use the historical information efficiently to help it. Software defect prediction is a significant guidance for studies into software reliability[11]. The technology for defect prediction can be used to discover high-risk software module. Software designers can focus on risky modules with more defects to save costly testing and time[13], and then use a restricted test funds for risky modules. The significant thing is to discover high-risk modules in the software goods for anticipating software errors. In the classification method the characteristics in the samples play various roles for issues of classification. At the same moment, the interaction between the different characteristics impacts classification performance[16]. Very little study focuses on relationships between attributes. In most times the characteristics of traditional algorithms are always presumed to be distinct during the classification phase. In practical issues however, the interplay of characteristics occurs. Therefore, when predicting software defects, the interaction between the characteristics must be taken into account. Fuzzy integral is a non-linear component, based on fuzzy measures. The non-additiveness of fluid measurements makes the interaction between classification features complementary to the fluid. The fluorescent measures corresponding to these are essential to achieving high-grade efficiency in the essential classification of the fluorescent. In general, the measurement of fluctuations is very complex[14]. The reciprocal data between characteristics is a significant tool for efficient assessment of the related degree between attributes in order to evaluate the correlations between the characteristics of data theory.

Static and dynamic defect prediction

Open-default information metrics are evaluated and a sub-set of smaller-scale software metrics are developed from current NASE project information from PROMISE. Research has shown that, compared to two other kinds of subset function algorithms, the suggested Algorithm enhances the efficiency of the three famously classified kinds. Most techniques of this approach extend the type systems of existing programming languages[15]. The extended type systems check a program correctly follows a given programming rules to ensure that the desired properties holds for the program. The type systems are normally implemented as a part of compilers. [16] Introduce an extended Java type system to avoid concurrency errors including deadlock and data race. This type system requires every shared member has a specification of its synchronization object. However, there are two shortcomings of this approach to be applied to general programs. First, these systems restrict programmers to write codes in simple manner strictly. Second, these methods require programmers to specify the additional information related to synchronization used in a code[17]. These two shortcomings are unfeasible for targeting system programs. In system programs, fine tuning of synchronization operations are common in order to improve performance. Moreover, the size of program is normally too large for programmers to give the additional information manually.

In current research, there are two primary techniques for measuring software, extracting and selecting the characteristics. Feature extraction is a method which produces fresh characteristics by transforming or combining the initial set of attributes. And choice of features is a method which chooses a subset of the most significant software quality characteristics from the initial set of characteristics using repeated selection and search tests. Some researchers think that an optimum place can be used by a single method of choosing characteristics. Therefore, techniques such as an ensemble technology can be promoted, which incorporates distinct selection techniques, not a single method, and an iteration technique that repeatedly re-examples the features. Software metrics are also used using other techniques such as correlation assessment, logistic regression, and mutual information analysis[18]. There are research. Dynamic analysis techniques aim to verify a certain property of a program by evaluating its actual executions. By observing internal states during target program executions, the dynamic analysis techniques can use accurate information of program behaviors. In dynamic analysis, it is possible to achieve value-sensitive and alias-sensitive analysis with much less computation cost than in static analysis. Dynamic analysis extends traditional testing to check meaningful properties using intermediate state information in program executions. Dynamic analytical methods share inherently the test constraints. Full evaluation for target programs can not be supported by dynamic analysis because the controlled partial conduct of the goal programs is used. The other restriction is that it is hard to apply dynamic analysis methods unless target programs are full. Executable settings and sample instances are required for the dynamic analysis technique. These can only be provided at the subsequent stage of software development, in particular for embedded software.

2. Related works

Kapur al. have developed various SRGMs concerning the growth rate software reliability index for error detection[19]. Ohishi et.al, proposed a measuring method as an indicator collection, gathering data for the testing of all those metrics[20]. Tarinejad al. suggested the non-homogeneous Poisson method-based software reliability growth pattern because the detection of these errors might also lead to detection of other errors without failure[21]. Nagaraju proposed an evolutionary model of the neural network to estimate and predict the software reliability based on a multimedia architecture input and output. In this study, the development of neural network models for software-reliability predictions was proposed using an Exponential and Logarithmic Encoding Scheme. Neural network

models with the two encoding schemes above have shown a better prediction of cumulative failures than some statistical models. Su [23] proposed a neural network approach focused on predictions of software reliability. They compared the approach to parametric model recalibration with some meaningful predictive measures with the same data sets. Roy et al.[24] proposed a system in which software reliability based on the neural network would be expected. They used the reverse propagation algorithm for instructions. Ravishanker [25] submitted a neural network approach focused on the evolutionary prediction of device reliability. They used single output architecture with multiple delayed inputs. Pham [26] suggested two models for cumulative system failure estimation, such as exponential neural network encoding (NNEE) and logarithmic encoding (NNLE). He encoded the data with the above two encoding scheme, i.e. the time of execution.

Li et al.[27] have proposed to reusit data from previous projects / releases for failure to boost early reliability for current projects / releases. Yazdanbakhsh et al.[28] proposed the combinational dynamic weighted model (DWCM) based on a neural network for the prediction of device reliability. The method was used on two sets of data and the effect was compared with certain statistical models. The neural network is a methodology for performance computation. The machine performance can previously be predicted on the basis of our neural network architecture. The system is also trained unless its desired output or destination can be achieved. For training purposes, we use different learning techniques that are freely described as supervised and unattended learning.[29] Software reliability is a quantitative study of every software designed since it affects directly software quality[30]. The classification system is trained with functions[31-32] from the history of software revision that classifies software changes as buggy or clean when applied. There was a 78 percent accuracy in the results. The changes were higher due to a small granular prediction and the seminal information on the source code was not required for classification. A wide range of programming languages are used for the changes of classification. The best way to model the software components at various failure levels is to have the strong back propagation algorithm based on the neural network. Yuan et al. introduced a software reliability evaluation methodology for Fuzzy-Neural. This paper identified an adaptive network-based fluid inference system (ANFIS) reliability prevision model to enhance the evaluation accuracy. The model uses the software's reliable information as input data (default lines every thousand lines), using reliability prediction as output data, the neural network trainings Adaptive–Fuzzy, membership of defect counters every thousand lines. This includes both evaluation and forecasting. During the evaluation phase, the selected scheme evaluates various systems of learning. In the prediction phase, a predictor with all historical data is then used with the best learning scheme. Finally, the predictor is employed for the prediction of the new data defect. Bayesian networks used Ahmet Okutan to determine the likelihood of influence between software metrics and defect proneness[33]. Chehade et.al compared the k-NN Network which has been implemented as either fault or non-default susceptible in classifying software components. Multiple Linear Regression (MLR), logistic regression for data modeling used the earliest attempts to model maintainability and defects with static source code metrics as predictors. The problem with MLR is that it is not easy to interpret relationships between predictor software metrics and response variables [34]. The choice of modeling technique greatly influences the accuracy of maintenance and prediction of defects models, but different prediction model comparison studies have reached different, inconsistent and divergent conclusions about the superiority of one modeling technique over the other models. Dyck et.al, [35] conducted a systematic review of the prediction of defects and found that two-thirds of the prediction of defects studies were based on private datasets and their results could not be verified. With a small number of private data sets, different experiment design, different measurements of accuracy and lack of application of statistical significance tests, it is not

possible to understand the strengths and weaknesses of different machine learning techniques. Saito [36] et al. argued that research on models of prediction should use public data sets and focus on finding relevant data on training. The work embodied in this thesis concerns the comparison with public data sets of a wide range of machine learning techniques for early maintenance and prediction defects. Public data sets allow other researchers to examine the validity of proposed models by replicating experiments and constructing reproducible or refutable models. The work in this thesis also evaluates machine learning techniques as predictor variables for defects with different metric categories such as source code metrics, micro-interaction metrics and software entropy metrics. Zheng et.al, presented an effective multi-objective naïve Bayes learning for cross-project defect prediction[37] . They introduced multi objective learning mechanisms and implemented those in cross project environments. This approach has three prime objectives and those objectives completely depend upon the process of class imbalance. In this piece of research work, a new algorithm known as harmony search algorithm is implemented. The above proposed algorithm has the responsibility of resolving multi objective Bayes issues. Numbers of solutions along with various PD, PF balance values are generated by analyzing the source data. After that NB or NBNN is constructed along with an individual optimal solution. Additionally, it can determine the fault proneness of the targeted data. It is responsible for producing frequent item sets for every individual partition. The item set has numbers of abnormalities and known as focused item set. Depending upon real item set, they introduced a new pre-processing technique which is responsible for setting real items those are missing in partition Pf only. These changed data have significant role during the development process of Naïve Bayes classifier. It is also responsible for detection of defective software modules. In the evaluation phase, the performance of NB model with ten bins is considered. It can be noticed that, this performance is not much satisfactory. It may either increase or decrease with respective to the inclusion of missing item sets.

Defect prediction using statistical models

Maintenance Index (MI) is a traditional model used to predict software application maintenance. It is defined by a4-meter polynomial equation [38] consisting of metrics of Halstead, cyclomatic complexity of McCabe, lines of code (LOC) and number of comments. Researchers criticized MI model as they found problems applying this model to large and diverse collections of mission-critical projects . Recently, Wu et al.[39] found that MI's predicted for five software system releases were the same where the actual maintenance effort observed to maintain these systems varied considerably. Peng et al.[40] proposed a linear software maintenance prediction model based on a minimum set of software design level metrics. Ye et.al,[41] studied for a maintenance period of three years two commercial object-oriented systems and developed a predictability model using Multiple Linear Regression (MLR). They studied C++ systems software maintenance using MLR as predictors with object-oriented metrics. Byun et,al, [42] studied the relationship with the MLR modeling technique between design metrics and sustainability. Machine learning techniques were not considered in these studies. When there is no acceptable theory that can relate maintenance to its software predictor metrics, parametric techniques such as MLR are not useful[43]. Machine learning techniques can therefore be used to predict the maintenance of software because they are non-parametric in nature. Turhan et.al, [44] used Bayes Network, Regression Trees and MLR to predict the maintenance of software. They concluded that only one system studied was superior to MLR by the Bayes Network. Hui et.al,[45] applied MLR, Neural Networks, Regression Trees, Support Vector Machine (SVM) and Multivariate Regression Splines (MARS) and concluded that one SVM system yielded better results while MARS yielded better results on another system. These studies[46-48] did not compare their results with other techniques of machine learning or MLR or used different measurements of accuracy, so the results were not comparable. Pham et al.[49] used fuzzy logic techniques for software

maintenance measurement. It was concluded that there were no obvious choices to build predictive models for maintainability. A number of techniques for machine learning were investigated in prediction of defects[50]. The nature of data sets for defect prediction is skewed. Non-defective modules are negative examples (or negative class, majority class) in terms of machine learning literature, and defective modules in training data are positive examples (or positive class, minority class). This is referred to as the problem of class imbalance. Class imbalance greatly degrades the performance of machine learning techniques. Data sampling is one solution to this problem. Another solution is to reduce the bias generated by negative examples by using a variety of learners and to achieve better predictive performance[51]. Li et.al, [52] investigated the methods of ensembles bagging and boosting over NASA MDP datasets and found that ensembles bagging and boosting were more accurate than single base classifiers (learners). In Bagging and Boosting Ensembles, they employed seven base learners. However, through statistical significance tests, they did not evaluate their models.

Zhao, et.al, proposed an advanced kernel-based technique in order to achieve better software defect detection [53]. The kernel completely depends upon the category of every individual software defect. Vallee, et.al, proposed a general software defect-proneness prediction framework [54]. It involves an unbiased and comprehensive comparison technique. A minor modification during the evaluation phase may influence the resulted outcomes significantly. This suggested technique is very much efficient for real world implementation irrespective of the nature of data. There are certain cases, where data is skewed. Model cannot predict sufficient number of defective instances for the process of learning. Suppose a method is performing very well in case of balanced dataset, it will result poorest performance in case of imbalanced dataset.

3.Proposed Model

In this section, a statistical quartile deviation-based improved SVR prediction model is proposed on the software reliability datasets. In this work, a hybrid exponential distribution based SVR model is implemented in order to predict the software reliability on the training and test software fault data. This model is integrated with the quartile deviation growth function in order to fit the S shaped curve. The main aim of this approach is to improve the prediction accuracy and to minimize the error rate for software quality and reliability estimation. The S-shaped models show the asymptotic behavior similar to the concave and exponential models. Therefore, the S-shape curve acts in the same way as the concave curve at later testing stages. In the proposed model, reliability estimation is performed in two phases. In the initial phase, quartile deviation based error estimation is calculated on the training data for software reliability prediction. In the second phase, a hybrid support vector regression model is designed and implemented on the computed S-shaped training data as shown in figure 1.

Section A-Research paper
ISSN 2063-5346

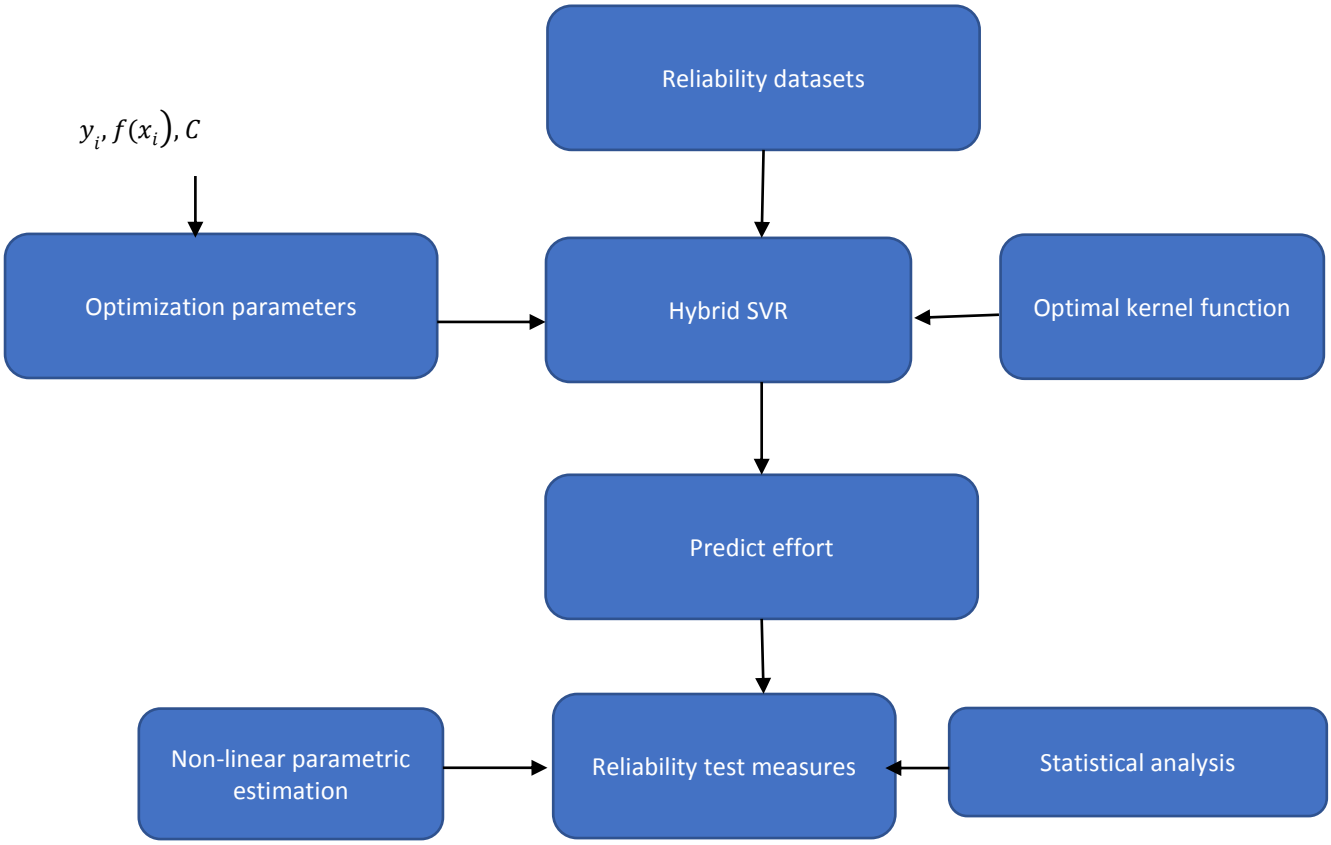

Figure 1: Proposed Framework

The following proposed SVR model is implemented on the fault data. Initially, input data is given to hybrid SVR model to predict the effort rate.

$$\min_{\xi_k,\xi_k^*} C(x) = \frac{1}{2} \| w \|^2 + \lambda.\psi(x).\phi(x) + b$$

$$\min_{\xi_k,\xi_k^*} C(x) = \frac{1}{2} \| w \|^2 + \lambda.|\xi_k - \xi_k^*|.\phi(x) + b$$

The prediction values of the SVR are tested using the non-linear parametric deviation model and maximized composite reliability measures. These measures are used to find the deviation, skewness and shape of the dataset.

In this phase, each new test sample is tested against the defect or non-defect type using the following classification based procedure. Initially, the mean and covariance matrix are computed on the training dataset and then proposed classifier is applied on it as P_i. Similarly, test data are added to the training data and then mean , covariance are computed to find the predicted classes as P^*_i. Finally, the new class is predicted using the argmin.

Section A-Research paper
ISSN 2063-5346

New defect class prediction

Input : Training data PD_c^{Tr}

Procedure:

1. To each instance in PD_c^{Tr}
2. Do
3. $\mu_i^p = \text{PMean}(\text{PD}_c^{\text{Tr}})$
   $\sum_i^p = \text{CovMat}(\text{PD}_c^{\text{Tr}})$
   $\text{P}_i = \text{classify}(\text{Ens}, (\mu_i^p, \sum_i^p))$
4. Done
5. To each test sample t in PD_c^{Ts}
6. $\mu_i^{p^*} = \text{PMean}(\text{PD}_c^{\text{Tr}} \cup \{\text{t}\})$
   $\sum_i^{p*} = \text{CovMat}(\text{PD}_c^{\text{Tr}} \cup \{\text{t}\})$
   $\text{P}^*_{\text{i}} = \text{classify}(\text{Ens}, (\mu_i^{p^*}, \sum_i^{p*}))$
7. find Δp_i using the (P_i, P^*_i) as Euclidean norm

$$\text{class(t)} = \text{argmin}(\Delta p_i)$$

Proposed Reliability test measure

The mathematical function used to find the mean time fault detection process is given as:

$$\text{F(t)} = \textbf{log} \mid \alpha.(\frac{\text{k}}{1+\text{k.exp}(\alpha)}) \mid ----- (1)$$

where $\alpha, \text{k}>\textbf{0}$

$$\text{f}_1(\text{x}) = \log(\text{x})$$

$$\text{f}_2(\text{x}) = \frac{\text{k}}{1+\exp(\text{x})}$$

$$\text{f}_3(\text{x}) == \alpha.\text{x}$$

$$\text{f}_4(\text{x}) = \log \mid \text{x} \mid$$

Now, these functions derive mean

value function as shown below:

Section A-Research paper
ISSN 2063-5346

$$f_4(f_3(f_2(f_1(x)))) = f_4(f_3(f_2(\alpha.\log(k))))$$
$$= f_4(f_3(\frac{k}{1+\exp(\alpha.\log(k))}))$$
$$== f_4(f_3(\frac{k}{1+k.\exp(\alpha)}))$$
$$= f_4(\alpha.(\frac{k}{1+k.\exp(\alpha)}))$$
$$= \mathbf{log}\,|\,\alpha.(\frac{k}{1+k.\exp(\alpha)})\,| = F(x) - - - (2)$$

Here, proposed maximization function in eq (2)

satisfies the composite function of four functions.

4.Experimental results

Experimental results are carried out on the software failure datasets taken form the DS1 reported by K.Okumoto. During 56 weeks of testing, a total of 124 faults are identified to test the stability. The second , third and fourth datasets DS2,DS3,DS4 are taken from Rome air development center(RADC) projects.

Akaike information criterion (AIC)

AIC = − 2(maximum log-likelihood) + 2(degrees of freedom),

where the degrees-of-freedom is generally identical to the number of free model parameters.

$$BIC = -2\ln(L) + k\ln(n)$$

where L is the likelihood of the model given the data, k the number of estimated parameters in the model, and n the number of entries in the database.

The "Kolmogorov–Smirnov" test (K–S test) is another nonparametric test to determine the equality between one-dimensional probability distributions.

Table 1: DS1 for fault prediction based on severity level

W	CF	Label
1	16	L
2	24	L
3	27	L
4	55	M
5	41	L
6	49	L
7	54	M
8	58	M
9	69	M

10	75	H
11	81	H
12	86	H
13	90	H
14	93	H
15	96	H
16	98	H
17	99	H
18	100	H
19	100	H
20	100	H

Table 2: DS2 for fault prediction based on severity level

W	CF	Label
1	28	L
2	29	L
3	29	L
4	29	L
5	29	L
6	37	M
7	63	M
8	92	H
9	116	H
10	125	H
11	139	H
12	152	H
13	164	H
14	164	H
15	165	H
16	168	H
17	170	H
18	176	H

Table 3: DS3 for fault prediction based on severity level

W	F	label
40	71	M
41	72	M
42	74	M
43	74	M
44	80	M
45	84	M

46	84	M
47	84	M
48	84	M
49	85	H
50	86	H
51	89	H
52	90	H
53	90	H
54	92	H
55	108	H
56	120	H
57	128	H
58	129	H
59	139	H
60	146	H

Table 4: DS4 for fault prediction based on severity level

W	F	Label
33	79	L
34	80	L
35	82	L
36	83	L
37	83	L
38	84	L
39	84	L
40	85	M
41	85	M
42	87	M
43	87	M
44	87	M
45	89	M
46	89	M
47	91	H
48	91	H
49	94	H

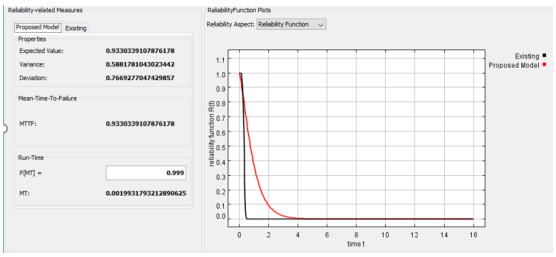

Figure 2: Mean time to failure rate and runtime of the proposed model to the exponential model.

Figure 2, describes the mean time to failure rate of the proposed model to the traditional exponential model on testing data. From the figure, it is clear that the present model has low error rate and better mean time to failure rate than the traditional model.

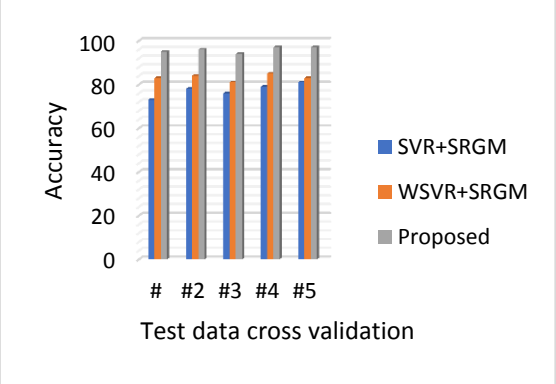

Figure 3: Comparison of proposed fault prediction model to existing weighted SGRM model on all datasets.

Figure 3, describes the performance of proposed approach to the conventional models for fault prediction. From the figure3, it is noted that the proposed non-linear kernel function based SVR model has better accuracy on four training datasets DS1,DS2,DS3 and DS4.

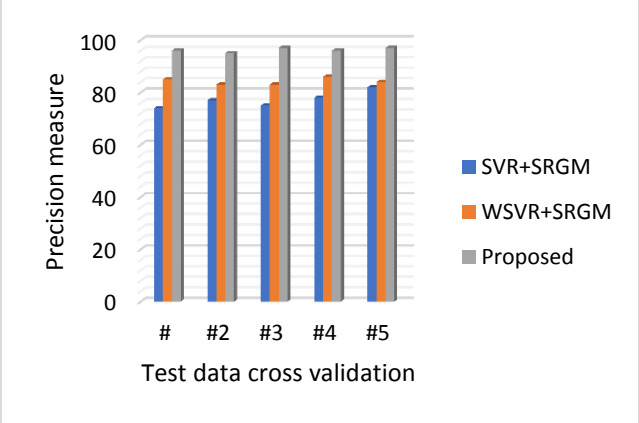

Figure 4: Comparison of proposed fault prediction precision to existing improved weighted SGRM model on all datasets.

Figure 4, describes the performance of proposed approach to the conventional models for fault prediction. From the figure3, it is noted that the proposed non-linear kernel function based SVR model has better precision on four training datasets DS1,DS2,DS3 and DS4.

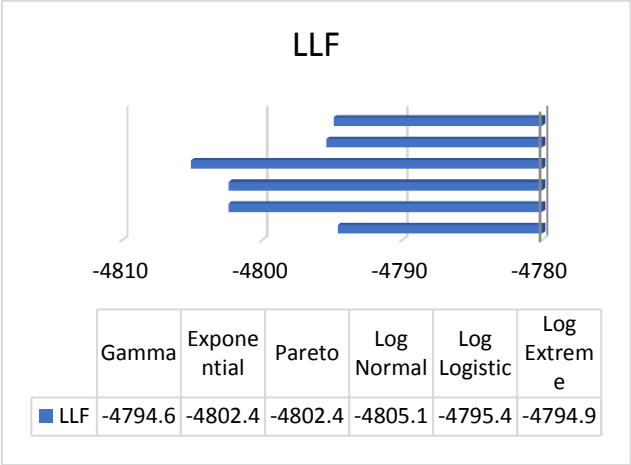

Figure 5:Average LLF of different traditional functions and its scores.

Figure 5, describes the conventional average log likelihood function estimation on all training datasets. Proposed non-linear exponential mode has -4594 rate than the conventional models for LLF estimation.

Table 5: Degrees of freedom for parametric estimation

Model	DF
Gamma	3
Exponential	2
Pareto	3
Log Normal	3
Log Logistic	3
Log Extreme	3
Proposed	3

Table 5, describes the degrees of freedom for parametric estimation process on the given training datasets. Here, each distribution and its degrees of freedom are tabulated for the software reliability

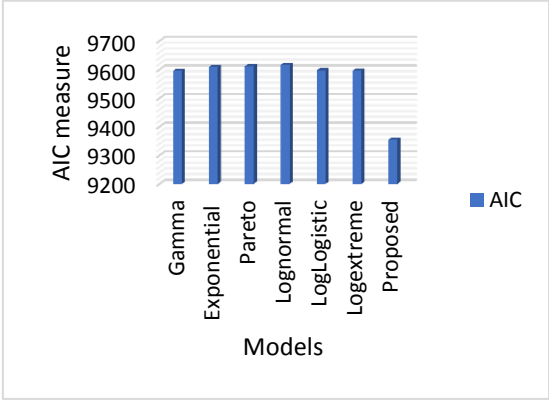

Figure 6: Comparative analysis of proposed AIC measure to the conventional AIC models on the training datasets.

Figure 6, describes the comparative analysis of proposed model to the conventional models on different software reliability models using AIC measure. From the figure, it is observed that the proposed model has less AIC measure than the conventional models on the different reliability datasets.

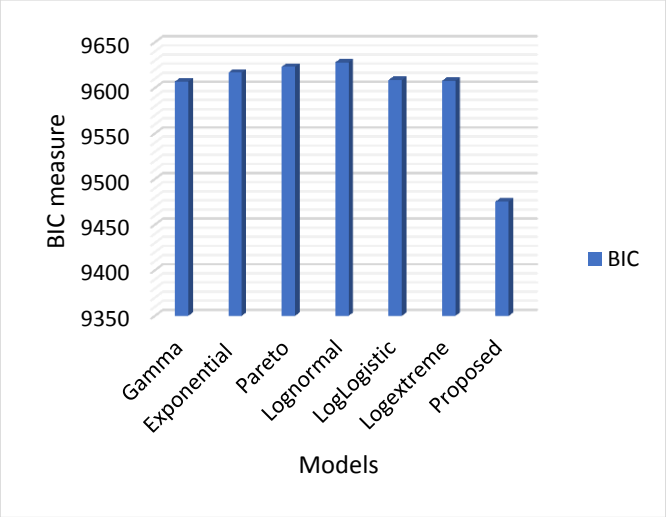

Figure 6: Comparative analysis of proposed BIC measure to the conventional AIC models on the training datasets.

Figure 6, describes the comparative analysis of proposed model to the conventional models on different software reliability models using BIC measure. From the figure, it is observed that the proposed model has less BIC measure than the conventional models on the different reliability datasets.

Table 6: Comparative analysis of KS testing measure on the different software reliability growth functions.

Model	**KS(90%)**
Gamma	FALSE
Exponential	TRUE
Pareto	FALSE
Log Normal	FALSE
Log Logistic	FALSE
Log Extreme	FALSE
Proposed	TRUE

5.Conclusion

Software reliability fault prediction plays a vital role in small- and large-scale software applications. In this paper, a hybrid support vector regression-based non-linear parametric model is implemented on the different software reliability datasets. Most of the traditional machine learning based fault prediction models are integrated with standard software reliability growth measures for reliability severity classification. However, these models are used to predict the reliability level of binary class with less standard error. Experimental results proved that the proposed reliability fault prediction model has better performance in terms of prediction and time are concerned.

References

[1]J.-Y. Park, G. Lee, and J. H. Park, "A class of coverage growth functions and its practical application," Journal of the Korean Statistical Society, vol. 37, no. 3, pp. 241–247, Sep. 2008, doi: 10.1016/j.jkss.2008.01.002.

[2]H. Soltanali, A. Rohani, M. H. Abbaspour-Fard, and J. T. Farinha, "A comparative study of statistical and soft computing techniques for reliability prediction of automotive manufacturing," Applied Soft Computing, vol. 98, p. 106738, Jan. 2021, doi: 10.1016/j.asoc.2020.106738.

[3]P. Rani and G. S. Mahapatra, "A novel approach of NPSO on dynamic weighted NHPP model for software reliability analysis with additional fault introduction parameter," Heliyon, vol. 5, no. 7, p. e02082, Jul. 2019, doi: 10.1016/j.heliyon.2019.e02082.

[4]L. V. Utkin and F. P. A. Coolen, "A robust weighted SVR-based software reliability growth model," Reliability Engineering & System Safety, vol. 176, pp. 93–101, Aug. 2018, doi: 10.1016/j.ress.2018.04.007.

[5]B. Yang and M. Xie, "A study of operational and testing reliability in software reliability analysis," Reliability Engineering & System Safety, vol. 70, no. 3, pp. 323–329, Dec. 2000, doi: 10.1016/S0951-8320(00)00069-7.

[6]K.-C. Chiu, Y.-S. Huang, and T.-Z. Lee, "A study of software reliability growth from the perspective of learning effects," Reliability Engineering & System Safety, vol. 93, no. 10, pp. 1410–1421, Oct. 2008, doi: 10.1016/j.ress.2007.11.004.

[7]S. L. Ho, M. Xie, and T. N. Goh, "A study of the connectionist models for software reliability prediction," Computers & Mathematics with Applications, vol. 46, no. 7, pp. 1037–1045, Oct. 2003, doi: 10.1016/S0898-1221(03)90117-9.

[8]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, pp. 27–42, Sep. 2018, doi: 10.1016/j.cl.2017.12.002.

[9]I. Lakshmanan and S. Ramasamy, "An Artificial Neural-Network Approach to Software Reliability Growth Modeling," Procedia Computer Science, vol. 57, pp. 695–702, Jan. 2015, doi: 10.1016/j.procs.2015.07.450.

[10]J.-H. Lo and C.-Y. Huang, "An integration of fault detection and correction processes in software reliability analysis," Journal of Systems and Software, vol. 79, no. 9, pp. 1312–1323, Sep. 2006, doi: 10.1016/j.jss.2005.12.006.

[11]J. Wang, Z. Wu, Y. Shu, and Z. Zhang, "An optimized method for software reliability model based on nonhomogeneous Poisson process," Applied Mathematical Modelling, vol. 40, no. 13, pp. 6324–6339, Jul. 2016, doi: 10.1016/j.apm.2016.01.016.

[12]V. Ivanov, A. Reznik, and G. Succi, "Comparing the reliability of software systems: A case study on mobile operating systems," Information Sciences, vol. 423, pp. 398–411, Jan. 2018, doi: 10.1016/j.ins.2017.08.079.

[13]S. Kaliraj, D. Vivek, M. Kannan, K. Karthick, and M. Dhasny Lydia, "Critical review on software reliability models: Importance and application of reliability analysis in software development," Materials Today: Proceedings, Nov. 2020, doi: 10.1016/j.matpr.2020.10.076.

[14]S. Inoue and S. Yamada, "Discrete software reliability assessment with discretized NHPP models," Computers & Mathematics with Applications, vol. 51, no. 2, pp. 161–170, Jan. 2006, doi: 10.1016/j.camwa.2005.11.022.

[15]K.-Y. Cai, D.-B. Hu, C.-G. Bai, H. Hu, and T. Jing, "Does software reliability growth behavior follow a non-homogeneous Poisson process," Information and Software Technology, vol. 50, no. 12, pp. 1232–1247, Nov. 2008, doi: 10.1016/j.infsof.2007.12.001.

[16]S. Sinha, N. Kumar Goyal, and R. Mall, "Early prediction of reliability and availability of combined hardware-software systems based on functional failures," Journal of Systems Architecture, vol. 92, pp. 23–38, Jan. 2019, doi: 10.1016/j.sysarc.2018.10.007.

[17]C.-J. Hsu, C.-Y. Huang, and J.-R. Chang, "Enhancing software reliability modeling and prediction through the introduction of time-variable fault reduction factor," Applied Mathematical Modelling, vol. 35, no. 1, pp. 506–521, Jan. 2011, doi: 10.1016/j.apm.2010.07.017.

[18]C.-T. Lin, "Enhancing the accuracy of software reliability prediction through quantifying the effect of test phase transitions," Applied Mathematics and Computation, vol. 219, no. 5, pp. 2478–2492, Nov. 2012, doi: 10.1016/j.amc.2012.08.083.

[19]P. K. Kapur, D. N. Goswami, A. Bardhan, and O. Singh, "Flexible software reliability growth model with testing effort dependent learning process," Applied Mathematical Modelling, vol. 32, no. 7, pp. 1298–1307, Jul. 2008, doi: 10.1016/j.apm.2007.04.002.

[20]K. Ohishi, H. Okamura, and T. Dohi, "Gompertz software reliability model: Estimation algorithm and empirical validation," Journal of Systems and Software, vol. 82, no. 3, pp. 535–543, Mar. 2009, doi: 10.1016/j.jss.2008.11.840.

[21]A. Tarinejad, H. Izadkhah, M. M. Ardakani, and K. Mirzaie, "Metrics for assessing reliability of self-healing software systems," Computers & Electrical Engineering, vol. 90, p. 106952, Mar. 2021, doi: 10.1016/j.compeleceng.2020.106952.

[22]A. S. de Bustamante and B. S. de Bustamante, "Multinomial-exponential reliability function: a software reliability model," Reliability Engineering & System Safety, vol. 79, no. 3, pp. 281–288, Mar. 2003, doi: 10.1016/S0951-8320(02)00160-6.

[23]Y.-S. Su and C.-Y. Huang, "Neural-network-based approaches for software reliability estimation using dynamic weighted combinational models," Journal of Systems and Software, vol. 80, no. 4, pp. 606–615, Apr. 2007, doi: 10.1016/j.jss.2006.06.017.

[24]P. Roy, G. S. Mahapatra, and K. N. Dey, "Neuro-genetic approach on logistic model based software reliability prediction," Expert Systems with Applications, vol. 42, no. 10, pp. 4709–4718, Jun. 2015, doi: 10.1016/j.eswa.2015.01.043.

[25]N. Ravishanker, Z. Liu, and B. K. Ray, "NHPP models with Markov switching for software reliability," Computational Statistics & Data Analysis, vol. 52, no. 8, pp. 3988–3999, Apr. 2008, doi: 10.1016/j.csda.2008.01.010.

[26]H. Pham and X. Zhang, "NHPP software reliability and cost models with testing coverage," European Journal of Operational Research, vol. 145, no. 2, pp. 443–454, Mar. 2003, doi: 10.1016/S0377-2217(02)00181-9.

[27]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, pp. 68–85, Nov. 2017, doi: 10.1016/j.apm.2017.06.034.

[28]O. Yazdanbakhsh, S. Dick, I. Reay, and E. Mace, "On deterministic chaos in software reliability growth models," Applied Soft Computing, vol. 49, pp. 1256–1269, Dec. 2016, doi: 10.1016/j.asoc.2016.08.006.

[29]T. K. Nayak, S. Bose, and S. Kundu, "On inconsistency of estimators of parameters of non-homogeneous Poisson process models for software reliability," Statistics & Probability Letters, vol. 78, no. 14, pp. 2217–2221, Oct. 2008, doi: 10.1016/j.spl.2008.01.089.

[30]A. BahooToroody et al., "On reliability challenges of repairable systems using hierarchical bayesian inference and maximum likelihood estimation," Process Safety and Environmental Protection, vol. 135, pp. 157–165, Mar. 2020, doi: 10.1016/j.psep.2019.11.039.

[31]V. Nagaraju, C. Jayasinghe, and L. Fiondella, "Optimal test activity allocation for covariate software reliability and security models," Journal of Systems and Software, vol. 168, p. 110643, Oct. 2020, doi: 10.1016/j.jss.2020.110643.

[32]T. Yaghoobi, "Parameter optimization of software reliability models using improved differential evolution algorithm," Mathematics and Computers in Simulation, vol. 177, pp. 46–62, Nov. 2020, doi: 10.1016/j.matcom.2020.04.003.

[33]C.-Y. Huang, "Performance analysis of software reliability growth models with testing-effort and change-point," Journal of Systems and Software, vol. 76, no. 2, pp. 181–194, May 2005, doi: 10.1016/j.jss.2004.04.024.

[34]A. Chehade, Z. Shi, and V. Krivtsov, "Power–law nonhomogeneous Poisson process with a mixture of latent common shape parameters," Reliability Engineering & System Safety, vol. 203, p. 107097, Nov. 2020, doi: 10.1016/j.ress.2020.107097.

[35]J. Van Dyck and T. Verdonck, "Precision of power-law NHPP estimates for multiple systems with known failure rate scaling," Reliability Engineering & System Safety, vol. 126, pp. 143–152, Jun. 2014, doi: 10.1016/j.ress.2014.01.019.

[36]Y. Saito and T. Dohi, "Predicting software reliability via completely monotone nonparametric estimator with grouped data," Journal of Systems and Software, vol. 117, pp. 296–306, Jul. 2016, doi: 10.1016/j.jss.2016.03.047.

[37]J. Zheng, "Predicting software reliability with neural network ensembles," Expert Systems with Applications, vol. 36, no. 2, Part 1, pp. 2116–2122, Mar. 2009, doi: 10.1016/j.eswa.2007.12.029.

[38]X. Li, Y. F. Li, M. Xie, and S. H. Ng, "Reliability analysis and optimal version-updating for open source software," Information and Software Technology, vol. 53, no. 9, pp. 929–936, Sep. 2011, doi: 10.1016/j.infsof.2011.04.005.

[39]B. Wu and L. Cui, "Reliability analysis of periodically inspected systems with competing risks under Markovian environments," Computers & Industrial Engineering, vol. 158, p. 107415, Aug. 2021, doi: 10.1016/j.cie.2021.107415.

[40]W. Peng, L. Shen, Y. Shen, and Q. Sun, "Reliability analysis of repairable systems with recurrent misuse-induced failures and normal-operation failures," Reliability Engineering & System Safety, vol. 171, pp. 87–98, Mar. 2018, doi: 10.1016/j.ress.2017.11.016.

[41]Z.-S. Ye, M. Xie, and L.-C. Tang, "Reliability evaluation of hard disk drive failures based on counting processes," Reliability Engineering & System Safety, vol. 109, pp. 110–118, Jan. 2013, doi: 10.1016/j.ress.2012.07.003.

[42]J.-E. Byun, H.-M. Noh, and J. Song, "Reliability growth analysis of k-out-of-N systems using matrix-based system reliability method," Reliability Engineering & System Safety, vol. 165, pp. 410–421, Sep. 2017, doi: 10.1016/j.ress.2017.05.001.

[43]E. Abuta and J. Tian, "Reliability over consecutive releases of a semiconductor Optical Endpoint Detection software system developed in a small company," Journal of Systems and Software, vol. 137, pp. 355–365, Mar. 2018, doi: 10.1016/j.jss.2017.12.006.

[44]P. Kumar, S. K. Singh, and S. Deo Choudhary, "Reliability prediction analysis of aspect-oriented application using soft computing techniques," Materials Today: Proceedings, vol. 45, pp. 2660–2665, Jan. 2021, doi: 10.1016/j.matpr.2020.11.518.

[45]Z. Hui and X. Liu, "Research on Software Reliability Growth Model Based on Gaussian New Distribution," Procedia Computer Science, vol. 166, pp. 73–77, Jan. 2020, doi: 10.1016/j.procs.2020.02.019.

[46]Q. Zhang, Q. Ma, M. Liu, K. Zhong, B. Xu, and L. Wu, "Research on the software reliability quantitative evaluation of nuclear power plant digital control system based on non-homogeneous poisson process model," Annals of Nuclear Energy, vol. 144, p. 107589, Sep. 2020, doi: 10.1016/j.anucene.2020.107589.

[47]M. Xie and G. Y. Hong, "Software release time determination based on unbounded NHPP model," Computers & Industrial Engineering, vol. 37, no. 1, pp. 165–168, Oct. 1999, doi: 10.1016/S0360-8352(99)00046-7.

[48]C.-Y. Huang and T.-Y. Hung, "Software reliability analysis and assessment using queueing models with multiple change-points," Computers & Mathematics with Applications, vol. 60, no. 7, pp. 2015–2030, Oct. 2010, doi: 10.1016/j.camwa.2010.07.039.

[49]H. Pham, "Software reliability and cost models: Perspectives, comparison, and practice," European Journal of Operational Research, vol. 149, no. 3, pp. 475–489, Sep. 2003, doi: 10.1016/S0377-2217(02)00498-8.

[50]A. L. Goel and K.-Z. Yang, "Software Reliability and Readiness Assessment Based on the Non-homogeneous Poisson Process," in Advances in Computers, vol. 45, M. V. Zelkowitz, Ed. Elsevier, 1997, pp. 197–267. doi: 10.1016/S0065-2458(08)60709-3.

[51]M. Kimura, S. Yamada, and S. Osaki, "Software reliability assessment for an exponential-S-shaped reliability growth phenomenon," Computers & Mathematics with Applications, vol. 24, no. 1, pp. 71–78, Jul. 1992, doi: 10.1016/0898-1221(92)90230-F.

[52]L. Li, "Software Reliability Growth Fault Correction Model Based on Machine Learning and Neural Network Algorithm," Microprocessors and Microsystems, vol. 80, p. 103538, Feb. 2021, doi: 10.1016/j.micpro.2020.103538.

[53]J. Zhao, H.-W. Liu, G. Cui, and X.-Z. Yang, "Software reliability growth model with change-point and environmental function," Journal of Systems and Software, vol. 79, no. 11, pp. 1578–1587, Nov. 2006, doi: 10.1016/j.jss.2006.02.030.

[54]F. M. Vallee and A. Ragot, "Reliability evaluation using NHPP models," in Proceedings. 1991 International Symposium on Software Reliability Engineering, May 1991, pp. 157–162. doi: 10.1109/ISSRE.1991.145372.