A Neural Network for Covariate Software Reliability Model Selection

Brendan Thibault[1], Caroline D'Amato[1], Vidhyashree Nagaraju[2], and Lance Fiondella[1]
[1]Electrical and Computer Engineering, University of Massachusetts Dartmouth, MA, USA
[2]Department of Computer Science, Stonehill College, MA, USA
Email: {bthibault,caroline.damato,lfiondella}@umassd.edu, vidhyashreenagaraju@gmail.com

***Abstract*—Many software reliability models exist, making it especially difficult for individuals new to software reliability to apply models effectively. Researchers have suggested a wide variety of statistical model selection techniques as well as frameworks that combine multiple selection techniques. However, these techniques often require a level of expertise and subjectivity that continue to make it difficult to apply software reliability models in practice. To overcome these limitations of past research, this paper presents a neural network approach to rank models based on their ability to accurately predict future defect discovery. Unbiased simulation techniques are used to generate training data as well as additional test data not used for training. Our results indicate that the model that predicts future defects most accurately was recommended by the neural network over 60% of the time and the first or second most accurate model was recommended by the neural network over 95% of the time. Moreover, in over 50% of cases where the second most accurate model was recommended, the predictive sum of squares error was no more than a factor of two times greater than the most accurate model, suggesting that neural networks may be a viable model selection approach for software reliability growth models.**

***Index Terms*—Software reliability, model selection, neural network, covariate model, predictive accuracy**

I. Introduction

Traditional software reliability growth models (SRGM) [1] model the defect discovery process and failure intensity as a function of time on test, while covariate SRGM [2], [3] employ time series of one or more test activities or metric. Despite the widely accepted use of statistical models to predict future events, countless publications have proposed progressively more complex parametric forms focused on characterizing the observed data well, but often fail to predict future failures accurately. Standard practice compares a proposed model to well known alternatives on one or more goodness of fit measures [4], [5], while fewer studies propose model selection frameworks [3], [6]. However, past studies stop short of assessing these approaches in a systematic manner on many data sets in order to unambiguously validate their suitability as a model selection technique. Since users, such as software engineers, often lack a background in statistics and therefore depend heavily on automated selection techniques to identify models capable of accurate predictions. Without robust automated model selection tools that can be trusted to recommend models on desired performance metrics, many decision makers will forgo the opportunity to incorporate software reliability growth models into the daily practices of their organizations.

Past papers applying goodness-of-fit measures to select software reliability growth models include Goel and Okumoto [7] who compared predicted values with the observed data and later generalized [8] the approach by comparing different models on a single dataset. Khoshgoftaar [9] applied the Akaike Information Criterion (AIC) [10] as a model assessment criteria and subsequently [11] applied to multiple models and simulated data sets as an alternative method to the log-likelihood function [12]. Knafl et al. [13] compared models based on predictive performance measures in which failures predicted in 100(1-s)% of the remaining time was compared to the observed data. Lyu and Nikora [14] implemented prequential-likelihood and AIC in the CASRE (Computer-Aided Software Reliability Estimation) tool, while Gaudoin et al. [15] evaluated several statistical measures, including the R-squared [16], [17], Kolmogorov-Smirnov, Cramer-von Mises, and Anderson-Darling tests. Other commonly employed measures [6] include bias [18], [19], mean square error (MSE) [20], mean absolute error (MAE) [17], accuracy of estimation (AE) [17], predictive ratio risk (PRR) [5], variance [18], [19], root mean square prediction error (RMSPE) [18], [19], sum of squared error (SSE) [21], the Theil statistic [22], Bayesian information criterion (BIC) [4], and predictive sum of squared error (PSSE) [4], [5].

Since different measures may recommend different model, researchers have also proposed model selection frameworks in which multiple goodness-of-fit measures are utilized. For example, Asada et al. [23] presented an algorithm to select a model based on weights assigned to each measure of the model. Garg et al. [24] proposed a matrix method with diagonal elements representing different ratings for models. Sharma et al. [25] proposed a distance based approach [26] that compares Euclidean Composite Distance between SRGM. Wu et al. [27] proposed a weighted model selection approach based on the prequential log-likelihood value. Similar studies [28], [29], [30] demonstrated the applicability of weighted selection approaches. Rana et al. [31] presented a framework to select a model with best observed and predicted MSE and Balanced Predictive Relative Error. Ullah [32] compared several measures to pick the best model among seven OSS projects, while iteratively applying models. Saidi et al. [33] presented a survey on model selection studies, considering

data trend, predictive bias, residual defects, and predictive performance-based. Gupta et al. [34], [35] proposed a multi-criteria decision making (MCDM) model based on a Weighted Euclidean Distance Based Approximation. Kumar et al. [36] developed a method to rank SRGM based on the fuzzy data envelopment analysis approach. More recently, Yaghoobi et al. [37] demonstrated two MCDM methods for model selection. Kumar et al. [38] proposed a hybrid entropy weight-based MCDM method and Technique for Order Preference by Similarity to an Ideal Solution approach, while Garg et al. [39] proposed a hybrid entropy-combinative distance-Based assessment. Despite these efforts, frameworks require substantial user expertise, which is not suitable for tool users who may prefer simpler techniques that recommend a model for a specific measure with high accuracy. Moreover, past studies have not considered model selection in the context of covariate software reliability models.

To support automated model selection, this paper presents a neural network-based approach to recommend one of multiple alternative hazard functions by optimizing a measure of goodness of fit. The approach is demonstrated by minimizing predictive sum of squares error, but the approach is general and can be extended to other measures of goodness of fit or frameworks. To train the neural network, unbiased simulation techniques were used to generate data sets for three alternative hazard functions. The network was then trained with a subset of the full data, but the prediction targets were computed with the full data, in order to enable model selection with only a subset of the data. Additional data not used for training, was generated to test the network. Our results indicate that the network successfully recommended the best hazard function over 60% of the time and one of the two best hazard functions over 95% percent of the time. Since the second most accurate hazard function may also predict well, we conducted an analysis of the error experienced when the second most accurate model is recommended by the network and found that such misclassification lead to PSSE values of no more than two times that of the best performing hazard function on 50% of these occasions. Thus, neural networks are a potentially viable model selection approach for individuals who lack statistical knowledge but desire to apply software reliability growth models in their work.

The remainder of the paper is organized as follows: Section II provides an overview of covariate software reliability models and goodness of fit measures employed, while Section III explains how neural networks are applied to perform model selection. Section IV illustrates the approach through a series of examples. Section V offers conclusions and directions for future research.

II. Covariate Software Detection Model Based on Discrete Cox Proportional Hazard Model

The discrete Cox proportional hazard NHPP SRGM [3] correlates m covariates to the number of events in each of n intervals. In the context of software defect discovery, these covariates can be (i) distinct activities to discover defects or (ii) metrics associated with defect discovery such as the number of test cases executed or code coverage attained. The matrix $\mathbf{x}_{n\times m}$ quantifies the amount of effort dedicated to each activity in each interval or value of associated metrics in each interval. For example, $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{im})$ denotes the amount of each activity $1 \leq j \leq m$ performed in interval i.

The mean value function predicts the number of defects discovered through the n^{th} interval given covariates $\mathbf{x}$

$$m(\mathbf{x}) = \omega \sum_{i=1}^{n} p_{i,\mathbf{x}_i} \tag{1}$$

where $\omega > 0$ represents the number of defects that would be discovered with infinite testing and

$$p_{i,\mathbf{x}_i} = \Big(1 - (1 - h(i))^{g(\mathbf{x}_i;\boldsymbol{\beta})}\Big) \prod_{k=1}^{i-1} (1 - h(k))^{g(\mathbf{x}_k;\boldsymbol{\beta})} \tag{2}$$

is the probability that a defect is discovered in interval i, given that it was not discovered in the first $(i-1)$ intervals under the assumption of independent intervals, $h(\cdot)$ is the baseline hazard function, and $\boldsymbol{\beta}$ is the vector of m parameters contained within the Cox proportional hazards model

$$g(\mathbf{x}_i;\boldsymbol{\beta}) = \exp(\beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_m x_{im}) \tag{3}$$

A. Hazard functions

This section presents three examples of discrete hazard functions that can be incorporated in Equation (2) and were originally employed in the covariate software reliability model of Shibata et al. [2]. Five additional hazard functions have been taken from the survey by Bracquemond and Gaudoin [40] and implemented in the Covariate Software Failure and Reliability Assessment Tool (C-SFRAT) [41].

1) Geometric (GM):

$$h(b) = b \tag{4}$$

where $b \in (0, 1)$ is the probability of detecting a defect.

2) Negative binomial of order two (NB2):

$$h(i; b) = \frac{ib^2}{1 + b(i-1)} \tag{5}$$

where $b \in (0, 1)$ and 2 indicates the order.

3) Discrete Weibull of order two (DW2):

$$h(i; b) = 1 - b^{i^2-(i-1)^2} \tag{6}$$

where $b \in (0, 1)$ and 2 indicates the order.

B. Goodness of Fit Measures

This section summarizes the goodness of fit measure [4], [5] employed in this study to compare how well alternative models characterize a data set.

1) Predictive Sum of Squares Error (PSSE): PSSE compares the predictions of a model with data not used to perform model fitting.

$$PSSE = \sum_{i=n-k+1}^{n} (\widehat{m}(i) - N(t_i))^2 \tag{7}$$

where the maximum likelihood estimates of the model parameters are determined from the first $n-k$ intervals, $\widehat{m}(i)$ is the number of defects estimated by the fitted model, and $N(t_i)$ is the number of defects discovered by the ith interval.

III. Neural Networks

This section explains how unbiased simulated techniques are used to generate training data and mapped to a neural network to directly select a hazard function that will accurately predict the number of future defects discovered.

There are very few covariate data sets [2], [42] in the published literature. Therefore, we employ simulation techniques to generate training and testing data. To ensure randomness, a candidate hazard function is selected uniformly at random from Equations (4)-(6). Next, model parameters for the number of defects to be discovered (ω), covariate defect detection coefficients ($\boldsymbol{\beta}$), and hazard function parameters such as b as well as the test activity matrix $\mathbf{x}_{n\times m}$ are randomly generated and the nonhomogeneous Poisson process for the number of defects detected in each of n intervals ($\mathbf{y}$) simulated. Specifically, the random model parameters and test activity data are substituted into Equations (2) and (3) respectively and a Poisson random variable with mean $\omega \times p_{i,\mathbf{x}_i}$ simulated for each interval i to obtain the number of defects detected in each interval. Thus, unbiased statistical simulation techniques, enable the generation of an arbitrary number of data sets for covariate defect detection models possessing the hazard functions described in Section II-A. For this approach to be valid, it is necessary to assume that the variety of the covariate models possessing these hazard functions characterize real-world data sufficiently. Empirical analysis [43] suggests that this is indeed the case, since the mean value function and failure intensity function of covariate models tends to fit and predict defect discovery data more accurately than traditional NHPP models without covariates.

For each data set generated and each hazard function the code underlying the C-SFRAT (Covariate Software Failure and Reliability Assessment Tool) [43] was applied with 90% of the test activity matrix $\mathbf{x}_{n\times m}$ and $\mathbf{y}$ to estimate the model parameters. The remaining 10% of the data not used to fit the model was then used to compute the PSSE defined in Equation (7).

The neural network is composed of an input layer, a hidden layer, and an output layer. In the training stage, the inputs to the neural network are the test activity data $\mathbf{x}$ for each interval i. Thus, the input layer is composed of $(n \times m + 1)$ neurons, one for each test activity data and number of defects discovered per interval. This includes the 10% of test activity data not used to fit the model but not the number of defects detected, which implies that the test plan for the remainder of the test process is known.

The output layer possesses one neuron for each hazard function and the target values of the neurons in the output layer are set to the PSSE values computed with the model fit to 90% of the data, as described above. In other words, the input layer provides test activities allocated so far as well as those planned, but the PSSE values to be predicted by the neural network in the output layer lack information on the number of defects discovered in the last 10% of the test intervals. By minimizing the error between the PSSE and the predictions in the output layer, the neural network can predict which hazard function will exhibit the lowest PSSE. Thus, the output layer directly predicts the PSSE of each hazard function, enabling model selection without needing to directly rely on traditional statistical measures of goodness of fit that do not necessary recommend the model that predicts most accurately.

The activation of the jth neuron in the hidden layer ℓ is

$$a_j^{(\ell)} = \sum_{i=1}^{|\ell-1|} w_{ji}^{(\ell)} \xi_i^{(\ell-1)} + b_j^{(\ell)} \tag{8}$$

where $|\ell - 1|$ is the number of neurons in the previous layer, $w_{ji}^{(\ell)}$ the weight connecting the ith neuron in layer $\ell - 1$ to the jth neuron in layer $|\ell|$, $\xi_i^{(\ell-1)}$ is the input from the ith neuron in layer $\ell - 1$, and $b_j^{(\ell)}$ is of the bias of the jth neuron in layer ℓ. The output the jth neuron is computed with a ReLU (Rectified Linear Unit), possessing the form

$$z_j := \max(0, a_j) \tag{9}$$

In the testing stage, additional data sets are randomly generated and the test activities ($\mathbf{x}$) input to the neural network. The ranking of the hazard functions produced by the neural network are then compared to the traditional procedure of fitting 90% of the data to the model with each hazard function, computing PSSE. Under ideal circumstances, the model that best minimizes PSSE will be ranked most highly.

IV. Illustrations

This section assesses the neural network approach to select a hazard function that minimizes the predictive sum of squares error. The examples construct a CDF of the probability that the network selects the first, second, or third best hazard function, interprets the corresponding confusion matrix, and subsequently examines the consequences of using the second best hazard function for making predictions.

To demonstrate the neural network approach to select a hazard function for the covariate software defect detection model that accurately predicts PSSE, $\phi = 1000$ training data sets were generated from one of three hazard functions, including the GM, DW2, and NB2, thus requiring three output layer neurons. To promote robustness, the hazard function for each data set was selected uniformly at random. Prior to simulation a unique value of the b parameter was selected

from the ranges $b = (0.025, 0.25)$ for GM, $b = (0.025, 0.2)$ for NB, and $b = (0.95, 0.99)$ for DW2. These ranges were selected to avoid large deviations in the simulated data sets that would have made all models a poor fit to the data. Similar the range $\omega = (25, 125)$ was used for the number of defects and $\beta = (0.01, 0.05)$ for the covariate test effectiveness parameter.

Each simulation produced data for $n = 25$ intervals. Therefore, the number of neurons in the input layer was 50 (25 for the covariate (amount of test activity) and 25 for the number of defects detected). Although holding the number of intervals constant is necessary to ensure the data was compatible with the neural network, it is possible to normalize data sets with more or less intervals to preserve applicability. One hidden layer was included, containing 100 neurons and the network was fully connected, meaning that the output of each neuron in the input layer served as an input to each neuron in the hidden layer. The output of each neuron in the hidden layer serves as input to the three neuron in the output layer.

Figure 1 shows a histogram of the rankings produced by the neural network as well as the corresponding CDF.

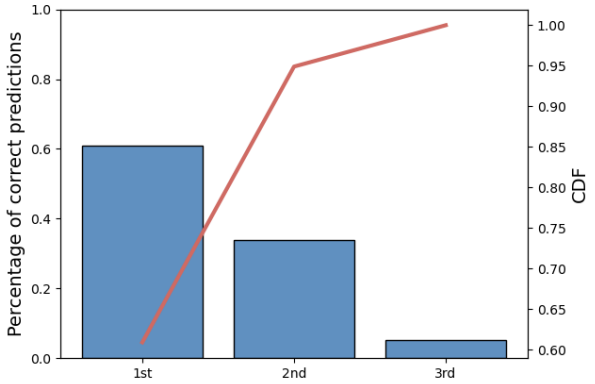

Fig. 1: Histogram and CDF of hazard function lowest PSSE rankings

Figure 1 indicates that the neural network successfully recommended the hazard function with the lowest PSSE over 60% of the time and that it recommended the first or second best hazard function over 95% of the time.

To more closely examine the behavior of the network, Table I shows the confusion matrix, where rows indicate the model that achieved the best PSSE and columns correspond to the hazard function that was recommended by the network.

TABLE I: Lowest PSSE confusion matrix

	GM	NB2	DW2
GM	329	95	142
NB2	23	18	15
DW2	116	0	262

Elements on the diagonal correspond to correct classifications, whereas off-diagonal elements indicate instances where the neural network did not successfully select the correct hazard function to minimize PSSE. Even though a third of the $\phi = 1000$ data sets use to test the network were simulated from each hazard function, DW2 rarely performed best, indicating that it may be less flexible. There were also several instances (11.6%) where the GM was recommended even though DW2 minimized PSSE and the DW2 selected even though GM performed best (14.2%). In these cases, it is possible that the variations in simulation made the simpler GM fit less well. Hence, it is possible that the more flexible DW2 would predict more accurately. Similarly, simulation of the DW may have produced trends that were sufficiently well characterized by simpler the GM hazard function.

To further elucidate the relatively large number of classifications on the off diagonal and assess if outputs recommending a second best model produced substantially poorer predictive performance, we computed the distribution of the ratio of errors by dividing the PSSE of the second best model by the PSSE of the preferred model for all instances when the second best hazard function was recommended. This analysis includes all instances in the middle column of Figure 1 and may be anywhere on the off-diagonal of TableI. The results of this analysis indicated that over half of the time the neural network recommended the second best hazard function the ration of the PSSE between the second and first best hazard functions was less than 2.0. In cases where the PSSE of the best hazard function is low, the PSSE of the second best hazard function will also be low, indicating that use of the second best hazard function may not significantly impact the model's predictive ability. Thus, use of a neural network to automatically select a model may be a viable alternative to other model selection techniques that do not guarantee the chosen model will produce accurate predictions.

V. Conclusions and Future Research

This paper presented a neural network based approach to select a hazard function for a covariate model. The approach was demonstrated on the predictive sum of squares error measure, but the approach is general and can be applied to other measures as well as combinations of measures. Unbiased simulation techniques were used to generate training data and additional data sets simulated to test the network. Our results indicated that the network recommended the best performing hazard function over 60% of the time and the first or second best hazard function over 95%. Moreover, in cases where the second best model was recommended by the network, the PSSE did not increase by more than a factor of 2.0 in 60% of these cases.

Future research will consider other more advanced forms of neural networks to improve the accuracy of the proposed hazard function selection technique.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant Number 1749635. Any opinions, findings, and conclusions or recommendations expressed

in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] W. Farr, *Handbook Of Software Reliability Engineering*. New York, NY: McGraw-Hill, 1996, chapter Software Reliability Modeling Survey, pp. 71–117.

[2] K. Shibata, K. Rinsaka, and T. Dohi, "Metrics-based software reliability models using non-homogeneous poisson processes," in *International Symposium on Software Reliability Engineering*, 2006, pp. 52–61.

[3] V. Nagaraju, C. Jayasinghe, and L. Fiondella, "Optimal test activity allocation for covariate software reliability and security models," *Journal of Systems and Software*, p. 110643, 05 2020.

[4] D. Kleinbaum, L. Kupper, A. Nizam, and K. Muller, *Applied Regression Analysis and Other Multivariable Methods*, 4th ed., ser. Applied Series. Belmont, CA: Duxbury Press, 2008.

[5] H. Pham and C. Deng, "Predictive-ratio risk criterion for selecting software reliability models," in *International Conference on Reliability and Quality in Design*, 2003, pp. 17–21.

[6] K. Sharma, R. Garg, C. Nagpal, and R. K. Garg, "Selection of optimal software reliability growth models using a distance based approach," *IEEE Transactions on Reliability*, vol. 59, no. 2, pp. 266–276, 2010.

[7] A. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, vol. 28, no. 3, pp. 206–211, 1979.

[8] A. Goel, "Software reliability models: Assumptions, limitations, and applicability," *IEEE Transactions on Software Engineering*, no. 12, pp. 1411–1423, 1985.

[9] T. Khoshgoftaar, "On model selection in software reliability," in *Symposium in Computational Statistics*, 1988, pp. 13–14.

[10] Y. Sakamoto, M. Ishiguro, and G. Kitagawa, "Akaike information criterion statistics," *Dordrecht, The Netherlands: D. Reidel*, vol. 81, no. 10.5555, p. 26853, 1986.

[11] T. Khoshgoftaar and T. Woodcock, "A simulation study of the performance of the Akaike information criterion for the selection of software reliability growth models," in *Annual South East Region ACM Conference*, 1989, pp. 419–423.

[12] ——, "Software reliability model selection: A case study," in *IEEE International Symposium on Software Reliability Engineering*, 1991, pp. 183–184.

[13] G. Knafl and J. Sacks, "Software reliability model selection," in *IEEE International Computer Software and Applications Conference*, 1991, pp. 597–598.

[14] M. Lyu and A. Nikora, "CASRE-A computer-aided software reliability estimation tool," in *International Workshop on Computer-Aided Software Engineering, Montreal, CA*, 1992, pp. 264–275.

[15] O. Gaudoin, B. Yang, and M. Xie, "A simple goodness-of-fit test for the power-law process, based on the Duane plot," *IEEE Transactions on Reliability*, vol. 52, no. 1, pp. 69–74, 2003.

[16] C. Stringfellow and A. Andrews, "An empirical method for selecting software reliability growth models," *Empirical Software Engineering*, vol. 7, no. 4, pp. 319–343, 2002.

[17] 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, 2008.

[18] C.-Y. Huang and S.-Y. Kuo, "Analysis of incorporating logistic testing-effort function into software reliability modeling," *IEEE Transactions on Reliability*, vol. 51, no. 3, pp. 261–270, 2002.

[19] K. Pillai and V. Nair, "A model for software development effort and cost estimation," *IEEE Transactions on Software Engineering*, vol. 23, no. 8, pp. 485–497, 1997.

[20] S. Hwang and H. Pham, "Quasi-renewal time-delay fault-removal consideration in software reliability modeling," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 1, pp. 200–209, 2008.

[21] X. Zhang, X. Teng, and H. Pham, "Considering fault removal efficiency in software reliability assessment," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 33, no. 1, pp. 114–120, 2003.

[22] P. Li, J. Herbsleb, and M. Shaw, "Forecasting field defect rates using a combined time-based and metrics-based approach: A case study of OpenBSD," in *IEEE International Symposium on Software Reliability Engineering*, 2005, pp. 10–pp.

[23] C. Asad, M. Ullah, and M. Rehman, "An approach for software reliability model selection," in *IEEE International Computer Software and Applications Conference*, 2004, pp. 534–539.

[24] R. Garg, K. Sharma, R. Kumar, and R. Garg, "Performance analysis of software reliability models using matrix method," *World Academy of Science, Engineering and Technology*, vol. 71, pp. 31–38, 2010.

[25] K. Sharma, R. Garg, C. Nagpal, and R. K. Garg, "Selection of optimal software reliability growth models using a distance based approach," *IEEE Transactions on Reliability*, vol. 59, no. 2, pp. 266–276, 2010.

[26] L. Ong, M. Isa, D. Jawawi, and H. HALIM, "Improving software reliability growth model selection ranking using particle swarm optimization," *Journal of Theoretical & Applied Information Technology*, vol. 95, no. 1, 2017.

[27] W. Wu, K. Han, C. He, and S. Wu, "A dynamically-weighted software reliability combination model," in *IEEE International Conference on Quality, Reliability, Risk, Maintenance, and Safety Engineering*, 2012, pp. 148–151.

[28] M. Anjum, M. Haque, and N. Ahmad, "Analysis and ranking of software reliability models based on weighted criteria value," *International Journal of Information Technology and Computer Science*, vol. 2, no. 1, pp. 1–14, 2013.

[29] N. Hung-Cuong, H. Quyet-Thang, and L. Hai-Trieu, "Different ranking of NHPP software reliability growth models with generalised measure and predictability," *International Journal of Applied Information Systems*, vol. 7, no. 11, pp. 1–6, 2014.

[30] B. Khalid and K. Sharma, "Ranking of software reliability growth models using bacterial foraging optimization algorithm," in *IEEE International Conference on Computing for Sustainable Global Development*, 2015, pp. 1643–1648.

[31] R. Rana, M. Staron, C. Berger, J. Hansson, M. Nilsson, F. Törner, W. Meding, and C. Höglund, "Selecting software reliability growth models and improving their predictive accuracy using historical projects data," *Journal of Systems and Software*, vol. 98, pp. 59–78, 2014.

[32] N. Ullah, "A method for predicting open source software residual defects," *Software Quality Journal*, vol. 23, no. 1, pp. 55–76, 2015.

[33] M. M. Saidi, M. Isa, D. Jawawi, and L. Ong, "A survey of software reliability growth model selection methods for improving reliability prediction accuracy," in *IEEE Malaysian Software Engineering Conference*, 2015, pp. 200–205.

[34] A. Gupta, N. Gupta, and R. Garg, "Implementing weighted entropy-distance based approach for the selection of software reliability growth models," *International Journal of Computer Applications in Technology*, vol. 57, no. 3, pp. 255–266, 2018.

[35] A. Gupta, N. Gupta, R. Garg, and R. Kumar, "Evaluation, selection and ranking of software reliability growth models using multi criteria decision making approach," in *IEEE International Conference on Computing Communication and Automation*, 2018, pp. 1–8.

[36] V. Kumar, V. Singh, A. Garg, and G. Kumar, "Selection of optimal software reliability growth models: a fuzzy DEA ranking approach," in *Quality, IT and Business Operations*. Springer, 2018, pp. 347–357.

[37] T. Yaghoobi, "Selection of optimal software reliability growth model using a diversity index," *Soft Computing*, vol. 25, no. 7, pp. 5339–5353, 2021.

[38] V. Kumar, P. Saxena, and H. Garg, "Selection of optimal software reliability growth models using an integrated entropy–technique for order preference by similarity to an ideal solution (TOPSIS) approach," *Mathematical Methods in the Applied Sciences*.

[39] R. Garg, S. Raheja, and R. Garg, "Decision support system for optimal selection of software reliability growth models using a hybrid approach," *IEEE Transactions on Reliability*, 2021.

[40] C. Bracquemond and O. Gaudoin, "A survey on discrete lifetime distributions," *International Journal of Reliability, Quality and Safety Engineering*, vol. 10, no. 1, pp. 69–98, 2003.

[41] J. Aubertine, V. Nagaraju, and L. Fiondella, "The covariate software failure and reliability assessment tool (C-SFRAT), miami, fl," in *In Proc. of International Conference on Reliability and Quality in Design*, 2021.

[42] J. Sorrentino, P. Silva, G. Baye, G. Kul, and L. Fiondella, "Covariate software vulnerability discovery model to support cybersecurity test & evaluation (practical experience report)," in *IEEE International Symposium on Software Reliability Engineering*, 2022, pp. 157–168.

[43] J. Aubertine, K. Chen, V. Nagaraju, and L. Fiondella, "A covariate software tool to guide test activity allocation," *SoftwareX*, vol. 17, p. 100909, 2022.