e-Informatica Software Engineering Journal
Volume 19, Issue 1, pages: 1–29
DOI: 10.37190/e-Inf250101
BibTeX

Bug Report Analytics for Software Reliability Assessment using Hybrid Swarm – Evolutionary Algorithm

Sangeeta*, Sitender*, Rachna Jain*, Ankita Bansal*
*Corresponding authors: sangeeta@msit.in, sitender@msit.in, rachnajain@bpitindia.com, ankita.bansal06@gmail.com

Article info

Dataset link: https://zenodo.org/records/13147825

Keywords:
software development process
bug report analysis
optimization
swarm evolutionary algorithms
software reliability

Submitted: 30 Sep. 2023
Revised: 12 Aug. 2024
Accepted: 17 Aug. 2024
Available online: 3 Oct. 2024

Abstract

Background: With the growing advances in the digital world, software development demands are increasing at an exponential rate. To ensure reliability of the software, high-performance tools for bug report analysis are needed.
Aim: This paper proposes a new "Iterative Software Reliability" model based on one of the most recent Software Development Life Cycle (SDLC) approach.
Method: The proposed iterative failure rate model assumes that new functionality enhancement occurs in each iteration of software development and accordingly design modification is made at each stage of software development. In terms of defects, testing effort, and added functionality, these changing needs in each iteration are reflected in the proposed model using iterative factors. The proposed model has been tested on twelve Eclipse and six JDT software failure datasets. Proposed model parameters have been estimated using a hybrid swarm – evolutionary algorithm.
Results: The proposed model has about 32% and 55% improved efficiency on Eclipse and JDT datasets, respectively, as compared to other models like Jelinski Moranda Model, Shick–Wolverton Model, Goel Okumotto Imperfect Model, etc.
Conclusion: In each analysis done, the proposed model is found to be reaching acceptable performance and could be applied on other software failure datasets for further validation.

1. Introduction

With the growing advances in cutting-edge innovations in digital world, software development demand from software industries is increasing exponentially. Due to limited budget, high demand and lack of time, there is a high probability of fault occurrence in the newly developed software, making reliability a concern. The reliability of software is considered as the most crucial quality attribute [1]. At one end, software has made today's life much more comfortable but at other end failures in unreliable software may cause life threatening issues to humans. So unpredictable software failures make life challenging due to the decreasing reliability of the newly developed systems. Software reliability could be only

measured using bug report analysis of the software. Using the latest tools and techniques at most care has been taken to develop reliable software, but practically a software developer cannot produce defect-free software [2]. Therefore, there should be a way to avoid software failures to further avoid severe losses to human life or any financial losses in industry. The prerequisite of software developers is to know whether developed software is reliable before they are dispatched to customers. In the competitive arcade of software development, software industries must ensure the reliability of their software to satisfy their customers and make an outlook in the global market [3–5]. As the size and complexity of software grow, so does the number of errors per 1000 lines of code. It becomes more difficult to obtain valuable data for reliability estimation. Software reliability models are only the way to estimate software reliability. Many software reliability estimation models have been developed or proposed in literature, but these developed models have been designed only for specific software applications, works in particular environment and on specific datasets and assumptions of the model. Thus, developing a new software reliability estimation model is a question behind the software developers. There are mainly three reasons that reflect the need to develop a new software reliability model.

- Existing models have been developed only based on traditional software development approaches and do not incorporate the latest SDLC approach [6–10].
- Available research publication has only 31% experimental research, and in this, only 13% pure experimental work has been found[11]. The reason behind this is the narrow failure dataset availability as software companies are not releasing their fault data for public research.
- Existing models are made application-specific, and these can precisely estimate the reliability of application-specific software only. These are not well proving their capacity if applied to other applications. The models are further developed based on the Non-homogeneous Poisson Process-based approach [12–23].

Software reliability models are divided into two categories: failure rate-based models and NHPP behavior-based models. The first category of software reliability models used in business and research were failure rate-based models [9]. However, these models need enhancement to incorporate the latest software development methods and estimate software reliability. Furthermore, more realistic assumptions need to be included in these models to be more suitable in the latest software development environments for reliability estimation. Keeping in mind the need to use the latest software development methods, the authors recommend the iterative method as a top-down refinement approach for software development. The more recent agile method, iterative enhancement method, and spiral method are well-known models supporting the iterative software development process [24, 25]. The authors in this paper propose a failure rate model known as Iterative Software Reliability (ISR) model which incorporates the behavior of these latest software development methods, mainly the iterative SDLC approach [26], for estimating the reliability of big data generating software. There is a necessity to add a factor that will reflect all shifting needs based on the defect analysis in each software development process model [27–29]. Identifying a varying need in each iteration using some factor could be very beneficial for iterative software development inappropriate resource usage. In this paper, varying conditions in each iteration are represented with an iterative factor. The iterative element's value is determined by the iterative parameter, which ranges from 0 to 1 based on the assumption that user acceptability improves with new capabilities in each iteration. The value of an iterative factor is obtained, representing all the requirements that have been changed from the previous iteration in current iteration releases using iterative parameters.

Value of iterative factor increases in some initial iterations because there is an abrupt change in requirements due to many defects found and added functionality in an iteration. Later, when the system has added almost all the functionalities for full implementation of the system, the value of the modulation factor represents that the change in requirements decreases, so its value changes from higher to lower one. When the value of the iterative parameter reaches again near to one, it is assumed that the system is reliable enough and has attained all the vital functionalities to fulfil the end-user needs. Thus, the iterative parameter reflects the level of user acceptance at each phase. Further in the cutting-edge software development environments, using the latest testing techniques, there is always a possibility of removing more than one fault at a time. Fault introduction and removal probabilities are well incorporated in the proposed ISR model development. To the best of authors' knowledge, no studies are available in the literature that includes such factor in the ensemble for reliability estimation via software reliability models. Further, in modeling software, reliability parameter estimation methods play an essential role. However, classical numerical optimization techniques are highly based on constraints, and they may not converge to maxima or minima in multimodal cases [7, 30]. To address non-differential, non-linear, and multimodal problems, new parameter optimization approaches based on nature-inspired optimization algorithms are available [31, 32]. This paper proposes model parameter optimization using a hybrid swarm evolutionary algorithm [33]. This algorithm incorporates the best feature of the artificial bee colony and differential evolution in parameter estimation. Swarm evolutionary algorithm is a new hybrid swarm-evolutionary algorithm for software reliability model parameter estimation is proposed in this paper. This algorithm is centered on the social behavior of artificial bee colonies (Yang 2010a; Abu-Mouti and El-Hawary 2012) and evolutionary behavior of DE algorithm (Storn and Price 1997). Here swarm intelligence of employee bee is enhanced for providing exploitation to provide a better local search of neighborhood positions using evolutionary principle based DE algorithm. Onlooker bee phase has been improved here by incorporating a new factor, showing the fitness probability of the ecological space.

The proposed ISR model is empirically validated using Eclipse and JDT software failure datasets. Furthermore, the Goodness-of-fit of the proposed work is estimated using Sum of Squared Errors (SSE), Mean Square Error (MSE), Accuracy of Estimate (AE), and Theil Statistics (TS). The proposed ISR model fits the estimation of software reliability under the iterative software development process. Summary of the significant contribution of our work is as follows:

1. Objective 1: To study existing software reliability estimation models and specially failure rate behavior-based models.
2. Objective 2: To propose a new model that considers the behavior of the recently used iterative software development life cycle processes.
3. Objective 3: To validate the proposed model (ISR) on twelve Eclipse common software failure datasets and six JDT software failure dataset iterations.
4. Objective 4: To compare the proposed model with five existing models named Jelinski Moranda Model, Shick–Wolverton Model, Goel Okumotto Imperfect debugging Model, GS Mahapatra and P Roy Model and Modified SW Model.

Proposed ISR model is found to be supporting software developers and end-users in estimating software reliability at each software development iteration and hence in software systems evolution. The rest of the paper is structured as follows: Section 2 discusses the existing literature in software reliability modelling and various parameter estimation algorithms used in software reliability model development. Then, in Section 3 proposed

software reliability model is discussed. In Section 4, experimental setup and results are discussed. Section 5 discusses the threats to validity and finally Section 6 concludes the paper with future scope.

2. Literature review

A thorough literature has been done by authors which is elaborated in 2 subsections of this section. The first section focusses on development of software reliability models of the software that has been developed using latest software development life cycle processes (SDLC). The next subsection presents a survey of parameter optimization algorithm that is the major requirement for precise software reliability measurement using software reliability models. Already existing parameter optimization algorithms applied on software reliability modeling can only better estimate the reliability of software that has been developed with their specific assumptions and applications.

2.1. Software reliability models and their assumptions

About 300 software reliability models have been developed in the last four decades. The developed models have specific environments, assumptions, and applications [34–40]. Failure rate behavior-based models are used to analyze the program failure rate per fault and study how failure changes at the time of failure in an interval of time. The JM model [8] is the first to estimate software reliability. This behavior-based debugging approach assumes the program initially has a fixed, constant, and unknown number of defects. The duration between failures is considered to be independent and distributed exponentially in these models. Table 1 discusses software reliability models along with their assumptions.

Table 1. Survey of Software Reliability Models

Reference	Year	Application domain and assumptions Related to software reliability models
[9]	1972	The first Markov process-based model assumes that the total number of initial software defects is unknown and fixed. Furthermore, it assumes that the duration between failures is always a random variable that is distributed exponentially.
[41]	1973	A Bayesian reliability growth model is presented here and assumes that the program is complete to work for a continuous time-period between the failures. It also considers a repair rule for program developers at each failure. It does not take into account the program's internal structure.
[42]	1988	This model fits well into a general framework of the Bayes problem and assumes a Bayesian approach for inference by considering the conditions as an empirical Bayes problem.
[10]	1978	The Hazard function is proportional to the current number of total fault content and the time since the previous failure in this model originated from the JM model.
[43]	1977	This model extends the JM Model and SW Model; It allows more than one fault at each time interval.
[44]	1979	It follows a Markov process like the JM Model. This characterizes the transition between the modules while execution as following the Markov property.
[43]	1979	This is evolved using the JM and geometric model-based and follows Poisson distribution-based failure rate. It assumes that the number of faults occurring at intervals follows a Poisson distribution with an intensity rate of Dki-1.
[45]	1998	It has extended the JM geometric model by describing the behavior of software as having safe and unsafe states.

Table 1 continued

Reference	Year	Application domain and assumptions related to software reliability models
[46]	1979	This model considers the phenomenon of imperfect debugging for software development and testing.
[13]	1981	A variation of the JM model develops this model. And assumes that: – Time separations between error detections – Number of errors per written instruction – Failure rate of the software is considered as proportional to current error content in software.
[47]	1981	This model is presented as a particular case of the JM and NHPP models.
[48]	1985	This is evolved by considering the Bayesian process in the JM model and Mein-hold and Singpurwalla model. It also assumes that it is easy to calculate the distribution of undetected errors at the end of testing to see the relative effects of uncertainty in several errors and fault detection efficiency.
[49]	1985	In this model, an alternative formulation of the JM and Little-wood models is presented. Here a formulation in terms of failure rate rather than inter failure time is given.
[50]	1991	It is assumed that various defects contribute differently to the failure rate. In addition, the software's structure is taken into account in this method.
[43]	2000	This model is extended from the JM Model. The effect of environmental factors has been incorporated in the model development.
[51]	2003	A Moranda de-eutrophication model is proposed by assuming time between failures as a statistically independent exponential random variable with a given failure rate.
[37]	1991	It has evolved from the JM model's assumptions and formulates the total expected software costs with two different release policies.
[51]	2006	This model extends the JM model by using a negative binomial prior distribution for the number of remaining faults and a Gamma distribution for the rate at which each fault becomes disclosed.
[15]	2011	This is the modification of the famous JM model, and it is based on cloud theory.
[16]	2012	This model considers imperfect debugging in fault removal and considers that when a failure occurs, then the detected fault is assumed to be removed with probability p, and it is not removed perfectly with probability q. It also assumes that a new fault may be generated with probability r.
[52]	2016	They discussed that software reliability analysis can be done at various phases during the development of engineering software. JM and SW SRGMs are two exceptional cases of this general SRGM.
[53]	1985	This model proposes that the reliability of computer software may be evaluated holistically by taking a Bayesian perspective, and it provides an alternate justification for the widely utilized JM model.
[54]	2017	To estimate the parameters of the JM model, objective Bayesian inference was developed. The Bayesian estimators, credible intervals, and coverage probability of the parameters are obtained using Gibbs sampling.
[55]	2018	1) It takes into account the impact of software upgrades on subsequent releases. 2) It also implies that software problems are classified as soft or hard depending on the amount of work and time required to correct them.
[56]	2018	They are assumed as a special case of the famous inflection S-shaped model and generalized GO model. Special attention is given to non-existence issues of MLE.
[57]	2018	A variable η is considered as a random variable to represent the uncertainty of FDR in the operating environment.
[58]	2019	The fault Removal process for multi-release OSS systems is assumed by considering the concept of change point.
[22]	2018	Assumes that all corrected issues in a current software release are used to determine the next software release; entropy-based methods assess uncertainty concerns.
[59]	2019	Complexity issues like the debugging process, coverage factor, and delay time Function in a distributed computing environment are concerned.
[60]	2021	Uncertain factors are taken into account while developing a logistic growth model for software reliability estimates.

Table 1 continued

Reference	Year	Application domain and assumptions related to software reliability models
[61]	2021	The extension predicts the software reliability of the Jelinski Moranda model to a stack of feature-specific models.
[62]	2021	Application of EM algorithm is done to NHPP software reliability assessment with generalized failure count data.
[63]	2020	The importance and need of software reliability from an industry perspective have been discussed.
[64]	2020	A new statistical time series and ARIMA approach-based software reliability prediction is proposed. time series based technique is very flexible as it needs a very few assumptions.
[65]	2019	Quantitative Testing Effort based estimate of software Reliability for Multi Release Open Source Software Systems is proposed. Testing Effort is assumed to be impacting the number of faults generated in the software at a particular time interval.
[66]	2021	Green computing based software systems is discussed here with some new modulation parameters that are quantifying the amount of fault introduced in the software.
[67]	2021	New iterative failure rate behavior based model is proposed for iterative software development life cycle approach based softwares. In each iteration new fault may be introduces and it assumes that some faults may gete flow from the previous iterations.
[68]	2022	Grey-based approach for estimating software reliability under nonhomogeneous Poisson process is proposed for software reliability assessment using mean value function.
[69]	2022	Various Machine learning techniques can be used in software reliability prediction. Various ML techniques have been used here for reliability prediction. But the dataset required must be large enough here.
[70]	2023	During testing process optimal time management is a need and an approach using S-curve two-dimensional software reliability growth model is proposed here.
[71]	2023	Reliability Assessment for Open-Source Software using Probabilistic approach is proposed. For assessing reliability more accurately Imperfect debugging along with Marine Predators Algorithm with six probabilistic models is used here.
[72]	2023	A new Model for software reliability growth estimation based on generalized inflection S-shaped is proposed. This has considered fault reduction factor and optimal release time for proposed model development.
[73]	2023	On the basis of assuming latest Emerging trends and future directions in software reliability growth modeling. Engineering reliability and risk assessment is discussed.
[74]	2023	Testing coverage is used here for software reliability growth modeling and considers uncertainty of operating environment.

2.2. Parameter optimization processes

Parameter optimization processes play a pivotal role in estimating the reliability of software. Traditional techniques [30, 73, 74] have been found in use for the estimation of parameters of software reliability models. Fatefully, all model parameters usually have non-linear relationships. Because of this, traditional techniques for optimizing parameters suffer various problems in finding the optimum value of models to predict software reliability better. In recent years, meta-heuristic algorithms have become very popular due to their simplicity, flexibility, derivative-free nature, and capability to avoid the local optima problem. These algorithms explore the feasible solution space using some specified rules. Several nature-inspired algorithms have been created in the literature and may be used to address numerical optimization-based issues in various domains [75–83]. Mirjalili [79] created a hybrid PSOGSA method for mathematical function optimization in 2010 by

combining PSO and PSOGSA. Furthermore, In 2014, Mirjalili [83] presented a binary optimization method based on a hybrid PSOGSA algorithm. Liu and Zhou [84] used a new QPSO (Quantum Particle Swarm Optimization) technique to solve high-dimensional complex issues in 2014. Li et al. [85] proposed ABC-assisted DE for ORPF (Optimal Reactive Power Flow) in 2013. They have used ABC in the DE algorithm to recover the shortcoming of large population requirements to avoid premature convergence. Tiwari et al.[86] presented a hybrid ABC method with DE, which he used to solve welded beam design issues. This method changed the employee bee phase position update equation and utilized DE for the onlooker bee phase position update. Sangeeta et al. [87] have suggested a magnetic navigation-based optimizer to analyze proper software reliability model parameters. Tripathi [88] has used the military Dog algorithm to estimate phone reviews and found it one of the most effective parameter estimation methods available. Sangeeta et al. [89] have created an ecological space-based hybrid swarm evolutionary method for parameter estimation in software reliability models. The authors [90] suggested a firefly-based multi-level picture thresholding method. Khan, Jabeen et al. [91] has proposed a new Metaheuristic algorithm in optimizing deep neural network model for software effort estimation. Kassaymeh et al.[92] has developed a new swarm optimizer for modeling software reliability prediction problems. Authors suggested that proposed algorithm can be applied in various optimization problems. Lakra and Chug [93] has given a study on Application of metaheuristic techniques in software quality prediction. Lilly, Jothi [94] has enhanced software reliability and fault detection process using hybrid brainstorm optimization-based LSTM Model. Kumar and Gopalan [95] have developed an efficient parameter optimization of software reliability growth model by using chaotic grey wolf optimization algorithm. Singh, Kumar et al. [96] has proposed a new Adaptive infinite impulse response system identification using teacher learner-based optimization algorithm and has analyzed good prediction on the basis of proposed algorithm. Rakhi and Pahuja [97] has developed a method for solving reliability redundancy allocation problem using grey wolf optimization algorithm. Kaushik et al. [98] has discussed about the role of neural networks and metaheuristics in agile software development effort estimation. Yadav N. and Yadav V. [99] proposed a software reliability prediction and optimization approach using machine learning algorithms.

2.3. Insights of the literature survey

From the literature in Table 1 it has been found that failure rate behavior-based models are the earliest models and these are the basis of newly proposed models. These models are studying the program failure rate per fault and are considering only the traditional fixed behavior-based waterfall approach for software development but this is a static and oldest approach. There is a need to propose new model that consider latest method of software development in software reliability analysis on the basis of Bug report analysis. Hybrid algorithms are proving to be effective in a variety of domains. These techniques can be used to estimate the parameters of software reliability models. These swarm evolutionary algorithms are found to be performing good when applied to specific application and dataset. To make their appropriate usage, a new algorithm specific to software reliability application is used in this paper. This research employed a hybrid swarm evolutionary feature-based ABC-DE method for the suggested model parameter estimation. The code for the algorithm is available in the replication package available at https://zenodo.org/records/13147825.

3. Proposed model

Failure rate behavior-based models are the oldest of the software reliability models. These have primarily been used in academics and industry to assess software product reliability. Considering the most recent software development environments and technology, a new model based on the iterative software development cycle process has been proposed. Traditional software development lifecycle methods focus on the models available in the literature. The waterfall software development lifecycle method underpins all existing models. However, modern software development approaches outperform the waterfall software development life cycle methodology. These are life cycle processes that are iterative, spiral, and adaptable. The suggested model uses the most up-to-date software development methods, which are based on the iterative software development approach. The suggested model, which is concerned with the iterative software development process, is developed from a generic collection of failure-rate based models. The proposed Iterative Software Reliability (ISR) model is derived from a general group of failure-rate models and is concerned with the iterative software development process. The failure rate model incorporates changing demands with additional capabilities in each software development iteration using the iterative factor γ that reflects varying requirements in each iteration. The changing needs in each software development iteration include a report of bugs, additional features, and the amount of testing effort. These parameters are used to identify the number of flaws in each iteration that have been either injected or eliminated in each subsequent software development iterations. The modulation factor (γ) represents all of these shifting requirements during the software development period. In future iterations, new mistakes or mutually dependent errors may be introduced. With each iteration of the software development life cycle, the suggested model implies that new defects will be inserted with a probability (p_i) and removed with (r_i) each iteration. As a result, the varied demands in each iteration are unique, and they change according to 1.

3.1. Modified reliability estimation model

3.1.1. Assumptions of proposed model

1. An iteration begins with an undetermined and constant number of software faults.
2. Each of the faults is either mutually dependent or independent, and they may equally cause a failure during testing.
3. The time period between fault occurrences is thought to be independent.
4. The program failure rate is assumed initially to be constant; however, it varies with each iteration at failure times.
5. The failure rate may increase or decrease, depending on the remaining faults in software and the modulation factor.
6. Whenever a failure occurs in an iteration, an immediately debugging effort will be initiated. Each iteration has the fault removal probability, not removed with a probability. Instead, the fault is introduced with a probability. Here, probability.
7. When a fault occurs in iteration, it may not be ideally removed. This may pass in successive iterations because the testing environment of each iteration is different

from the operating environment. Regenerated fault in subsequent iterations introduces imperfect debugging in the proposed model.

The initial iterative faults are fixed and constant but unknown for iteration. Later on, new faults may get injected from the previous iteration, or new faults may get introduced in the current iteration. Based on these faults, developers can change resource allocation in debugging iteration. The Iterative factor represents the modified need for resources and integrates the iterative SDLC process in software reliability modeling. Iterative element defined in Equation (1).

$$\gamma = \mu + (1 - \mu)/\mu (0 < \mu \leq 10) \tag{1}$$

Here, μ the iterative parameter represents newly added functionality and level of user acceptance in an iteration. Its value is near zero initially and becomes almost one until the final software development iteration.

3.1.2. Model formulation

$\lambda(t_i)$, i.e., the failure rate function is modeled in Equation (2).

$$\lambda(t_i) = \phi \left[N - \frac{(n_{i-1})(\gamma)}{(i-1)} (p - r) \right], \quad i - 1, 2 \ldots N \tag{2}$$

Here γ – iterative factor that represents varying needs in each iteration n^{i-1} – cumulative number of failures N – number of initial faults ϕ – constant of proportionality $F(t_i)$, i.e., cumulative density function and $R(t_i)$, i.e., reliability function is given in eqations (3) and (4)

$$F(t_i) = 1 - e^{\left[-\phi \left[N - \frac{\left(n_{i-1}\right)\gamma}{i-1} (p-r) \right] t_i \right]} \tag{3}$$

$$R(t_i) = e^{\left[-\phi \left[N - \frac{\left(n_{i-1}\right)\gamma}{i-1} (p-r) \right] t_i \right]} \tag{4}$$

3.1.3. Parameter estimation

There are three parameters N, n and ϕ, these unknown parameters are measured at different values of γ. MLE function is used to estimate the value of all parameters. The probability density function $f(t)$ for the Proposed Iterative Software Reliability (ISR) model is given in Equation (5).

$$f(t_i) = \phi \left[N - \frac{(n_{i-1})\,\gamma}{i-1} (p - r) \right] e^{\left[-\phi \left[N - \frac{\left(n_{i-1}\right)\gamma}{i-1} (p-r) \right] t_i \right]} \tag{5}$$

Likelihood function $L(N)$ is calculated in Equation (6) using Equation (5).

$$\begin{aligned} \mathrm{L(N)} &= \prod f\left(t_i\right) \\ &= \prod_{i=1}^{n}\left[\phi\left[N-\frac{\left(n_{i-1}\right)\gamma}{i-1}(p-r)\right] e^{\left[-\phi\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)\gamma}{i-1}(p-r)\right]t_i\right]}\right. \end{aligned} \tag{6}$$

Log-likelihood function in parameter estimation needs calculation of partial derivatives w.r.t. N, n and ϕ, respectively. It then equates them to zero. Maximum likelihood estimates of parameters are obtained using (7), (8), and (9).

$$\frac{n}{\phi}-\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)y}{i-1}(p-r)\right]t \tag{7}$$

$$n=\phi\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)y}{i-1}(p-r)\right]t \tag{8}$$

$$\frac{1}{\sum_{i=1}^{n}\left[N-\frac{\left(n_{i-1}\right)y}{i-1}(p-r)\right]}-\phi\sum_{i=1}^{n}t_i \tag{9}$$

4. Experimental setup and results

4.1. Collection of bug data used in experimentation

The proposed ISR model is tested using a bug report taken from the Tera Promise repository. Datasets were preprocessed using various preprocessing techniques like outlier analysis, removal of redundant/missing values, etc. The datasets obtained after preprocessing are available at the repository https://zenodo.org/records/13147825 to encourage replication of the results. The bug report contains a table with *bug_id*, summary, description, time, associated commit, commit status, and saved files. Dataset is extracted from this file and structured in a required format. The multiple iteration dataset from this bug report is used in this paper to test the capability of proposed model to predict the remaining number of errors, reliability, and failure intensity of the OSS system.

4.2. Results

The proposed ISR model objective is to estimate the precise reliability growth of the software and includes the model parameters N, n, ϕ, γ, p and r. Probability p and r is dependent on the type of project and human skill involved in software testing. Based on these factors, the number of faults removed in testing is assumed to be 96%, and the number of faults introduced is considered as 2%. γ is defined as the new modulation factor showing the changing needs of the iterative software development process. γ values

are calculated using the modulation parameter (μ). The calculated values of γ and μ with varying parameters N, n, and ϕ are estimated using the hybrid Artificial Bee Colony and Differential Evolution Algorithm (the code for the used algorithm is available in the replication package; link available at https://zenodo.org/records/13147825) to maximize the log-likelihood function value. Model parameter values are estimated using the MLE technique. 80% of data has been used to estimate parameters. The predicted number of remaining errors, i.e., the expectation, is calculated for each value of modulation factor (γ). Goodness-of-fit for the Proposed Iterative Software Reliability (ISR) model is measured using SSE, MSE, AE, and TS for each application dataset [33]. These statistics are used further for comparison between iterations of datasets. Initially, the system is assumed to be having minimum features. Later on, with time, the requirements of the end-users need to be fulfilled; they can add more and more features to the system. These additional changes in upcoming iterations may incorporate some new errors and reduce errors from the earlier iterations. Depending on these new conditions, there is a need to change the iterative system development. Accordingly, the proposed work the parameter changes its shape. In the proposed Iterative Software Reliability (ISR) model, estimated values of represent additional feature changes incorporated in the current iteration, differing from previous ones. For initial iteration releases of Eclipse and JDT open-source software systems, the value should be estimated so that a critical sample of requirements is implemented with the minor features in the system. As time passes and more and more iterations are added with changing functionality in each iteration, the value should fluctuate from lower to higher with the user acceptance and increase in functionality of the system with time. Depending on values, it is estimated for each iteration release, and the model is implemented.

4.3. Result analysis

In this section the analysis of Eclipse software and JDT software failure dataset is done using Proposed model and other failure rate behaviour models. Table 2 shows the failure rate based models used in this research work and Table 3 shows the parameter estimation using Eclipse software failure dataset on various existing models along with the proposed model.

Table 2. Summary of failure rate based software reliability models

Sr. no.	Model name	Failure intensity	No. of actual parameters
1	Jelinski–Moranda model	$2(t) = [N - (i-1)]$	2
2	Schick–Wolverton model	$2(t) = \phi[N - (i-1)]t$	2
3	Goel–Okumotto imperfect debugging model	$2(t) - \varphi[N - p(i-1)]$	3
4	G.S. Mahapatra and P. Roy model [16]	$2(t) - f[N - p(i-1) + r(i-1)]$	4
5	Modified S–W model	$2(t) = \rho\,[N - (n_{2-1})]$	3
6	Proposed Iterative Software Reliability (ISR) model	$2(t) = \phi\left[N - \frac{(n_{i-1})\,(p)}{(i-1)}(p-r)\right]$	6

Table 3. Parameter estimation using Eclipse software failure dataset

Model name	Parameter estimated values (fit)	Predicted number of remaining errors	
		expectation (fit)	remarks (predicted)
Iteration 1.0			
JM model	$\phi = 5.74\text{E}{-6}, N = 4$	2	over estimation
GOI model	$\phi = 1.64\text{E}{-5}, N = 3$	1	exact estimation
SW model	$\phi = 2.124\text{E}{-6}, N = 3$	1	exact estimation
Mahapatra model	$\phi = 7.77\text{E}{-7}, N = 3$	1	exact estimation
SWM model	$\phi = 2.23\text{E}{-5}, N = 2, n = 58$	0	under estimation
Proposed ISR model	$\phi = 2.86\text{E}{-5}, N = 5, n = 11, \gamma = 2.2666, \mu = 0.3419$	3	over estimation
Iteration 2.0			
JM model	$\phi = 2.68\text{E}{-5}, N = 8$	-11	under estimation
GOI model	$\phi = 2.89\text{E}{-6}, N = 28$	9	over estimation
SW model	$\phi = 4.70\text{E}{-6}, N = 28$	9	over estimation
Mahapatra model	$\phi = 2.53\text{E}{-5}, N = 23$	4	under estimation
SWM model	$\phi = 1.52\text{E}{-6}, N = 9, n = 209$	-10	under estimation
Proposed ISR model	$\phi = 4.70\text{E}{-6}, N = 27, n = 362, \gamma = 2.8763, \mu = 0.2779$	8	over estimation
Iteration 2.1			
JM model	$\phi = 5.039\text{E}{-6}, N = 27$	4	under estimation
GOI model	$\phi = 1.95\text{E}{-6}, N = 26$	3	under estimation
SW model	$\phi = 3.83\text{E}{-6}, N = 29$	6	exact estimation
Mahapatra model	$\phi = 3.20\text{E}{-6}, N = 28$	5	under estimation
SWM model	$\phi = 1.36\text{E}{-5}, N = 18, n = 316$	-5	under estimation
Proposed ISR model	$\phi = 1.67\text{E}{-6}, N = 30, n = 415, \gamma = 2.6343, \mu = 0.2999$	7	over estimation
Iteration 3.0			
JM model	$\phi = 2.40\text{E}{-6}, N = 124$	48	over estimation
GOI model	$\phi = 1.37\text{E}{-6}, N = 100$	24	over estimation
SW model	$\phi = 2.92\text{E}{-5}, N = 121$	45	over estimation
Mahapatra model	$\phi = 1.25\text{E}{-6}, N = 125$	49	over estimation
SWM model	$\phi = 1.29\text{E}{-6}, N = 123, n = 298$	47	over estimation
Proposed ISR model	$\phi = 2.98\text{E}{-5}, N = 102, n = 5690, \gamma = 3.7891, \mu = 0.2188$	26	over estimation
Iteration 3.1			
JM model	$\phi = 2.99\text{E}{-5}, N = 100$	-8	under estimation
GOI model	$\phi = 3.07\text{E}{-7}, N = 126$	18	under estimation
SW model	$\phi = 2.99\text{E}{-5}, N = 125$	17	under estimation
Mahapatra model	$\phi = 6.62\text{E}{-6}, N = 130$	22	under estimation
SWM model	$\phi = 1.36\text{E}{-5}, N = 135, n = 4979$	27	under estimation
Proposed ISR model	$\phi = 2.96\text{E}{-5}, N = 136, n = 6132, \gamma = 3.2217, \mu = 0.2519$	28	exact estimation
Iteration 3.2			
JM model	$\phi = 2.21\text{E}{-6}, N = 122$	27	over estimation
GOI model	$\phi = 8.61\text{E}{-6}, N = 106$	11	under estimation
SW model	$\phi = 5.40\text{E}{-6}, N = 90$	-5	under estimation
Mahapatra model	$\phi = 1.25\text{E}{-6}, N = 115$	20	under estimation
SWM model	$\phi = 2.08\text{E}{-6}, N = 132, n = 4108$	37	over estimation
Proposed ISR model	$\phi = 2.98\text{E}{-5}, N = 122, n = 6910, \gamma = 2.5788, \mu = 0.3055$	27	over estimation
Iteration 3.3			

Table 3 continued

JM model	$\phi = 2.95\text{E}{-5}, N = 117$	22	under estimation
GOI model	$\phi = 1.25\text{E}{-6}, N = 115$	20	under estimation
SW model	$\phi = 2.95\text{E}{-5}, N = 123$	28	over estimation
Mahapatra model	$\phi = 3.06\text{E}{-6}, N = 114$	19	under estimation
SWM model	$\phi = 5.31\text{E}{-6}, N = 121, n = 3129$	26	over estimation
Proposed ISR model	$\phi = 2.96\text{E}{-5}, N = 123, n = 7879, \gamma = 2.3121,$ $\mu = 0.336$	28	over estimation
Iteration 3.4			
JM model	$\phi = 8.28\text{E}{-6}, N = 53$	13	over estimation
GOI model	$\phi = 3.06\text{E}{-6}, N = 50$	10	under estimation
SW model	$\phi = 2.93\text{E}{-5}, N = 54$	14	over estimation
Mahapatra model	$\phi = 1.63\text{E}{-5}, N = 55$	15	over estimation
SWM model	$\phi = 1.72\text{E}{-5}, N = 52, n = 2094$	12	over estimation
Proposed ISR model	$\phi = 2.92\text{E}{-5}, N = 53, n = 1411, \gamma = 2.7295,$ $\mu = 0.2908$	13	over estimation
Iteration 3.5			
JM model	$\phi = 1.63\text{E}{-6}, N = 30$	10	over estimation
GOI model	$\phi = 3.81\text{E}{-6}, N = 29$	9	over estimation
SW model	$\phi = 1.70\text{E}{-7}, N = 29$	9	over estimation
Mahapatra model	$\phi = 3.81\text{E}{-6}, N = 29$	9	over estimation
SWM model	$\phi = 1.26\text{E}{-5}, N = 17, n = 2663$	-3	under estimation
Proposed ISR model	$\phi = 2.82\text{E}{-5}, N = 25, n = 416, \gamma = 2.6434,$ $\mu = 0.2990$	5	under estimation
Iteration 3.6			
JM model	$\phi = 6.24\text{E}{-7}, N = 28$	6	exact estimation
GOI model	$\phi = 3.208\text{E}{-6}, N = 28$	6	exact estimation
SW model	$\phi = 3.74\text{E}{-6}, N = 27$	5	under estimation
Mahapatra model	$\phi = 6.65\text{E}{-7}, N = 28$	6	exact estimation
SWM model	$\phi = 2.97\text{E}{-5}, N = 17, n = 2624$	-5	under estimation
Proposed ISR model	$\phi = 2.94\text{E}{-5}, N = 30, n = 401, \gamma = 2.0435,$ $\mu = 0.3747$	8	over estimation
Iteration 4.1			
JM model	$\phi = 2.20\text{E}{-5}, N = 19$	7	over estimation
GOI model	$\phi = 1.58\text{E}{-5}, N = 17$	5	over estimation
SW model	$\phi = 1.93\text{E}{-5}, N = 19$	7	over estimation
Mahapatra model	$\phi = 5.53\text{E}{-6}, N = 17$	5	over estimation
SWM model	$\phi = 2.68\text{E}{-5}, N = 8, n = 1529$	-4	under estimation
Proposed ISR model	$\phi = 2.87\text{E}{-5}, N = 16, n = 136, \gamma = 1.1258,$ $\mu = 0.7026$	4	over estimation
Iteration 4.2			
JM model	$\phi = 1.92\text{E}{-5}, N = 31$	7	exact estimation
GOI model	$\phi = 3.20\text{E}{-6}, N = 30$	6	under estimation
SW model	$\phi = 6.50\text{E}{-6}, N = 31$	7	exact estimation
Mahapatra model	$\phi = 7.29\text{E}{-7}, N = 31$	7	exact estimation
SWM model	$\phi = 2.97\text{E}{-5}, N = 30, n = 3729$	6	under estimation
Proposed ISR model	$\phi = 2.82\text{E}{-5}, N = 32, n = 513, \gamma = 1.00009,$ $\mu = 0.9904$	8	over estimation

4.3.1. Data analysis of Eclipse software

In the first released iteration of the Eclipse project, the estimated value of γ is 2.2666 and represents that the software has fewer features with the least number of bugs. A new iteration is released, more and more features are added, and value γ reflects all changing requirements incorporated in the next released iteration. The value γ reflects minute changes in minor iteration releases because it involves only a few feature addition and bug fixing. After iteration 2.0, iteration 2.1 has been released with minor feature addition and bug fixing. The value of γ this iteration is 2.8763 in iteration 2.0, but within a minute, it changes in iteration 2.1 γ, takes the value as 2.6343. This value is differing by only a small amount from the value in iteration 2.0. In major iteration release 3.0, the value is γ 3.7891, showing the major change in its value from the 2.1 version. Further in minor releases of iteration 3.0, values γ are changing as 3.2217, 2.5788, 2.3121, 2.7295, 2.6434, and 2.0435 for iteration numbers $3.1, 3.2, 3.3, 3.4, 3.5$, and 3.6, respectively. Similarly, in iteration 4.0, the γ parameter's value reflects all major iteration's requirement updates, the value γ is 1.1258, and in its minor iteration release 4.1, the value γ is 1.0009. Using estimated parameter values, prediction of the remaining number of errors is made, remarks are made whether the model overestimates, underestimates, or precisely estimates the number of faults remaining in the system.

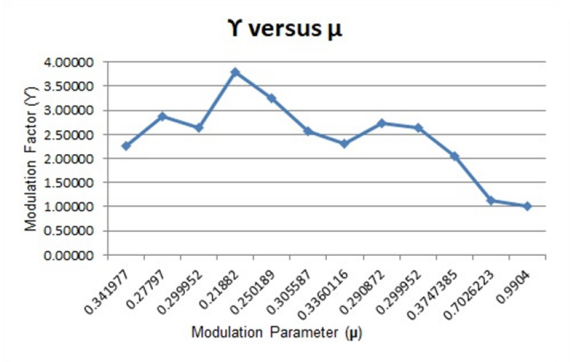

Figure 1. The plot of γ versus μ for Eclipse dataset

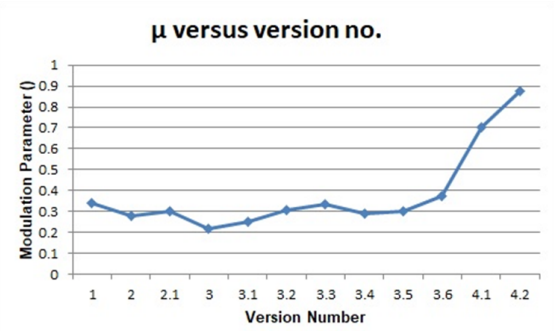

Figure 2. The plot of μ versus iteration for Eclipse dataset

Figure 1 represents how the modulation factor values change with respect to the modulation parameter for each iteration. Estimated values of the parameter are significantly reflecting all the changing requirements for each upcoming iteration numerically. These values meaningfully represent how much impact is of adding and removing new features with user acceptance in each upcoming iteration.

Figure 2 represent how the value of changes increases with the growth of the iterative software development process.

Table 4. Goodness-of-fit estimated using Eclipse software failure dataset

Sr. No.	Model	SSE	MSE	AE	TS
Iteration 1.0					
1	JM model	5.23	5	0.333	59.761
2	GOI model	1.51	1	0.0189	**26.721**

Table 4 continued

3	SW model	17.31	17	0.666	110.19
4	Mahapatra model	1.26	1	0.013	26.722
5	SWM model	5	4.9	0.297	65.94
6	**Proposed ISR model**	**1.24**	**0.666**	**0.012**	37.79
Iteration 2.0					
1	JM model	375.51	17.045	0.225	27.664
2	GOI model	92.005	4.181	**0.125**	13.702
3	SW model	218.81	9.909	0.266	21.092
4	Mahapatra model	55.46	2.51	0.166	12.371
5	SWM model	528.7	15.89	0.495	34.57
6	**Proposed ISR model**	**45.12**	**2.5**	**0.125**	**10.594**
Iteration 2.1					
1	JM model	452.2	16.74	0.054	22.985
2	GOI model	133.34	4.925	0.035	12.468
3	SW model	126.6	4.666	0.034	**12.136**
4	Mahapatra model	131.31	4.851	0.039	12.374
5	SWM model	118.67	**3.89**	0.028	13.9
6	**Proposed ISR model**	**101.02**	4.391	**0.023**	14.42
Iteration 3.0					
1	JM model	56948	605.82	0.302	43.602
2	GOI model	45510.1	559.22		38.978
3	SW model	39981	494.69	0.292	**36.507**
4	Mahapatra model	55122.2	586.4	0.395	42.898
5	SWM model	66890.4	679.32	**0.279**	54.89
6	**Proposed ISR model**	**39705.2**	**484.14**	0.584	52.069
Iteration 3.1					
1	JM model	**6091.1**	45.45	0.016	8.476
2	GOI model	1018.8	7.59	0.005	**3.465**
3	SW model	1123	8.38	0.0065	3.639
4	Mahapatra model	1015	**7.57**	**0.005**	3.56
5	SWM model	2000.7	15.9	0.006	5.69
6	**Proposed ISR model**	1589.82	12.22	0.007	4.551
Iteration 3.2					
1	JM model	13940.73	119.14	0.05	15.655
2	GOI model	7246.05	61.931	0.042	11.287
3	SW model	12585	107.564	0.111	14.871
4	Mahapatra model	7794.28	68.973	0.107	12.416
5	SWM model	9685.12	89.67	0.357	20.85
6	**Proposed ISR model**	**952.2**	**8.136**	**0.034**	**4.091**
Iteration 3.3					
1	JM model	6810.14	58.239	0.025	10.945
2	GOI model	9371.1	80.094	0.042	12.835
3	SW model	7318.81	62.547	0.034	11.342
4	Mahapatra model	2217.4	**18.948**	0.05	9.243
5	SWM model	2903.9	27.872	0.068	12.09
6	**Proposed ISR model**	**2203.05**	19.495	**0.025**	**7.099**
Iteration 3.4					
1	JM model	1153.32	23.53	0.058	50.122
2	GOI model	1545.41	1320.9	0.546	52.124
3	SW model	1153.73	23.53	0.049	15.914
4	Mahapatra model	905.4	18.46	0.058	14.099
5	SWM model	3589.96	29.79	0.089	34.956
6	**Proposed ISR model**	**520.23**	**11.55**	**0.039**	**12.809**

Table 4 continued

Iteration 3.5					
1	JM model	373.3	15.542	0.153	24.526
2	GOI model	137.74	5.7083	0.076	14.864
3	SW model	551.1	22.958	0.152	29.808
4	Mahapatra model	137.8	5.708	0.077	14.864
5	SWM model	489.74	18.24	0.0204	24.95
6	**Proposed ISR model**	**86.62**	**4.3**	**0.077**	**12.88**
Iteration 3.6					
1	JM model	199.91	7.654	0.036	16.064
2	GOI model	160.01	6.154	0.071	14.402
3	SW model	154.54	5.923	0.056	14.129
4	Mahapatra model	152.23	5.864	r|0.046	14.037
5	SWM model	315	14.92	0.184	23.68
6	**Proposed ISR model**	**91.15**	**4.136**	**0.035**	**14.669**
Iteration 4.1					
1	JM model	286.65	22	0.266	48.025
2	GOI model	111.21	8.538	0.256	29.91
3	SW model	284.86	22.1	0.266	48.025
4	Mahapatra model	129	9.923	0.133	32.254
5	SWM model	178.22	7.34	0.0836	**18.48**
6	**Proposed ISR model**	**49.95**	**5.444**	**0.246**	26.335
Iteration 4.2					
1	JM model	441.12	15.206	0.323	20.576
2	GOI model	121.21	4.172	0.24	10.778
3	SW model	193.34	6.655	0.289	13.612
4	Mahapatra model	166.65	5.724	0.224	12.624
5	SWM model	704.86	26.87	0.593	49.056
6	**Proposed ISR model**	**72.235**	**2.88**	**0.129**	**10.688**

The Goodness-of-fit of the models is shown in Table 4. From the analysis of results, it is found that the Proposed ISR model is fitted well in terms of SSE, MSE, and AE in iterations 1.0 and 2.1. But GOI model is beating the Proposed Iterative Software Reliability (ISR) model in terms of TS in iteration number 1.0 of the Eclipse software failure dataset. In iteration 2.0 Proposed ISR model is beating other models in terms of SSE and AE but also performing good. SW, and SWM are performing better than the proposed ISR model. In iteration 3.0, the Proposed Iterative Software Reliability (ISR) model performs better than other SSE and MSE, but its AE and TS values are exhausted by other models. In iteration number 3.1 Proposed Iterative Software Reliability (ISR) model is found to be less fitted than other models. Again, in further iterations of 3.2, 3.3, 3.4, 3.5, 3.6, and 4.2, the Proposed ISR) model estimates parameters more precisely than other models. In iteration number 3.3 and 4.1, the Proposed ISR model is not much good than the Mahapatra model in terms of MSE and TS. But they are performing a good fit than other models in terms of SSE, MSE and TS. From all the data analysis done in Table 4, it is found that the Proposed Iterative Software Reliability (ISR) model is performing fighting fit in terms of various Goodness-of-fit criteria and can be used for predicting the reliability of system software.

Optimal iteration releases can be selected based on reliability. The iterations 3.3, 3.6, and 4.1 are found to have higher reliability, and there are fewer falls in reliability values of these versions than other released iteration reliabilities. All iteration's reliability values are found to be decreasing up to 89%. Failure intensity graphs in Figure 4 show how

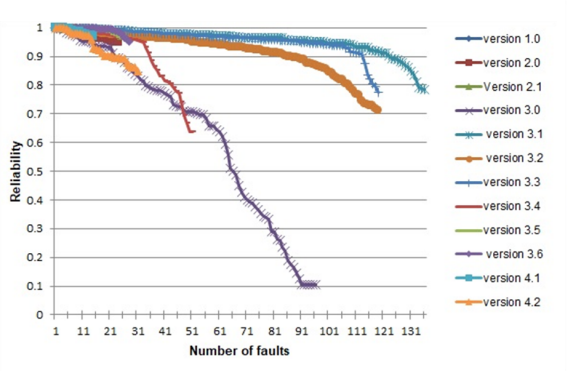

Figure 3. The plot of the reliability function for various iterations of the Eclipse failure dataset

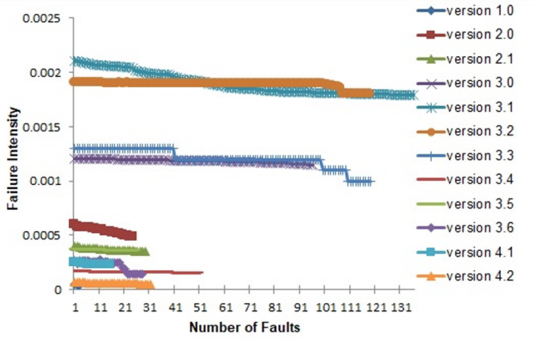

Figure 4. Plot of failure Intensity for various iterations for Eclipse failure dataset

much there is a decrease in failure behavior as the number of fault count increases. The iterations 3.3, 3.6, and 4.1 releases have a much-decreasing failure rate. There decrease in failure rate behavior is depicting that with time, the reliability of these systems increases. Reliability analysis has been done in Figure 3 and failure Intensity has been shown in Figure 4. Developers could decide the quality of the released iteration using estimated reliability. This is also helpful for end-users in choosing which iteration release will be most reliable in the future.

4.3.2. Result and data analysis of JDT dataset

In this section, analysis of the JDT software failure dataset is done using the proposed Iterative Software Reliability (ISR) model and other failure rate behavior-based models. Table 5 is showing the estimated values of model parameters.

Table 5. Parameter estimation using JDT software failure dataset

Model Name	Parameter estimated values (fit)	Predicted number of remaining errors	
		expectation (fit)	remarks (predicted)
Iteration 3.2.0			
JM model	$\phi = 2.35\text{E}{-6}, N = 50$	3	under estimation
GOI model	$\phi = 8.16\text{E}{-7}, N = 56$	9	under estimation
SW model	$\phi = 2.17\text{E}{-6}, N = 58$	11	under estimation
Mahapatra model	$\phi = 1.21\text{E}{-6}, N = 57$	10	under estimation
SWM model	$\phi = 2.91\text{E}{-5}, N = 52, b = 1723$	5	under estimation
Proposed ISR model	$\phi = 2.91\text{E}{-5}, N = 60, n = 1788, \gamma = 4.8479,$ $\mu = 0.1710$	13	over estimation
Iteration 3.3.0			
JM model	$\phi = 2.78\text{E}{-5}, N = 9$	-5	under estimation
GOI model	$\phi = 1.48\text{E}{-5}, N = 16$	2	under estimation
SW model	$\phi = 4.70\text{E}{-6}, N = 8$	-6	under estimation
Mahapatra model	$\phi = 8.92\text{E}{-6}, N = 26$	12	over estimation
SWM model	$\phi = 2.99\text{E}{-5}, N = 9, n = 1028$	-5	under estimation
Proposed ISR model	$\phi = 2.97\text{E}{-5}, N = 19, n = 179, \gamma = 3.8869,$ $\mu = 0.2140$	5	over estimation

Table 5 continued

Iteration 3.4.0			
JM model	$\phi = 8.86\text{E}{-6}, N = 18$	7	over estimation
GOI model	$\phi = 2.35\text{E}{-6}, N = 18$	7	over estimation
SW model	$\phi = 3.83\text{E}{-6}, N = 16$	5	over estimation
Mahapatra model	$\phi = 1.03\text{E}{-5}, N = 28$	17	over estimation
SWM model	$\phi = 2.91\text{E}{-5}, N = 13, n = 1124$	2	under estimation
Proposed ISR model	$\phi = 2.92\text{E}{-5}, N = 18, n = 119, \gamma = 2.9993,$ $\mu = 0.2680$	7	over estimation
Iteration 3.5.0			
JM model	$\phi = 2.49\text{E}{-5}, N = 8$	4	over estimation
GOI model	$\phi = 1.06\text{E}{-6}, N = 13$	9	over estimation
SW model	$\phi = 2.92\text{E}{-5}, N = 10$	6	over estimation
Mahapatra model	$\phi = 4.78\text{E}{-7}, N = 5$	1	exact estimation
SWM model	$\phi = 2.31\text{E}{-5}, N = 3, n = 137$	-1	under estimation
Proposed ISR model	$\phi = 2.96\text{E}{-5}, N = 7, n = 25, \gamma = 2.0918,$ $\mu = 0.3670$	3	over estimation
Iteration 3.6.0			
JM model	$\phi = 2.93\text{E}{-5}, N = 4$	-8	under estimation
GOI model	$\phi = 2.01\text{E}{-5}, N = 11$	-1	under estimation
SW model	$\phi = 2.99\text{E}{-5}, N = 29$		over estimation
Mahapatra model	$\phi = 1.26\text{E}{-5}, N = 85$	13	over estimation
SWM model	$\phi = 1.34\text{E}{-5}, N = 15, n = 134$	3	exact estimation
Proposed ISR model	$\phi = 2.67\text{E}{-5}, N = 18, n = 171, \gamma = 1.1529,$ $\mu = 0.678$	6	over estimation
Iteration 3.7.0			
JM model	$\phi = 1.50\text{E}{-6}, N = 16$	7	over estimation
GOI model	$\phi = 1.58\text{E}{-6}, N = 13$	4	over estimation
SW model	$\phi = 5.40\text{E}{-6}, N = 26$	17	over estimation
Mahapatra model	$\phi = 5.30\text{E}{-6}, N = 12$	3	exact estimation
SWM model	$\phi = 2.69\text{E}{-5}, N = 14, n = 1599$	5	over estimation
Proposed ISR model	$\phi = 2.74\text{E}{-5}, N = 15, n = 78, \gamma = 1.052,$ $\mu = 0.724$	6	over estimation

A predicted number of errors estimated using the Proposed Iterative Software Reliability (ISR) model is found to be representing accurate estimation when compared with the actual dataset values. There is a slight deviation only because of the poor debugging behavior in the iterative software development process. Remarks are made about whether the model underestimates, overestimates, or precisely estimates the number of remaining errors in the system.

Figures 5 and 6 are showing γ and μ behavior with respect to change in iterations. With each iteration, the value of functionality changes and corresponding to its requirements are made in upcoming iterations. Accordingly value of $\gamma\mu$ and changes their shape quantitatively, as shown in Figure 5. Figure 6 is showing the behavior of μ with change in iteration numbers. At initial iteration number 3.2, there are the least features and maximum changing needs from the end-users; the value $\gamma\,\mu$ are 4.8479 and 0.171, respectively. These values are varying in the same way in each iteration release. For example, in iteration 3.3, the value γ is 3.8869 and μ is 0.2140. For iteration 3.4 value γ is 2.9993, and the value μ is 0.2680 showing perfectly how the requirements and functionality change in each iteration with end-user acceptance. In the end, γ values move towards completion of end-user requirements, and acceptance level also increases, the value of γ the changes as 1.1052 and μ as 0.7240.

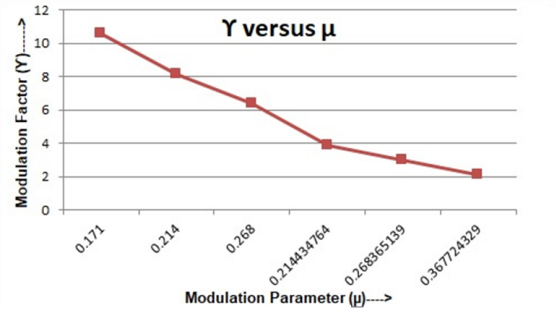

Figure 5. Plot of γ versus μ for JDT dataset

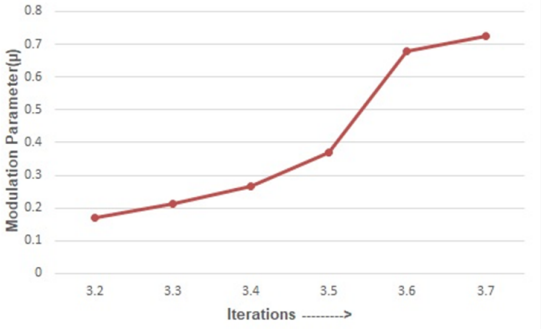

Figure 6. The plot ofμ versus Iteration for JDT dataset

Figures 5 and 6 are showing well, the iterative behavior of software development using γ and μ values.

Table 6. Goodness-of-fit of JDT software failure dataset

Sr. No.	model	SSE	MSE	AE	TS
1	JM model	179.79	11.187	0.0169	29.133
2	GOI model	317.67	5.561	**0.011**	6.719
3	SW model	1056	15.678	0.018	19.083
4	Mahapatra model	56500.09	991.228	0.915	89.706
5	SWM model	2989.06	29.689	0.0214	41.075
6	**Proposed ISR model**	**171.89**	**3.226**	0.0169	**5.585**
Iteration 3.3.0					
1	JM model	404	33.666	0.285	63.089
2	GOI model	989.9	61.825	0.666	68.479
3	SW model	589.1	40.78	0.39	69.93
4	Mahapatra model	604.01	37.75	0.388	53.515
5	SWM model	486.32	33.96	0.31	65.94
6	**Proposed ISR model**	**31**	**2.583**	**0.114**	**16.295**
Iteration 3.4.0					
1	JM model	37	12.33	0.6	82.02
2	GOI model	206.56	17.16	0.285	45.051
3	SW model	750.9	68	0.301	90.789
4	Mahapatra model	1241.01	103.41	**1.112**	110.574
5	SWM model	1839.56	146.92	1.947	130.05
6	**Proposed ISR model**	**91**	**11.37**	0.214	**3.365**
Iteration 3.5.0					
1	JM model	57	4.384	0.066	**21.44**
2	GOI model	76.53	25.33	1.6	117.551
3	SW model	3019.9	119.05	4.037	289.68
4	Mahapatra model	38225	318.54	8.5	613.678
5	SWM model	20687	156.94	5.923	493.8
6	**Proposed ISR model**	**18**	**-24.002**	**0.2**	66.057
Iteration 3.6.0					
1	JM model	76	7.6	0.333	**34.194**
2	GOI model	46721	359.39	6.6	613.8259
3	SW model	12089	367.83	17.89	156.9
4	Mahapatra model	11534	384.477	16.001	144.8
5	SWM model	34834.09	329.21	5.093	542.056

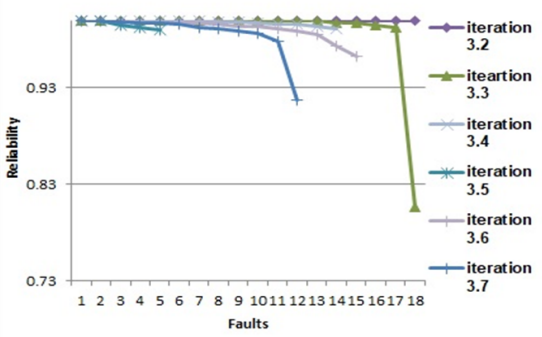

Figure 7. The plot of reliability versus number of faults for various iterations on JDT dataset

Figure 8. The plot of failure intensity versus faults for various iterations on JDT dataset

Table 6 continued

Sr. No.	model	SSE	MSE	AE	TS
6	**Proposed ISR model**	**192**	**21.3333**	**0.2**	44.45
Iteration 3.7.0					
1	JM model	280	45002	**0.071**	35.133
2	GOI model	90.011	9.001	0.166	37.21
3	SW model	200.9	11.08	0.189	49.057
4	Mahapatra model	952	**8.1368**	2.478	40.091
5	SWM model	1109.23	57.34	0.373	169.78
5	**Proposed ISR model**	**54**	9.0001	0.166	**25.5181**

Table 6 shows the Goodness-of-fit measures of the Proposed Iterative Software Reliability (ISR) model and other models. The iteration number 3.2 performs better than other models except in terms of AE compared to the GOI model. The Proposed Iterative Software Reliability (ISR) model is estimating values in a best fitted way than other models in iteration 3.3. In case of iteration 3.4 for proposed Iterative Software Reliability (ISR) model, AE value is comparatively higher than the Mahapatra model, but it performs well in other cases. For iteration number 3.5, the JM model beats the Proposed Iterative Software Reliability (ISR) model in terms of TS. In Iteration number 3.6, the proposed Iterative Software Reliability (ISR) model works well, compared to other models, except that it deviates well from the JM model at SSE, MSE, and TS values. For iteration number 3.7 proposed Iterative Software Reliability (ISR) model is fighting with all other models except at MSE and AE, where there is a bond in its performance with other models. Overall proposed Model is fitted well for estimating the parameters of the models and can be used for estimating the reliability of JDT systems.

The Proposed Iterative Software Reliability (ISR) model's reliability analysis using the JDT dataset is done in Figure 7. Failure intensity in Figure 8 is depicting the iteration's failure rate at each failure interval. There is a decreasing rate of failure at each iteration. In the initial iteration of JDT software, the failure rate decreases with low intensity; later on, when most bugs have been removed and almost all functionalities have been added, the iteration failure rate starts falling at high intensity. For example, in iteration numbers 3.6 and 3.7, there is a more decreasing failure rate than other iterations. This decreasing failure rate depicts that there is always an increase in end-user acceptance level and reliability with added functionalities and bug removal in each successive iteration. In the final iteration

release, the software is at its full implementation of functionalities and ready for conclusive acceptance from the end-users.

4.3.3. Statistical evaluation

From the above discussion, we concluded that the proposed ISR model outperformed all the other models for both datasets, Eclipse and JDT. To statistically evaluate the results, we have used a non – parametric Friedman test. It is used to evaluate whether statistical difference also exist among various models when applied over multiple iterations. For Friedman test, the following hypotheses are created and then the value of χ^2 is used to test the null hypothesis:

Null Hypothesis (H_0): There is no statistical difference between the performances of the compared models (JM model, GOI model), SW model, Mahapatra model, SWM model and the proposed ISR model). Alternative Hypothesis (H1): There exists statistical difference between the performances of the compared Models (JM model, GOI model), SW model, Mahapatra model, SWM model and the proposed ISR model). Friedman test was applied on all the performance measures for both the datasets. The *p*-values and the chi-square values given by Friedman test in shown in Table 7. From the Table 7, we can observe that for all the cases except one, the *p*-values proved that the results are significant at the 0.05 level of significance over 5 degrees of freedom. Since, the *p*-value is less than the significance level of 0.05 (shown in bold), the null hypothesis is rejected and alternate hypothesis is accepted. Thus, we say that there is statistical significant difference in the performance of the compared models. The ranks obtained by each model when using SSE, MSE, AE and TS are shown in Tables 8 and 9 for Eclipse and JDT datasets, respectively. The rank demonstrates the performance of all the models (lowest numerical rank value shows the highest performance). From Tables 8 and 9, we can observe that the top rank (shown in bold) is obtained by the proposed ISR model is all the cases for both the datasets. This demonstrates that the proposed ISR model is significantly better than the other compared models with respect to all the performance measures.

Table 7. Friedman results (*p*-value and χ^2 value)

	SSE		MSE		AE		TS	
Dataset	*p* value	χ^2	*p* value	χ^2	*p* value	χ^2	*p* value	χ^2
Eclipse	**0**	35.048	**0**	30.431	0.063	10.483	**0**	22.464
JDT	**0.001**	20	**0.013**	14.476	**0.003**	18.341	**0.004**	17.524

Table 8. Mean ranks obtained using Friedman test on Eclipse dataset

Models	SSE	MSE	AE	TS
JM model	5	5.04	4.29	4.79
GOI model	3.25	3.46	3.17	2.88
SW model	4.17	4.21	4.13	3.63
Mahapatra model	2.58	2.54	3.29	2.38
SWM model	4.75	4.33	3.92	4.92
Proposed ISR model	**1.25**	**1.42**	**2.21**	**2.42**

Table 9. Mean ranks obtained using Friedman test on JDT dataset

Models	SSE	MSE	AE	TS
JM model	2	2.67	2.08	2.33
GOI model	3.83	3.5	3.08	3.5
SW model	3.83	4.33	4.33	4.33
Mahapatra model	5	4.67	5.33	4.33
SWM model	5	4.5	4.5	5.17
Proposed ISR model	**1.33**	**1.33**	**1.67**	**1.33**

4.4. Discussion

The suitability of the model is shown on time-domain data sets. The Proposed Iterative Software Reliability (ISR) model is more adaptive to observed time-domain failure data sets than other failure rate models. Adaptation has been made possible because of the modulation parameter μ used in the model. As the software development moves towards completion, this parameter changes its values according to added functionalities and user acceptance level in each successive iteration of software development. This parameter assumes that there is added functionality and user acceptance in each iterative development cycle. According to the functionality added in each iteration, there is a change in the requirements of iterative software development. The users' varying needs are reflected with modulation factors in the Proposed Iterative Software Reliability (ISR) model. The Proposed Iterative Software Reliability (ISR) model provides a good fit for the observed failure data. Values estimated in Tables 3 and 5 shows acceptable parameter values at all weights γ. With each iteration, functionality values and user acceptance increase (demonstrated by the value of modulation parameter that increases from lower to higher). There is a corresponding change in needs for iteration development (shown by modulation factor). Values of modulation factor and modulation parameter change suitably with each upcoming iteration in both the failure datasets, demonstrating well the iterative software development behavior. Depending on the values of the estimated parameters, the number of remaining faults in terms of expectation is calculated. In some cases, the prediction deviates by a small amount due to the introduction and removal of mutually dependent and independent faults in each iteration. This verifies the imperfect debugging phenomenon associated with each iterative software development. The Goodness-of-fit for models is shown in Tables 4 and 6 regarding SSE, MSE, AE, and TS values. The Proposed Iterative Software Reliability (ISR) model fits well in both the failure datasets used. The Proposed Iterative Software Reliability (ISR) model has outstanding performance in data analysis of both the failure datasets than JM, GOI, SW, Mahapatra et al. and SWM models in terms of Goodness values. In all iterations of datasets, Proposed Iterative Software Reliability (ISR) model reliability increases as several iterations proceeds. The last iteration is assumed to be the reliable iteration. This change in reliability is showing a value-added software development process. All models have been implemented using a hybrid PSO-GSA algorithm for parameter estimation. The Proposed Iterative Software Reliability (ISR) model shows considerable performance with a hybrid algorithm even with more parameters than other models.

5. Threats to validity

In this section, we discuss potential threats which may affect the findings of the study. Conclusion validity threat in the proposed work is mitigated after evaluating the statistical viability of results with the use of two statistical tests. Researchers in this work have proposed a new Software reliability estimation model for reliability estimation of Eclipse and JDT software's. Proposed work has been compared with most relevant models available in the literature. Although no models is well fitted to all the failure datasets, each model is having its own assumptions and it works significantly well mainly for that type of dataset only for which it has been developed. Although other model comparison is done in the proposed research here for Eclipse and JDT software datasets and selection of most optimal model was not the primary aim of the study, this threat exists. However, to reach to a useful set of parameters, all the possible combinations of various parameters need to be explored. Parameter estimation plays an important role in the performance model development. Model parameters have been estimated and well pruned using hybrid swarm evolutionary algorithm. Another important factor which we considered was the use of stable performance metrics as Sum of Squared Errors, Mean Square Error, Absolute Error and Theil statistics to deal with the imbalanced nature of the datasets and models. Finally, the use of different datasets and models based on different assumptions also helps to reduce the conclusion validity threat. It may be noted that the models developed in the study can be used to detect software reliability at different iterations of software development. However, further studies should be conducted for analyzing the suitability of proposed model for determining change and acceptance levels at levels of iterations in the software development. This acts as a limitation to the applicability of the reported results. External validity threat concerned with the generalizability and replicability of the results is reduced in the proposed work due to the use of popular Tera Promise repository datasets used in this study which are freely available for the researchers to replicate the result, its findings and to conduct additional research. However more datasets belonging to different domains and applications need to be considered to ensure generalizability of the results. Further varied percentage of change-prone parameters of proposed model using hybrid swarm evolutionary algorithm across different datasets guarantees generalizability of the results. However, the study investigated parameter estimation using only hybrid swarm evolutionary algorithm only. Future experiments which involve more parameter estimation techniques likely using Machine learning or deep learning should be conducted to enhance the generalizability of the obtained results.

6. Conclusion and future scope

The majority of existing failure rate models are based on the classic waterfall software development lifecycle process model. However, new software development techniques, such as iterative life cycle processes, have been created and shown to be more effective than waterfall software development lifecycle processes. The Proposed Iterative Software Reliability (ISR) model makes use of the iterative nature of software development. The Proposed Iterative Software Reliability (ISR) model uses a modulation factor to represent the changing demands in each iteration. Software development features reduce at the end; accordingly modulation factor changes its value. Debugging is never perfect, so imperfect debugging has been incorporated by assuming fault introduction and removal. A realistic iterative software

development feature has been added to the Proposed Iterative Software Reliability (ISR) model perfectly and accurately. The Goodness-of-fit measure of the models is done using SSE, MSE, AE, and TS values. The Proposed Iterative Software Reliability (ISR) model fits all application datasets for each analysis and fulfils all the authentic assumptions during iterative software development. The proposed model has major beating SSE, MSE, AE, and TS values than JM, GO, SW, SWM, and Mahapatra et al. models when applied on both the datasets. According to various Goodness-of-fit measures, the proposed model is about 52% significant from other models in the case of Eclipse failure data analysis for iteration 1.0. It performs by about 30% using 2.0 to 2.1 and 3.0 to 3.6 iterations. Similarly proposed model has about 35% higher goodness than other used models. When the JDT software failure dataset is analyzed for Proposed Iterative Software Reliability (ISR) model fitness, the proposed model beats other failure rate models by about 55%. In each analysis done, the Proposed Iterative Software Reliability (ISR) model is found to be reaching acceptable performance and could be applied on other software failure datasets for further validation. Other reasonable criteria for finding Goodness-of-fit for the model can be developed, and a method for finding upper and lower bounds for estimation can be proposed in the future.

Data availability

Both the code for the algorithm and the datasets obtained after preprocessing are vailable in the replication package available at https://zenodo.org/records/13147825.

References

[1] S. Kumar and P. Ranjan, "A phase wise approach for fault identification," *Journal of Information and Optimization Sciences*, Vol. 39, No. 1, 2018, pp. 223–237.

[2] *IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990)*, IEEE Computer Society Std., 1990. [Online]. https://ieeexplore.ieee.org/document/159342

[3] P. Kapur, H. Pham, A.G. Aggarwal, and G. Kaur, "Two dimensional multi-release software reliability modeling and optimal release planning," *IEEE Transactions on Reliability*, Vol. 61, No. 3, 2012, pp. 758–768.

[4] C.Y. Huang and M.R. Lyu, "Optimal release time for software systems considering cost, testing-effort, and test efficiency," *IEEE Transactions on Reliability*, Vol. 54, No. 4, 2005, pp. 583–591.

[5] D. Greer and G. Ruhe, "Software release planning: An evolutionary and iterative approach," *Information and Software Technology*, Vol. 46, No. 4, 2004, pp. 243–253.

[6] 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.

[7] H. Pham, *System software reliability*. Springer Science and Business Media, 2007.

[8] K. Sahu and R. Srivastava, "Revisiting software reliability," *Data Management, Analytics and Innovation*, 2019, pp. 221–235.

[9] Z. Jelinski and P. Moranda, "Software reliability research," in *Statistical computer performance evaluation*. Elsevier, 1972, pp. 465–484.

[10] G.J. Schick and R.W. Wolverton, "An analysis of competing software reliability models," *IEEE Transactions on Software Engineering*, Vol. 2, 1978, pp. 104–120.

[11] J. Xavier, A. Macêdo, R. Matias, and L. Borges, "A survey on research in software reliability engineering in the last decade," in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, 2014, pp. 1190–1191.

[12] A.N. Sukert, "An investigation of software reliability models," in *Annual Reliability and Maintainability Symposium, Philadelphia, Pa*, 1977, pp. 478–484.
[13] P.B. Moranda, "An error detection model for application during software development," *IEEE Transactions on Reliability*, Vol. 30, No. 4, 1981, pp. 309–312.
[14] B. Littlewood and A. Sofer, "A Bayesian modification to the Jelinski–Moranda software reliability growth model," *Software Engineering Journal*, Vol. 2, No. 2, 1987, pp. 30–41.
[15] Z. Luo, P. Cao, G. Tang, and L. Wu, "A modification to the Jelinski-Moranda software reliability growth model based on cloud model theory," in *Seventh International Conference on Computational Intelligence and Security*. IEEE, 2011, pp. 195–198.
[16] G. Mahapatra and P. Roy, "Modified Jelinski–Moranda software reliability model with imperfect debugging phenomenon," *International Journal of Computer Applications*, Vol. 48, No. 18, 2012, pp. 38–46.
[17] 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.
[18] J.D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement," in *Proceedings of the 7th International Conference on Software Engineering*. Citeseer, 1984, pp. 230–238.
[19] C.Y. Huang, M.R. Lyu, and S.Y. Kuo, "A unified scheme of some nonhomogenous poisson process models for software reliability estimation," *IEEE Transactions on Software Engineering*, Vol. 29, No. 3, 2003, pp. 261–269.
[20] N. Pavlov, G. Spasov, A. Rahnev, and N. Kyurkchiev, "A new class of Gompertz-type software reliability models," *International Electronic Journal of Pure and Applied Mathematics*, Vol. 12, No. 1, 2018, pp. 43–57.
[21] P. Roy, G. Mahapatra, and K. Dey, "An NHPP software reliability growth model with imperfect debugging and error generation," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 21, No. 2, 2014, p. 1450008.
[22] V. Singh, M. Sharma, and H. Pham, "Entropy based software reliability analysis of multi-version open source software," *IEEE Transactions on Software Engineering*, Vol. 44, No. 12, 2017, pp. 1207–1223.
[23] X. Wei, Y. Dong, X. Li, and W.E. Wong, "Architecture-level hazard analysis using AADL," *Journal of Systems and Software*, Vol. 137, 2018, pp. 580–604.
[24] V.R. Basil and A.J. Turner, "Iterative enhancement: A practical technique for software development," *IEEE Transactions on Software Engineering*, Vol. 4, 1975, pp. 390–396.
[25] J. Erickson, K. Lyytinen, and K. Siau, "Agile modeling, agile software development, and extreme programming: The state of research," *Journal of Database Management*, Vol. 16, No. 4, 2005, pp. 88–100.
[26] C. Larman and V.R. Basili, "Iterative and incremental developments. A brief history," *Computer*, Vol. 36, No. 6, 2003, pp. 47–56.
[27] N. Kerzazi and F. Khomh, "Factors impacting software release engineering: A longitudinal study," in *2nd International Workshop on Release Engineering*, 2014, pp. 1–5.
[28] P. Kapur and R. Garg, "Optimal release policies for software systems with testing effort," *International Journal of Systems Science*, Vol. 22, No. 9, 1991, pp. 1563–1571.
[29] H. Pham and H. Pham, "Software reliability modeling," *System Software Reliability*, 2006, pp. 153–177.
[30] I.J. Myung, "Tutorial on maximum likelihood estimation," *Journal of Mathematical Psychology*, Vol. 47, No. 1, 2003, pp. 90–100.
[31] M.N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A comprehensive review of swarm optimization algorithms," *PLOS ONE*, Vol. 10, No. 5, 2015, p. e0122827.
[32] X.S. Yang, *Nature-inspired metaheuristic algorithms*. Luniver Press, 2010.
[33] 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, 2010, pp. 266–276.
[34] M. Xie, *Software reliability modelling*, Vol. 1. World Scientific, 1991.

[35] M. Ohba, "Inflection S-shaped software reliability growth model," in *Stochastic Models in Reliability Theory*. Springer, 1984, pp. 144–162.
[36] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on reliability*, Vol. 32, No. 5, 1983, pp. 475–484.
[37] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth models with testing-effort," *IEEE Transactions on Reliability*, Vol. 35, No. 1, 1986, pp. 19–23.
[38] S. Yamada, H. Ohtera, and M. Ohba, "Testing-domain dependent software reliability models," *Computers and Mathematics with Applications*, Vol. 24, No. 1–2, 1992, pp. 79–86.
[39] J. Xiang, F. Machida, K. Tadano, and Y. Maeno, "An imperfect fault coverage model with coverage of irrelevant components," *IEEE Transactions on Reliability*, Vol. 64, No. 1, 2014, pp. 320–332.
[40] P. Kapur and S. Younes, "Modelling an imperfect debugging phenomenon in software reliability," *Microelectronics Reliability*, Vol. 36, No. 5, 1996, pp. 645–650.
[41] B. Littlewood and J.L. Verrall, "A Bayesian reliability growth model for computer software," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, Vol. 22, No. 3, 1973, pp. 332–346.
[42] T. Mazzuchi and R. Soyer, "A Bayes empirical–Bayes model for software reliability," *IEEE Transactions on Reliability*, Vol. 37, No. 2, 1988, pp. 248–254.
[43] H. Pham, *Springer Handbook of Engineering Statistics*. Springer Nature, 2023.
[44] B. Littlewood, "Software reliability model for modular program structure," *IEEE Transactions on Reliability*, Vol. 28, No. 3, 1979, pp. 241–246.
[45] S. Yamada, K. Tokuno, and Y. Kasano, "Quantitative assessment models for software safety/reliability," *Electronics and Communications in Japan (*Part II*: Electronics)*, Vol. 81, No. 5, 1998, pp. 33–43.
[46] A.L. Goel and K. Okumoto, "A Markovian model for reliability and other performance measures of software systems," in *International Workshop on Managing Requirements Knowledge (MARK)*. IEEE, 1979, pp. 769–774.
[47] J. Shanthikumar, "A general software reliability model for performance prediction," *Microelectronics Reliability*, Vol. 21, No. 5, 1981, pp. 671–682.
[48] W.S. Jewell, "Bayesian extensions to a basic model of software reliability," *IEEE Transactions on Software Engineering*, Vol. 12, 1985, pp. 1465–1471.
[49] H. Joe and N. Reid, "On the software reliability models of Jelinski–Moranda and Littlewood," *IEEE transactions on Reliability*, Vol. 34, No. 3, 1985, pp. 216–218.
[50] T.F. Ho, W.C. Chan, and C.G. Chung, "A quantum modification to the Jelinski–Moranda software reliability model," in *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*. IEEE, 1990, pp. 339–342.
[51] P.J. Boland and H. Singh, "A birth-process approach to Moranda's geometric software-reliability model," *IEEE Transactions on Reliability*, Vol. 52, No. 2, 2003, pp. 168–174.
[52] L.I. Al Turk and E.G. Alsolami, "Jelinski–Moranda software reliability growth model: A brief literature and modification," *International Journal of Software Engineering and Applications*, Vol. 7, No. 2, 2016.
[53] N. Langberg and N.D. Singpurwalla, "A unification of some software reliability models," *SIAM Journal on Scientific and Statistical Computing*, Vol. 6, No. 3, 1985, pp. 781–790.
[54] Y. Lian, Y. Tang, and Y. Wang, "Objective Bayesian analysis of JM model in software reliability," *Computational Statistics and Data Analysis*, Vol. 109, 2017, pp. 199–214.
[55] A.G. Aggarwal, P. Kapur, and N. Nijhawan, "A discrete SRGM for multi-release software system with faults of different severity," *International Journal of Operational Research*, Vol. 32, No. 2, 2018, pp. 156–168.
[56] P. Erto, M. Giorgio, and A. Lepore, "The generalized inflection S-shaped software reliability growth model," *IEEE Transactions on Reliability*, Vol. 69, No. 1, 2018, pp. 228–244.
[57] D.H. Lee, I.H. Chang, H. Pham, and K.Y. Song, "A software reliability model considering the syntax error in uncertainty environment, optimal release time, and sensitivity analysis," *Applied Sciences*, Vol. 8, No. 9, 2018, p. 1483.

[58] A.G. Aggarwal, V. Dhaka, N. Nijhawan, and A. Tandon, "Reliability growth analysis for multi-release open source software systems with change point," *System Performance and Management Analytics*, 2019, pp. 125–137.
[59] R. Gupta, M. Jain, and A. Jain, "Software reliability growth model in distributed environment subject to debugging time lag," *Performance Prediction and Analytics of Fuzzy, Reliability and Queuing Models: Theory and Applications*, 2019, pp. 105–118.
[60] M. Asraful Haque and N. Ahmad, "A logistic growth model for software reliability estimation considering uncertain factors," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 28, No. 5, 2021, p. 2150032.
[61] W.D. van Driel, J. Bikker, and M. Tijink, "Prediction of software reliability," *Microelectronics Reliability*, Vol. 119, 2021, p. 114074.
[62] H. Okamura and T. Dohi, "Application of EM algorithm to NHPP-based software reliability assessment with generalized failure count data," *Mathematics*, Vol. 9, No. 9, 2021, p. 985.
[63] K. Sahu and R. Srivastava, "Needs and importance of reliability prediction: An industrial perspective," *Information Sciences Letters*, Vol. 9, No. 1, 2020, pp. 33–37.
[64] K. Kumaresan and P. Ganeshkumar, "Software reliability prediction model with realistic assumption using time series (S) ARIMA model," *Journal of Ambient Intelligence and Humanized Computing*, Vol. 11, 2020, pp. 5561–5568.
[65] K. Sharma, M. Bala et al., "A quantitative testing effort estimate for reliability assessment of multi release open source software systems," *Journal of Computational and Theoretical Nanoscience*, Vol. 16, No. 12, 2019, pp. 5089–5098.
[66] S. Malik, K. Sharma, and M. Bala, "Reliability analysis and modeling of green computing based software systems," *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, Vol. 14, No. 4, 2021, pp. 1060–1071.
[67] Sangeeta, Sitender, K. Sharma, and M. Bala, "New failure rate model for iterative software development life cycle process," *Automated Software Engineering*, Vol. 28, No. 2, 2021, p. 9.
[68] X. Liu and N. Xie, "Grey-based approach for estimating software reliability under nonhomogeneous Poisson process," *Journal of Systems Engineering and Electronics*, Vol. 33, No. 2, 2022, pp. 360–369.
[69] B.V. Devi and R.K. Devi, "Software reliability models based on machine learning techniques: A review," in *AIP Conference Proceedings*, Vol. 2463. AIP Publishing, 2022.
[70] V. Verma, S. Anand, and A.G. Aggarwal, "Optimal time for management review during testing process: An approach using S-curve two-dimensional software reliability growth model," *International Journal of Quality and Reliability Management*, 2023.
[71] I. Ramadan, H.M. Harb, H. Mousa, and M. Malhat, "Assessment reliability for open-source software using probabilistic models and marine predators algorithm," *International Journal of Computers and Information*, Vol. 10, No. 1, 2023, pp. 18–35.
[72] V. Pradhan, A. Kumar, and J. Dhar, "Modelling software reliability growth through generalized inflection S-shaped fault reduction factor and optimal release time," *Proceedings of the Institution of Mechanical Engineers, em Part O: Journal of Risk and Reliability*, Vol. 236, No. 1, 2022, pp. 18–36.
[73] V. Pradhan, A. Kumar, and J. Dhar, "Emerging trends and future directions in software reliability growth modeling," *Engineering Reliability and Risk Assessment*, 2023, pp. 131–144.
[74] V. Pradhan, J. Dhar, and A. Kumar, "Testing coverage-based software reliability growth model considering uncertainty of operating environment," *Systems Engineering*, 2023.
[75] J.E. Dennis Jr and R.B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.
[76] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN '95 – International Conference on Neural Networks*, Vol. 4. IEEE, 1995, pp. 1942–1948.
[77] J.H. Holland, "Genetic algorithms," *Scientific American*, Vol. 267, No. 1, 1992, pp. 66–73.
[78] A. Sheta and J. Al-Salt, "Parameter estimation of software reliability growth models by particle swarm optimization," *Management*, Vol. 7, 2007, p. 14.
[79] R. Malhotra and A. Negi, "Reliability modeling using particle swarm optimization," *International Journal of System Assurance Engineering and Management*, Vol. 4, 2013, pp. 275–283.

[80] C. Jin and S.W. Jin, "Parameter optimization of software reliability growth model with S-shaped testing-effort function using improved swarm intelligent optimization," *Applied Soft Computing*, Vol. 40, 2016, pp. 283–291.
[81] S. Mirjalili and S.Z.M. Hashim, "A new hybrid PSOGSA algorithm for function optimization," in *International Conference on Computer and Information Application*. IEEE, 2010, pp. 374–377.
[82] A. Abraham, R.K. Jatoth, and A. Rajasekhar, "Hybrid differential artificial bee colony algorithm," *Journal of Computational and Theoretical Nanoscience*, Vol. 9, No. 2, 2012, pp. 249–257.
[83] S. Mirjalili, G.G. Wang, and L.d.S. Coelho, "Binary optimization using hybrid particle swarm optimization and gravitational search algorithm," *Neural Computing and Applications*, Vol. 25, 2014, pp. 1423–1435.
[84] F. Liu and Z. Zhou, "An improved QPSO algorithm and its application in the high-dimensional complex problems," *Chemometrics and Intelligent Laboratory Systems*, Vol. 132, 2014, pp. 82–90.
[85] Y. Li, Y. Wang, and B. Li, "A hybrid artificial bee colony assisted differential evolution algorithm for optimal reactive power flow," *International Journal of Electrical Power and Energy Systems*, Vol. 52, 2013, pp. 25–33.
[86] S.S. Jadon, R. Tiwari, H. Sharma, and J.C. Bansal, "Hybrid artificial bee colony algorithm with differential evolution," *Applied Soft Computing*, Vol. 58, 2017, pp. 11–24.
[87] K. Sharma, M. Bala et al., "Magnetic navigation based optimizer: A new optimization algorithm for software reliability model parameter estimation," *Journal of Advanced Research in Dynamical and Control Systems*, 2018, pp. 1957–1968.
[88] A.K. Tripathi, K. Sharma, and M. Bala, "Military dog based optimizer and its application to fake review," *arXiv preprint arXiv:1909.11890*, 2019.
[89] K. Sharma, M. Bala et al., "An ecological space based hybrid swarm-evolutionary algorithm for software reliability model parameter estimation," *International Journal of System Assurance Engineering and Management*, Vol. 11, No. 1, 2020, pp. 77–92.
[90] A. Sharma, R. Chaturvedi, S. Kumar, and U.K. Dwivedi, "Multi-level image thresholding based on Kapur and Tsallis entropy using firefly algorithm," *Journal of Interdisciplinary Mathematics*, Vol. 23, No. 2, 2020, pp. 563–571.
[91] M.S. Khan, F. Jabeen, S. Ghouzali, Z. Rehman, S. Naz et al., "Metaheuristic algorithms in optimizing deep neural network model for software effort estimation," *IEEE Access*, Vol. 9, 2021, pp. 60 309–60 327.
[92] S. Kassaymeh, S. Abdullah, M. Al-Laham, M. Alweshah, M.A. Al-Betar et al., "Salp swarm optimizer for modeling software reliability prediction problems," *Neural Processing Letters*, Vol. 53, 2021, pp. 4451–4487.
[93] K. Lakra and A. Chug, "Application of metaheuristic techniques in software quality prediction: A systematic mapping study," *International Journal of Intelligent Engineering Informatics*, Vol. 9, No. 4, 2021, pp. 355–399.
[94] L. Raamesh, S. Jothi, and S. Radhika, "Enhancing software reliability and fault detection using hybrid brainstorm optimization-based LSTM model," *IETE Journal of Research*, 2022, pp. 1–15.
[95] P. Dhavakumar and N. Gopalan, "An efficient parameter optimization of software reliability growth model by using chaotic grey wolf optimization algorithm," *Journal of Ambient Intelligence and Humanized Computing*, Vol. 12, 2021, pp. 3177–3188.
[96] S. Singh, A. Ashok, M. Kumar, and T.K. Rawat, "Adaptive infinite impulse response system identification using teacher learner based optimization algorithm," *Applied Intelligence*, Vol. 49, 2019, pp. 1785–1802.
[97] Rakhi and G.L. Pahuja, "Solving reliability redundancy allocation problem using grey wolf optimization algorithm," *Journal of Physics: Conference Series*, Vol. 1706, 2020, p. 012155.
[98] A. Kaushik, D.K. Tayal, and K. Yadav, "The role of neural networks and metaheuristics in agile software development effort estimation," in *Research anthology on artificial neural network applications*. IGI Global, 2022, pp. 306–328.

[99] N. Yadav and V. Yadav, "Software reliability prediction and optimization using machine learning algorithms: A review," *Journal of Integrated Science and Technology*, Vol. 11, No. 1, 2023, p. 457.

Authors and affiliations

Sangeeta
e-mail: sangeeta@msit.in
ORCID: https://orcid.org/0000-0002-8691-3892
Department of Computer Science and Engineering,
Maharaja Surajmal Institute of Technology, India

Sitender
e-mail: sitender@msit.in
ORCID: https://orcid.org/0000-0003-0341-2927
Department of Information Technology,
Maharaja Surajmal Institute of Technology, India

Rachna Jain
e-mail: rachnajain@bpitindia.com
Department of Information Technology,
Bhagwan Parshuram Institute of Technology, India

Ankita Bansal
e-mail: ankita.bansal06@gmail.com
Netaji Subhas University of Technology, India