JOURNAL OF INFORMATION SCIENCE AND ENGINEERING XX, XXXX-XXXX (2024)
DOI: XX.XXX.XXXX.XX

Modeling software reliability growth with learning and fatigue effects*

UMASHANKAR SAMAL
Department of Mathematics
GLA University
Mathura, India
E-mail: umashankar.samal249@gmail.com

Software reliability is a fundamental pillar for ensuring the quality and dependability of software systems. Traditionally, software errors were commonly associated with coding mistakes. However, recent insights have shed light on the fact that human error is not static; rather, it is influenced by dynamic elements such as the processes of learning and the impact of fatigue. This study presents an approach that factors in the fatigue experienced by software testers during the debugging process, resulting in more reasonable software reliability growth models (SRGMs). By integrating S-shaped learning curves alongside an exponential function to model tester fatigue, the proposed models offer a more realistic representation of reliability growth over time. The models' quality, predictive capabilities, and accuracy are assessed to other existing models using three established fit criteria and two commonly used datasets. By including the fatigue component into SRGMs, a more comprehensive representation of software reliability dynamics is achieved. The inclusion of this feature enhances the precision and prediction capabilities of the models, therefore facilitating a more realistic evaluation of the long-term reliability of the software system.

Keywords: software reliability, mean value function, fatigue, learning

1. Introduction

In recent years, the advancement of technology has seamlessly integrated software systems into every aspect of human life. It's challenging to fathom even a single day without relying on software. This omnipresence of software has revolutionized daily routines, reshaping lifestyles. Its influence spans across individuals and organizations alike.

Software's widespread influence is clear in the variety of applications it powers: from smart devices that streamline everyday tasks to traffic light systems that manage road safety. With the growing demand for software, developers face increasing pressure to deliver highly reliable products. From initial concept to market launch, ensuring software reliability is crucial in a competitive environment. Developers must rigorously evaluate and test their products' reliability, as this quality is essential before presenting the software to a discerning and competitive market.

Received ** **, 2024; revised ****, ****; accepted ****.
Communicated by the editor.

In software development, reliability refers to the software's ability to function seamlessly within a specified time frame and under defined conditions [1, 2, 3]. Assessing software reliability involves a mathematical analysis that examines how factors like the number of detected faults or successfully fixed issues align with the testing duration. Researches like [4, 5, 6, 7] have explored this area, leading to the development of software reliability growth models (SRGMs). Identifying the "best" SRGM is a challenging task due to the numerous factors involved in these models.

Debugging is a fundamental part of the software development process, focused on identifying and fixing errors that can compromise software reliability. Although achieving perfection is nearly impossible, the goal is to minimize bugs and produce a high-quality product. Debugging is rarely simple, as it often involves navigating the complexity of modern software systems. Effective debugging relies on a careful balance of various factors, one of which is the learning curve of software testers. This refers to how quickly testers familiarize themselves with the software's code, logic, and behavior. As their understanding deepens over time, testers become more adept at handling the complexities of debugging.

In this complex environment, one significant challenge is the subtle yet impactful role of fatigue. Extensive research has shown that fatigue has a strong influence on cognitive performance [8, 9]. Studies indicate that prolonged work without breaks can lead to diminished focus, as fatigue causes a gradual decline in attention. This happens due to reduced dopamine levels in the brain, a chemical critical for maintaining concentration. This decline follows an exponential decay pattern over time, continuing until a critical threshold is reached [7].

Debugging requires intense focus and concentration, as developers must meticulously review code, spot patterns, and find solutions. Fatigue can severely impact this focus, increasing the risk of missing critical details or making mistakes. The debugging process can be lengthy and challenging, particularly when dealing with complex issues. The relationship between learning and fatigue is evident in this context. Novice developers or those unfamiliar with the codebase often experience greater fatigue as they invest more effort in understanding the code and finding issues. On the other hand, fatigue can hinder the learning process, making it harder for developers to grasp new concepts or gain deeper insights into the software. This interaction between learning and fatigue highlights the challenges developers face in achieving effective debugging and ensuring reliable software.

Based on these insights, this research explores a new dimension of debugging: the combined effects of learning and tester fatigue. It introduces two SRGMs that incorporate both the learning rate and the impact of fatigue on testers. The study then performs a thorough comparative analysis, evaluating these new models against existing SRGMs to assess their effectiveness.

The paper is organized as follows: Section 2 provides a detailed review of the existing literature. Section 3 outlines the fundamental definitions necessary for understanding the study. Section 4 discusses the research process, explaining the assumptions and formulation of the mathematical model. In Section 5, an overview of the datasets, models, and criteria used for comparison is presented. The study's findings are discussed in Section 6. Finally, Section 7 wraps up with a summary of key findings and suggests directions for future research.

2. Literature review

Historically, SRGMs have primarily been based on non-homogeneous Poisson process (NHPP). Additionally, some researchers have investigated alternative approaches, such as Bayesian and Markov models, as well as machine learning techniques [10, 11, 12, 13]. The domain of reliability modeling is broad, featuring a wide array of models based on varying assumptions and techniques. For instance, Goel and Okumoto [4] introduced the GO model, a stochastic approach for reliability evaluation based on NHPP, which uses an exponential curve to represent the failure occurrence pattern.

Several researchers have explored the learning effect of software testers when calculating software reliability. As testers gain experience, their ability to detect and resolve faults improves over time, which can significantly influence the accuracy of reliability models. This learning effect often leads to higher fault detection rates in the later stages of testing. Ahmad et al. [3] proposed a model that integrates the exponentiated Weibull testing-effort function into inflection S-shaped SRGM based on NHPP. Roy et al. [14] introduced an S-shaped SRGM which accounts for imperfect debugging with an exponentially increasing fault content function and an S-shaped fault detection rate. Zhang et al. [15] proposed a software reliability model that integrates a constant fault removal efficiency (FRE) and fault introduction rate. The model reflects the learning process of software developers through an inflection S-shaped curve. Samal and Kumar [16] introduced an SRGM that accounts for the removal efficiency of testers, while also incorporating the learning effect through an S-shaped function. Iqbal et al. [17] introduced an SRGM that incorporates two types of learning effects: autonomous learning and acquired learning. Al-Turk and Al-Mutairi [18] proposed a model that incorporates the one-parameter Lindley distribution, integrating learning effects. Wang and Zhang [19] proposed a model that accounts for the continuous learning required during the open-source software testing process due to its iterative development. Chiu [2] proposed SRGMs incorporating multiple change-points in the software testing and debugging process, considering time-dependent learning effects. Yaghoobi and Leung [7] examined the influence of learning and fatigue on software testing errors. They introduced two SRGMs: one utilizing the hyperbolic tangent function to capture the effects of learning, and the other applying an exponential function to represent fatigue during the testing process.

2.1 Contributions

From the aforementioned discourse, it's evident that while the learning effect has been acknowledged in numerous studies by S-shaped functions, these endeavors have not comprehensively encompassed all potential factors related to human resources, such as fundamental abilities, learning efficiency, negligence by testing staff, and fatigue. In this context, the present study aims to make a distinctive contribution by addressing these previously overlooked dimensions. Specifically, an exponential curve is introduced to encapsulate the influence of fatigue while incorporating two S-shaped learning curves to accurately model the learning process of testers.

3. Preliminaries

i) **NHPP:** A computational process $N(t)$ is categorized as NHPP with an intensity function $\lambda(t)$ when its behavior can be described by a Poisson distribution with a mean value function $m(t)$, applicable for $t \geq 0$. Mathematically,

$$\Pr\{N(t) = k\} = \frac{[m(t)]^k}{k!} e^{-m(t)},\ k = 0, 1, 2, \cdots \tag{1}$$

$$\text{where } m(t) = E[N(t)].$$

ii) **Software reliability:** $R(x/t)$ is the likelihood that no software fault will be found between the times $(t, t+x)$, assuming that the prior fault occurred at time t, where $t \geq 0$, $x > 0$. Mathematically,

$$R(x/t) = e^{-[m(t+x)-m(t)]}. \tag{2}$$

4. Model development

4.1 Notations

Table 1 presents the symbols and abbreviations used throughout the study.

Table 1. Notations

$m(t)$	Projected number of software faults resolved by time t
N	Total number of faults in the system
$f(t)$	Rate of fatigue accumulation
$b(t)$	Rate of fault identification
β	Parameter influencing the learning process
b	Initial fault identification rate
w	Parameter related with the fatigue function
s	Scaling coefficient for the fatigue function
m_0	Baseline value for fault rectification

4.2 Assumptions

The proposed model is based on the following key assumptions:

1. The software failure process follows NHPP, accounting for the time-varying nature of fault occurrences.

2. The fault detection process reflects a learning curve effect, where testers become more efficient as they gain experience over time.

3. Tester fatigue gradually impacts the fault detection rate, diminishing the efficiency of the testing process as fatigue builds up.

4. All identified faults are assumed to be completely resolved upon detection, with no partial fixes or residual issues.

5. An initial testing phase, often a pre-testing analysis, is carried out to correct basic issues, improving the effectiveness of the formal testing that follows.

4.3 Formulation

Taking into account assumptions 1, 2, 3, and 4, an SRGM that incorporates the NHPP framework can be formulated as follows [7]:

$$\frac{dm(t)}{dt} = (b(t) - f(t))(N - m(t)). \tag{3}$$

Assumption 2 suggests that the fault detection process follows a learning curve, where testers become more proficient as they gain experience over time. To represent this behavior, two different functions are introduced. The first function, $b_1(t) = b/\left(1+\beta e^{-bt}\right)$, models an initial rapid improvement in detection efficiency as testing progresses. This captures the early learning phase, where testers quickly enhance their skills, leading to a sharp rise in fault detection rates. Over time, the curve levels off, indicating that efficiency gains begin to slow down as testers reach their maximum potential. The second function, $b_2(t) = b^2 t/(1+bt)$, shows a different trend. In the early stages, the increase in efficiency is more gradual compared to the first function. However, as time goes on, the learning process accelerates, resulting in a more significant improvement in detection rates later in the testing phase. This function reflects a scenario where testers take longer to gain momentum but eventually exhibit a faster rate of efficiency growth. By incorporating these two learning functions, the proposed model effectively captures different learning behaviors—one where testers adapt rapidly and then stabilize, and another where learning is slower initially but accelerates over time.

Assumption 3 highlights the influence of tester fatigue as a critical factor affecting the fault detection rate, ultimately reducing the overall effectiveness of the testing process. To address this, a fatigue factor is incorporated, modeled by the function $f(t) = se^{wt}$. This function captures the progressively growing fatigue over time, with its exponential form reflecting the increasing impact that fatigue has on testers as the testing period extends. As fatigue builds up, it gradually diminishes the testers' ability to maintain high fault detection efficiency. Here, s represents the scaling coefficient, which determines the initial intensity of fatigue when the testing process begins. A higher s value indicates that testers experience fatigue more rapidly from the outset, while a lower s suggests a slower onset of fatigue. The parameter w governs the rate of fatigue accumulation over time. A larger w value implies that fatigue increases more rapidly, leading to a faster decline in fault detection efficiency, whereas a smaller w value indicates a slower accumulation of fatigue, allowing testers to sustain their efficiency for a longer duration.

Now, Eq. (3) will have two parts, Eq. (4) and (5).

$$\frac{dm_1(t)}{dt} = \left(\frac{b}{1+\beta e^{-bt}} - se^{wt}\right)(N - m_1(t)). \tag{4}$$

$$\frac{dm_2(t)}{dt} = \left(\frac{b^2 t}{1+bt} - se^{wt}\right)(N - m_2(t)). \tag{5}$$

As stated in assumption 5, an initial pre-analysis phase is carried out to correct fundamental errors and minor glitches, improving the efficiency of the formal testing process that follows. To account for this, the model assumes that:

$$m_1(0) \text{ or } m_2(0) = m_0,\ m_0 > 0. \tag{6}$$

Now, solving Eq. (4) and (5) using Eq. (6),

$$m_1(t) = N - \frac{e^{\frac{s\,(e^{tw}-1)}{w}}\,(N-m_0)\,(\beta+1)}{\beta+e^{bt}}. \tag{7}$$

$$m_2(t) = N - e^{-\frac{s-se^{tw}+btw}{w}}\,(N-m_0)\,(bt+1). \tag{8}$$

5. Experimental design

5.1 Model evaluation

The potency of a model is assessed by examining its advantages, limitations, and accuracy. The performance analysis of the proposed models are compared with several existing SRGMs, as outlined in Table 2.

5.2 Dataset

To evaluate the prformance of the proposed models, data from two versions of the Tandem computer project is used [16]. These versions are labeled release-1 and release-2, with their data presented in Table 3 and Table 4, respectively. Over a 20-week period, release-1 had a total of 100 reported faults. The effort utilization for the first release was 10,000 units. In comparison, release-2, tested over 19 weeks, reported 120 faults. The effort utilization for the second release was 10,272 units. These datasets are used by various authors [5, 16, 7]. These datasets represent real-world fault detection and debugging efforts over time, capturing the cumulative number of faults detected during each testing week. These reflect a controlled system environment where systematic debugging and reliability assessment were conducted.

Although various authors have utilized these datasets, they are subject to certain limitations. The datasets primarily focus on fault detection trends over time without explicitly considering factors such as varying testing team sizes, adaptive debugging strategies, or dynamically changing system environments. However, to evaluate the performance of the proposed model, these datasets are chosen as they are well-established in software reliability research. It is important to emphasize that no single SRGM can be universally optimal for all scenarios, as different software systems exhibit unique fault detection and debugging characteristics.

5.3 Model fit evaluation criteria

To evaluate the accuracy of the proposed models, three standard metrics are used: mean squared error (MSE), mean absolute error (MAE), and the coefficient of determination (R^2). Various authors have used these metrics in their works [5, 11]. MSE measures

Table 2. SRGMs considered for comparison

Sl.No.	Model	$m(t)$	Reference
1.	GO model	$N\left(1-e^{-bt}\right)$	[4]
2.	DSS model	$N(1-(1+bt)e^{-bt})$	[20]
3.	ISS model	$\frac{N\left(1-e^{-bt}\right)}{1+\beta e^{-bt}}$	[21]
4.	ETC model	$\frac{N}{\beta}\left(e^{\beta k\left(1-e^{-bt^{\phi}}\right)}-1\right)$	[22]
5.	TC model	$N\left[1-\left(\frac{\beta}{\beta+(at)^{b}}\right)^{a}\right]$	[23]
6.	WFRF model	$\frac{N}{p-\alpha}\left(1-e^{-b\beta(p-\alpha)t^{k}}\right)$	[24]
7.	SU model	$N\left(1-\frac{\beta}{\beta+\ln\left(\frac{a+e^{bt}}{1+a}\right)}\right)^{\alpha}$	[25]
8.	NFRE model	$\frac{N}{\varphi}\left[1-\frac{1+\beta}{(\beta+e^{bt})^{\varphi}}\right]$	[26]
9.	Proposed model-1 (P1)	$N-\frac{\frac{s\,(\mathrm{e}^{tw}-1)}{w}\,(N-m_0)\,(\beta+1)}{\beta+\mathrm{e}^{bt}}$	Equation 7
10.	Proposed model-2 (P2)	$N-\mathrm{e}^{-\frac{s-s\mathrm{e}^{tw}+btw}{w}}\,(N-m_0)\,(bt+1)$	Equation 8

Table 3. Release-1 dataset

week	C.F.	week	C.F.	week	C.F.
1	16	8	58	15	96
2	24	9	69	16	98
3	27	10	75	17	99
4	33	11	81	18	100
5	41	12	86	19	100
6	49	13	90	20	100
7	54	14	93	-	-

Table 4. Release-2 dataset

week	C.F.	week	C.F.	week	C.F.
1	13	8	75	15	112
2	18	9	84	16	114
3	26	10	89	17	117
4	34	11	95	18	118
5	40	12	100	19	120
6	48	13	104	-	-
7	61	14	110	-	-

the average squared difference between predicted and actual values, where a lower MSE indicates better accuracy. MAE quantifies the average absolute error, with lower values signifying improved predictive performance. R^2 assesses how well the model explains the variance in the observed data, where higher values indicate a better fit.

6. Results and discussion

When evaluating the software reliability models for release-1, Table 5 clearly indicates that models P1 and P2 outperform the others. Model P1, in particular, achieves the lowest MSE at 1.7581, the smallest MAE at 0.7439, and the highest R^2 value of 0.9984. These metrics collectively demonstrate P1's superior accuracy in predicting cumulative failures. Model P2 also performs well, though slightly behind P1. It ranks second with an MSE of 5.0395 and an MAE of 1.5007, making it a strong alternative. P2's R^2 value of

0.9952, while slightly lower than P1's, still indicates excellent predictive capability. The evaluation extends beyond numerical metrics, as shown in Figure 1 and Figure 2. Figure 1 provides a visual comparison of the actual and predicted cumulative faults across all ten SRGMs for release-1, with P1 closely tracking the observed data. Similarly, Figure 2 presents a boxplot, where P1 shows the least variation, indicating that its predictions are consistently close to the actual outcomes. P2 follows as a strong second, with slightly more spread in its predictions.

For release-2, the evaluation of software reliability models reveals that P1 and P2 are superior to the other models, as shown in Table 6. Model P1 excels with the lowest MSE of 3.9338, the smallest MAE of 1.2596, and the highest R^2 value of 0.9979, demonstrating its exceptional accuracy in predicting cumulative failures. Model P2 also performs admirably, with the second-lowest MSE of 4.8543 and the second-smallest MAE of 1.4319. While its R^2 value of 0.9972 is slightly lower than that of P1, it still reflects strong predictive performance. Further insights are provided by Figures 3 and 4. Figure 3 shows that P1's predictions closely match the actual cumulative failures, while Figure 4 highlights that P1 exhibits minimal spread in the boxplot, indicating consistent accuracy. P2 also demonstrates strong performance, with only a slight increase in spread compared to P1, affirming that both models are highly effective for release-2.

Table 5. Estimation of parameters for release-1

S.N.	Model	Parameters	MSE	MAE	R^2
1.	GO	$b = 0.0832$, $N = 133.2011$	16.2257	3.1508	0.9858
2.	DSS	$b = 0.2651$, $N = 103.9841$	28.0626	3.1679	0.9831
3.	ISS	$b = 0.1721$, $\beta = 1.3615$, $N = 109.8287$	14.9372	2.9110	0.9919
4.	ETC	$b = 0.2281$, $\beta = 3.6248$, $\phi = 0.8573$, $k = 0.5308$, $N = 74.5108$	17.8475	2.8219	0.9895
5.	TC	$b = 1.1102$, $a = 0.0912$, $\alpha = 39.1723$, $\beta = 36.1021$, $N = 119.8553$	14.6305	2.7226	0.9873
6.	WFRF	$b = 0.0562$, $\alpha = 1.5773$, $\beta = 1.7843$, $p = 2.3408$, $k = 1.1093$, $N = 90.5226$	15.5297	2.7015	0.9874
7.	SU	$a = 0.7161$, $b = 3.0966$, $\alpha = 3.0371$, $\beta = 6.4381$, $N = 141.3354$	44.1457	4.2472	0.9676
8.	NFRE	$b = 0.1916$, $\varphi = 1.0614$, $\beta = 1.3112$, $N = 110.6624$	16.3895	2.9112	0.9839
9.	P1	$b = 0.5353$, $\beta = -0.4117$, $s = 0.6398$, $w = -0.0752$, $m_0 = 0.0367$, $N = 102.2702$	1.7581	0.7439	0.9984
10.	P2	$b = 0.3266$, $s = 0.0813$, $w = 0.0529$, $m_0 = 22.6325$, $N = 122.3659$	5.0395	1.5007	0.9952

Table 6. Estimation of parameters for release-2

S.N.	Model	Parameters	MSE	MAE	R^2
1.	GO	$b = 0.0615$, $N = 182.9517$	24.5574	4.3561	0.9836
2.	DSS	$b = 0.2417$, $N = 131.3989$	21.5280	3.8544	0.9928
3.	ISS	$b = 0.2534$, $\beta = 3.7784$, $N = 124.4452$	6.7074	1.7643	0.9952
4.	ETC	$b = 0.1561$, $\beta = 3.9217$, $\phi = 1.0772$, $,k = 0.5260\, N = 74.8926$	17.4502	3.0831	0.9949
5.	TC	$a = 0.1025$, $b = 1.4552$, $\alpha = 41.7413$, $\beta = 40.4027$, $N = 128.5296$	22.4067	3.5998	0.9943
6.	WFRF	$b = 0.0489$, $\alpha = 1.5474$, $\beta = 1.2210$, $p = 2.2529$, $k = 1.4381$, $N = 91.2062$	15.0811	2.4154	0.9936
7.	SU	$a = 1.7804$, $b = 6.7509$, $\alpha = 4.3357$, $\beta = 14.4449$, $N = 202.0216$	37.6864	4.1728	0.9915
8.	NFRE	$b = 0.2541$, $\varphi = 1.0321$, $\beta = 3.7784$, $N = 124.1792$	10.3948	2.5884	0.9951
9.	P1	$b = 0.3395$, $\beta = -0.4426$, $s = 0.5173$, $w = -0.1473$, $m_0 = 6.0722$, $N = 121.5872$	3.9338	1.2596	0.9979
10.	P2	$b = 0.2875$, $s = 0.0805$, $w = -0.1231$, $m_0 = 19.2409$, $N = 125.1794$	4.8543	1.4319	0.9972

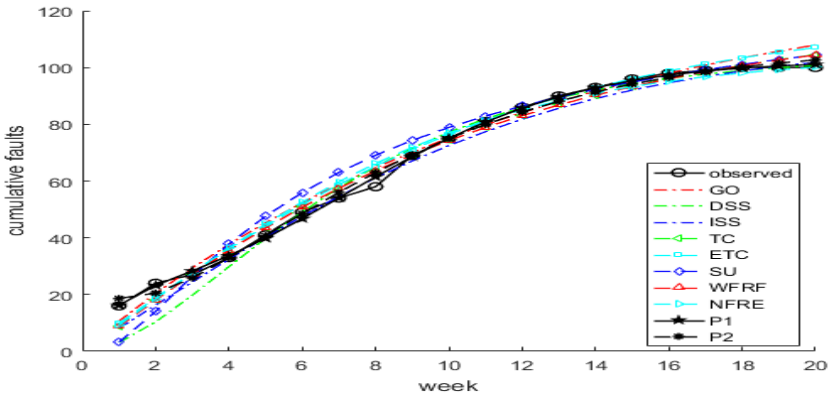

Fig. 1. Goodness of fit plot for release-1

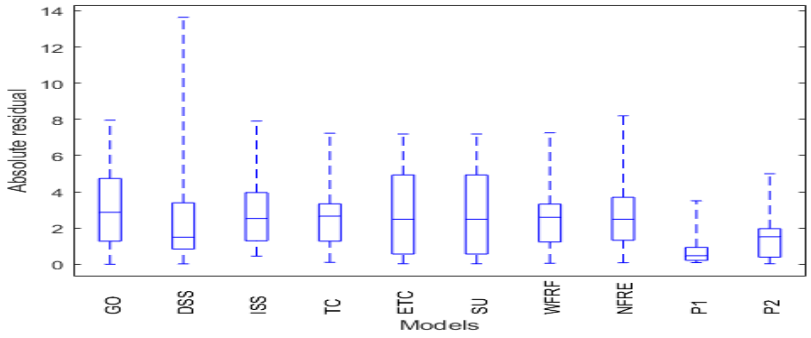

Fig. 2. Boxplot analysis of results for release-1

7. Conclusion and future scope

This paper introduces two NHPP based SRGMs that integrate the crucial factors of learning and fatigue. The efficacy of these models is substantiated through rigorous validation, employing real-life fault datasets obtained from Tandem computers. It is worth noting that the existing body of research on SRGMs rarely considers both learning and fatigue, making this study particularly innovative. Throughout the comprehensive evaluation of various models, the proposed ones consistently exhibit exceptional performance. Beyond advancing the understanding of software reliability dynamics, these models demonstrate practical utility in real-world contexts, thanks to their unique integration of learning and fatigue variables. Looking ahead, the future idea is to incorporate the concept of change points and environmental factors. This will enable to evaluate the effectiveness of testers within the framework of learning and fatigue, further enriching the practical relevance and utility of the proposed models.

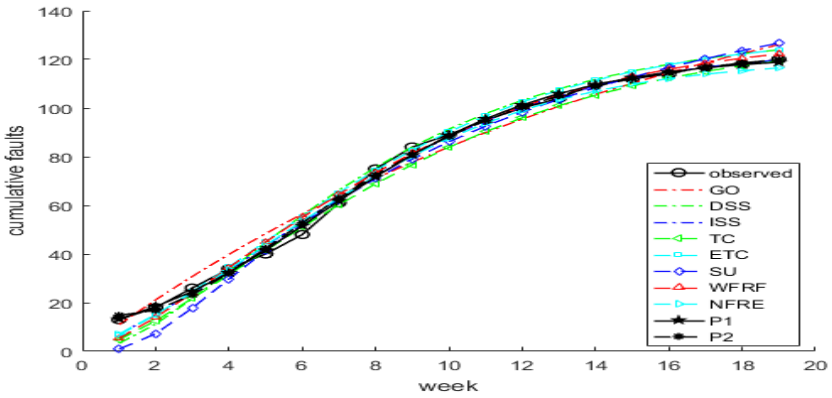

Fig. 3. Goodness of fit plot for release-2

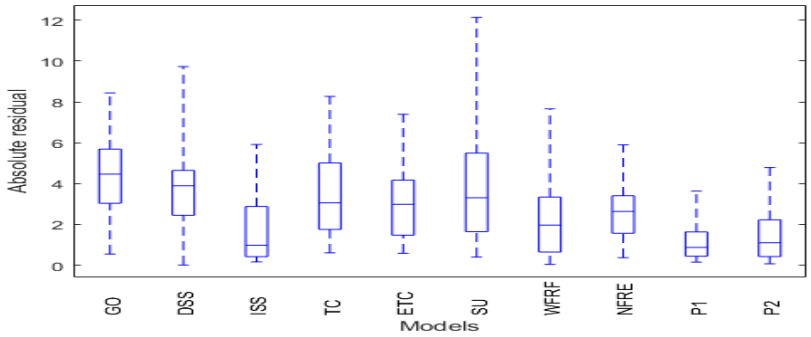

Fig. 4. Boxplot analysis of results for release-2

REFERENCES

1. U. Samal and A. Kumar, "Empowering software reliability: Leveraging efficient fault detection and removal efficiency," *Quality Engineering*, 2024, pp. 1–12.
2. K.-C. Chiu, "An exploration on debugging performance for software reliability growth models with learning effects and change-points," *Journal of Industrial and Production Engineering*, Vol. 32, no. 6, 2015, pp. 369–386.
3. N. Ahmad, M. G. Khan, and L. S. Rafi, "A study of testing-effort dependent inflection s-shaped software reliability growth models with imperfect debugging," *International Journal of Quality & Reliability Management*, Vol. 27, no. 1, 2010, pp. 89–110.
4. A. L. 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, 1979, pp. 206–211.

5. U. Samal and A. Kumar, "Redefining software reliability modeling: embracing fault-dependency, imperfect removal, and maximum fault considerations," *Quality Engineering*, 2023, pp. 1–10.
6. U. Samal and A. Kumar, "An enhanced software reliability growth model considering dynamic fault removal efficiency and residual error change rate," *Journal of Information Science and Engineering*, Vol. 40, no. 6, 2024, pp. 1321–1333.
7. T. Yaghoobi and M.-F. Leung, "Modeling software reliability with learning and fatigue," *Mathematics*, Vol. 11, no. 16, 2023, p. 3491.
8. S. Sarkar and C. Parnin, "Characterizing and predicting mental fatigue during programming tasks," in *2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*. IEEE, 2017, pp. 32–37.
9. U. Samal and A. Kumar, "Incorporating human dynamics into software reliability analysis: learning, fatigue, and efficiency considerations," *International Journal of System Assurance Engineering and Management*, 2024, pp. 1–10.
10. K. Y. Song, I. H. Chang, and H. Pham, "An nhpp software reliability model with s-shaped growth curve subject to random operating environments and optimal release time," *Applied Sciences*, Vol. 7, no. 12, 2017, p. 1304.
11. U. Samal and A. Kumar, "A neural network approach for software reliability prediction," *International Journal of Reliability, Quality and Safety Engineering*, 2024, p. 2450009.
12. X. Zhang and H. Pham, "A software cost model with warranty cost, error removal times and risk costs," *IIE transactions*, Vol. 30, no. 12, 1998, pp. 1135–1142.
13. U. Samal and A. Kumar, "Enhancing software reliability forecasting through a hybrid arima-ann model," *Arabian Journal for Science and Engineering*, Vol. 49, no. 5, 2024, pp. 7571–7584.
14. P. Roy, G. Mahapatra, and K. N. Dey, "An s-shaped software reliability model with imperfect debugging and improved testing learning process," *International Journal of Reliability and Safety*, Vol. 7, no. 4, 2013, pp. 372–387.
15. 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, 2003, pp. 114–120.
16. U. Samal and A. Kumar, "A software reliability model incorporating fault removal efficiency and it's release policy," *Computational Statistics*, 2023, pp. 1–19.
17. J. Iqbal, N. Ahmad, and S. Quadri, "A software reliability growth model with two types of learning," in *2013 International Conference on Machine Intelligence and Research Advancement*. IEEE, 2013, pp. 498–503.
18. L. I. Al-Turk and N. N. Al-Mutairi, "Enhancing reliability predictions by considering learning effects based on one-parameter lindley distribution," in *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*. IEEE, 2020, pp. 1–7.
19. J. Wang and C. Zhang, "An open-source software reliability model considering learning factors and stochastically introduced faults," *Applied Sciences*, Vol. 14, no. 2, 2024, p. 708.
20. S. Yamada, M. Ohba, and S. Osaki, "S-shaped software reliability growth models and their applications," *IEEE Transactions on Reliability*, Vol. 33, no. 4, 1984, pp. 289–292.

21. M. Ohba, "Inflection s-shaped software reliability growth model," in *Stochastic Models in Reliability Theory: Proceedings of a Symposium Held in Nagoya, Japan, April 23–24, 1984*. Springer, 1984, pp. 144–162.
22. S. Chatterjee and A. Shukla, "A unified approach of testing coverage-based software reliability growth modelling with fault detection probability, imperfect debugging, and change point," *Journal of Software: Evolution and Process*, Vol. 31, no. 3, 2019, p. e2150.
23. I. H. Chang, H. Pham, S. W. Lee, and K. Y. Song, "A testing-coverage software reliability model with the uncertainty of operating environments," *International Journal of Systems Science: Operations & Logistics*, Vol. 1, no. 4, 2014, pp. 220–227.
24. V. Verma, S. Anand, P. Kapur, and A. G. Aggarwal, "Unified framework to assess software reliability and determine optimal release time in presence of fault reduction factor, error generation and fault removal efficiency," *International Journal of System Assurance Engineering and Management*, Vol. 13, no. 5, 2022, pp. 2429–2441.
25. K. Y. Song, I. H. Chang, and H. Pham, "Nhpp software reliability model with inflection factor of the fault detection rate considering the uncertainty of software operating environments and predictive analysis," *Symmetry*, Vol. 11, no. 4, 2019, p. 521.
26. M. A. Haque and N. Ahmad, "A software reliability model using fault removal efficiency," *Journal of Reliability and Statistical Studies*, 2022, pp. 459–472.

Umashankar Samal graduated with a master's degree in Mathematics in 2019 from Sant Longowal Institute of Engineering & Technology, India. He has done his PhD from Atal Bihari Vajpayee-Indian Institute of Information Technology and Management, Gwalior, India. His research interests include safety, quality, and reliability engineering.