Global Journal of Modeling and Intelligent Computing (GJMIC)

Vol. 3, No.1, January, 2023 ISSN: 2767-1917

OPTIMAL RELEASE TIME FOR MODULAR SOFTWARE IN RANDOM FIELD ENVIRONMENT

Ritu Gupta, T A Pai Management Institute, India (ritu.gupta@manipal.edu)
Sanghita P. Nath, Amity University, Noida, India (sanghitapnath@gmail.com)
M. Jain, Indian Institute of Technology Roorkee, India(madhu.jain@ma.iitr.ac.in)

ABSTRACT

We propose a software reliability growth model (SRGM) to estimate the reliability of a large and complicated modular system. With the aim of making the fault detection and correction procedures suitable to real-world circumstances, the influence of a random field environment (RFE) along with imperfect debugging is considered. The software reliability in a field environment typically differs from the reliability prediction made during the testing phase and even from all its comparable relevance in other domains. We determine optimal release time (ORT)policy based on cost-reliability constraint. The policy aims to minimize the expected total maintenance cost by including various costs. The proposed SRGM may provide a valuable tool for software developers to predict the reliability of a complex modular system and make informed decisions about ORT.

Keywords: SRGM, Imperfect debugging, Random field environment, Optimal release time.

1. INTRODUCTION

Software has emerged as a fundamental component of modern technology and is utilized in several industries, including transportation, banking, and healthcare, etc. With the increasing reliance on software, the need for reliable software systems is more crucial than ever before. Consequently, the increased complexity of software systems has caused the emergence of software faults and failures during the development of these large software systems. Software failures can have disastrous consequences including equipment loss, financial losses, and even human fatalities in case of safety-critical systems. Hence, software reliability and its stability are key concerns, leading to the development of reliability engineering context. To avoid software failures and ensure high reliability, testers perform software testing. During the testing phase, the estimation of software reliability is done using a suitable software reliability growth model (SRGM). SRGM allows developers to make predictions about the reliability of their software during testing and can even aid them in identifying areas where more testing or debugging may be required to enhance the software's quality. SRGMs can also assist developers in estimating the time and materials required to reach a desired degree of reliability.

Numerous SRGMs were suggested over the last four decades to assess the software reliability. Software development is more than just about the skills and methods used by the software developers. Even the most skilled software engineers cannot produce software products that are fully error-free due to the complex concepts and logical frameworks of software development. Despite comprehensive testing, software errors can still occur, and it is critical to spot and correct them as soon as possible to prevent adverse effects. Debugging activity helps to identify the cause of a software failure, determining the problematic component of the program, and carrying out the necessary actions to correct the software fault. In a perfect debugging process, all faults in a software system are eliminated. In case of imperfect debugging, some faults may remain undetected, and the number of faults may increase or decrease in relation to the debugging time as a result of new fault introduction. Many of the earlier NHPP based SRGMs assume the perfect debugging process (cf. Jelinski and Moranda; 1972). Goel and Okumoto (1979) suggested a reliability growth model for perfect debugging environment by following the assumption that software faults were immediately removed after detection, resulting in simulations that were very close to real software reliability engineering. However, due to the human details involved in debugging a software program, perfect debugging is an unrealistic assumption. Goel (1985) was the first to acknowledge the concept of imperfect debugging. Jones (1996) discovered that in most cases, fault removal efficiency is less than 100%. The fault removal factor (FRF) reveals a major part to get improved software reliability by addressing the challenge of failure incidence (cf. Jain et al.; 2012). Zhu et al. (2016) proposed a SRGM with a time-dependent fault detection and removal rates. A comparative analysis between the SRGM performances for perfect and imperfect debugging has given by Gupta et al. (2021). Other recent and significant advancements of SRGMs with imperfect debugging environment that have shown excellent performance and superior to other models were studied by Chatterjee et al. (2021) and Zhang et al. (2022).

The software in field environment and in testing environment may be considered as identical reason being both exhibit the same failure-occurrence behavior. Nevertheless, this supposition may not be correct, since the software is utilized in a wide diversity of field of applications when it is launched whereas the in-house testing environment is a controlled setting with much less variance than the field environment. To characterize this difference based on conditions of usage in different environment, a specific proportional constant of the environmental factor is introduced. The first researchers to investigate this impact of the field environment factor was Teng and Pham (2004). Some more notable contributions towards estimation of software reliability in random field environment are due to Inoue and Yamada (2011), Sgarbossa and Pham (2010), Zhao et al. (2006), Pham (2014), Chang et al. (2014), Li and Pham (2017), Pradhan et al. (2020), Mishra et al. (2023) and many others.

In view of management of software industry, the significant usage of SRGM is to identify the ideal point of deliverance. The testing methods used, and the time allotted for testing determine how well a software system performs. Greater testing time results in software that is more reliable, but testing time also costs more money. As a result, several researchers have developed models for software release time. Many models prioritize either cost or reliability when determining the moment at which software delivery should begin. Pachauri et al. (2013) proposed an optimal software cost model with cost-reliability criteria under fuzzy concept. In the literature, the proportion of delay in removal of detected faults at any time has been introduced as delay effect factor. The delay in software releasing is not ideal for manufacturers as well as for customers. Gupta et al. (2019) proposed an optimal cost model considering delay effect. Anand (2020) proposed a cost optimization model that took two-dimensional delayed S-shaped SRGM into consideration. Verma et al. (2022) considered error generation, and time-dependent fault reduction factor to propose an optimal release policy. In a more recent proposal, an optimal release policy considering the exponential testing coverage function was proposed by Kumar et al. (2023).

In the real time software system, the failure pattern depends on the specific software module. There have been some studies on the reliability estimation of software systems using modules or components. The quantitative evaluation of component-based software systems was introduced by Popstojanova and Trivedi (2001). By merging the program architecture with the component and interface failure patterns, they analytically calculated the software reliability. The SRGM should be designed in such a way that it considers the behavior of all the software modules since the reliability of the entire piece of software depends on the reliability of each individual module/component. In this context, Jain and Gupta (2011) obtained optimum release time for modular software system incorporating testing effort function. The model took into consideration the faults of various kinds which are different as their severity levels. Furthermore in 2018, they suggested a modular software system reliability considering fault reduction factor.
In the proposed study, we build a module-based software reliability growth model assuming users use the software system in an uncertain environment. We study both fault detections and corrections in imperfect debugging along a random field environment. The rest of paper is structured as follows. Section 2 outlines the modular SRGM in a random field environment. Section 3 discusses the optimal release time problem, incorporating maintenance cost analysis. The model simulation is done in section 4 to validate the analytical findings. The paper concludes in section 5.

2. THE MODULAR SRGM

The proposed model focuses on a software system comprising N modules, each with distinct testing requirements and failure characteristics. Our approach utilizes a SRGM that assumes varying initial fault counts for each module.
Assumptions:

- **(i)** The fault detection and correction phenomena are governed by non-homogeneous Poisson process.
- **(ii)** Whenever a software error is detected, immediate action is taken to address it.
- **(iii)** The fault detection rate $b_n(t)$ is multiplied by η, anon-negative random variable which is consideredto reveal uncertainty of operating environment, where $n = 1,2,3, \ldots, N$.

(iv) Each identified fault in the software is considered independent of the others.

(v) The expected number of detected faults is proportional to the number of remaining undetected faults and the proportionality follows an inflection S-shaped time dependent function.

(vi) The expected number of corrected faults is proportional to the average number of faults that are identified but still not fixed and this proportionality alsofollows an inflection S-shaped time dependent function.

(vii) The total fault content function is assumed as linearly time dependent function as case of imperfect debugging wherein new faults may be introduced.

Notations:

$m_{d,n}(t)/m_{c,n}(t)$: Number of detected/corrected faults in n^{th} modulein $(0,t)$time interval.

$b_n(t)$:	Fault detection/correction rate in the n^{th}moduleat any time t.
α:	Fault introduction rate in imperfect debugging environment, where $0 \le \alpha \le 1$.
p_n:	Probability of successful removal of faults from the n^{th}module;$0 \le p_n \le 1\ and \alpha \le p_n$.
$a_n(t)$:	Fault content function in the n^{th} moduleat any time t.
$g(\eta)$:	Probability density functionof gamma distribution.
$G(x)$:	Laplace-transform of gamma distribution.
β:	Shape parameter of the learning curve.
λ/θ:	Scale/Shape parameter of gamma distribution function of random field environment (RFE), η.

Governing Equations:

With the aforementioned assumptions in mind, the following differential equations can be used to represent the mean value function (MVF) for number of fault detections and corrections in$n^{th}(n = 1,2,\ldots N)$ module under the NHPP model with random field environment as given below:

$$\frac{d}{dt}m_{d,n}(t) = \eta b_n(t)\left[a_n(t) - p_n m_{d,n}(t)\right] \tag{1}$$

$$\frac{d}{dt}m_{c,n}(t) = p_n b_n(t)\left[a_n(t) - m_{c,n}(t)\right] \tag{2}$$

where,

$$b_n(t) = \frac{b_n}{1+\beta e^{-b_n t}}, \qquad b_n > 0, 0 < \beta < 1 \tag{3}$$

$$\frac{d}{dt}a_n(t) = \alpha \frac{d}{dt}m_{d,n}(t) \tag{4}$$

along with initial settings$m_{d,n}(0) = 0, m_{c,n}(0) = 0,$ and$a_n(0) = a$. The random field environmentwith T, the time of time of software release is shown in figure 1 and η can be defined as;

$$\eta = \begin{cases} 1, & t \le T \\ r.v.\ with\ pdf\ g(\eta), & t \ge T \end{cases}$$

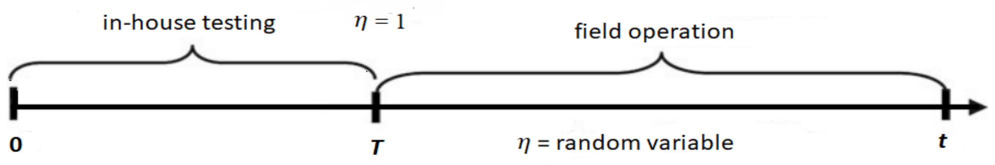


Figure 1. Random field environment

2.1: FDM (Fault Detection Model)

We solve the equation (1) for testing and operational phases under the initial settings and yield MVF for the detection process in n^{th}module;

(i). In-house testing phase $(t \leq T)$

$$m_{d,n}(t) = \frac{a_n}{(p_n - \alpha)}\left[1 - e^{-\int_0^t (p_n-\alpha) b_n(\tau)d\tau}\right] \tag{5}$$

(ii). Field operationalphase $(t \geq T)$

$$m_{d,n}(t) = \frac{a_n}{(p_n - \alpha)}\left[1 - e^{-\int_0^T (p_n-\alpha) b_n(\tau)d\tau} G((p_n - \alpha)\int_T^t b_n(\tau)d\tau)\right] \tag{6}$$

On solving equations (5-6) and using equation (3), the MVF for the detection process in n^{th}module are given as follows:

$$m_{d,n}(t) = \frac{a_n}{(p_n-\alpha)}\left[1 - \left(\frac{1+\beta}{\beta+e^{b_n t}}\right)^{(p_n-\alpha)}\right]; \quad \text{for } (t \leq T) \tag{7}$$

$$m_{d,n}(t) = \frac{a_n}{(p_n-\alpha)}\left[1 - \left(\frac{1+\beta}{\beta+e^{b_n T}}\right)^{(p_n-\alpha)}\left(\frac{\lambda}{\lambda+(p_n-\alpha)\log\left(\frac{\beta+e^{b_n t}}{\beta+e^{b_n T}}\right)}\right)^{\theta}\right]; \quad \text{for} \tag{8}$$

The total mean number of faults detected for both the testing and operational phase from each of the modules can be determined as

$$m_d(t) = \sum_{n=1}^{N} m_{d,n}(t)$$

$$= \begin{cases} \sum_{n=1}^{N} \frac{a_n}{(p_n - \alpha)}\left[1 - \left(\frac{1+\beta}{\beta + e^{b_n t}}\right)^{(p_n-\alpha)}\right] & , t \leq T \\ \sum_{n=1}^{N} \frac{a_n}{(p_n - \alpha)}\left[1 - \left(\frac{1+\beta}{\beta + e^{b_n T}}\right)^{(p_n-\alpha)}\left(\frac{\lambda}{\lambda + (p_n - \alpha)\log\left(\frac{\beta+e^{b_n t}}{\beta+e^{b_n T}}\right)}\right)^{\theta}\right] & , t \geq T \end{cases} \tag{9}$$

2.2: FCM (Fault Correction Model)

From equations (1) and (2), we obtain

$$m_{c,n}(t) = p_n m_{d,n}(t) \tag{10}$$

Now, the MVF for the correction process inn^{th}module is given as follows:

(i). In-house testing phase $(t \leq T)$

$$m_{c,n}(t) = \frac{p_n a_n}{(p_n - \alpha)}\left[1 - \left(\frac{1+\beta}{\beta + e^{b_n t}}\right)^{(p_n-\alpha)}\right] \tag{11}$$

(ii). Field operationalphase $(t \geq T)$

$$m_{c,n}(t) = \frac{p_n a_n}{(p_n - \alpha)}\left[1 - \left(\frac{1+\beta}{\beta + e^{b_n T}}\right)^{(p_n-\alpha)}\left(\frac{\lambda}{\lambda + (p_n - \alpha)\log\left(\frac{\beta+e^{b_n t}}{\beta+e^{b_n T}}\right)}\right)^{\theta}\right] \tag{12}$$

Therefore, the total mean number of faults corrected for both the testing and operational phase from each of the modules can be evaluated as follows:

$$m_c(t) = \sum_{n=1}^{N} m_{c,n}(t)$$

$$= \begin{cases} \sum_{n=1}^{N} \frac{p_n a_n}{(p_n-\alpha)}\left[1-\left(\frac{1+\beta}{\beta+e^{b_n t}}\right)^{(p_n-\alpha)}\right] & ,t \le T \\ \sum_{n=1}^{N} \frac{p_n a_n}{(p_n-\alpha)}\left[1-\left(\frac{1+\beta}{\beta+e^{b_n T}}\right)^{(p_n-\alpha)}\left(\frac{\lambda}{\lambda+(p_n-\alpha)\log\left(\frac{\beta+e^{b_n t}}{\beta+e^{b_n T}}\right)}\right)^{\theta}\right] & ,t \ge T \end{cases} \tag{13}$$

3. OPTIMAL RELEASE TIME POLICY

The testing and maintenance phases are crucial for ensuring the quality of a software system. Delays in releasing software can result in financial losses, and releasing software with faults can raise maintenance costs throughout the operating phase. The software must be reliable when it is delivered to clients with minimum cost. Some faults are not always readily recognized during human testing, hence automated techniques are used to find these faults and reach a specified quality level in a given period while controlling testing costs.
We find the optimumlaunching time of software at a minimum maintenance cost with desired and satisfactory level of software reliability. To achieve this, we propose a cost model that considers va4ious costs such as testing cost (C_3), debuggingfault correction costs before/after (C_1/C_2)releasing and the risk cost (C_4)on software module failure.

The reliability function for the time interval $(T, T+x]$is given as;

$$R(x/T) = e^{-[m_d(T+x)-m_d(T)]} \tag{14}$$

The total expected software maintenance cost function is constructed as:

$$EC(T) = C_1 m_d(T) + C_2[m_d(T_c) - m_d(T)] + C_3 T + C_4[1 - R(x/T)] \tag{15}$$

where, $T_c$is software life cycle length.
Now, using equation (9) and equation (13) in equation (15) respectively, we get the total expected maintenance cost as;
(i). In-house testing phase $(t \le T)$

$$\begin{aligned} E_1C(T) = (C_1 - C_2)\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{\beta+1}{\beta+e^{b_n T}}\right)^{(p_n-\alpha)}\right] \\ + C_2\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{\beta+1}{\beta+e^{b_n T_c}}\right)^{(p_n-\alpha)}\right] + C_3 T \\ + C_4\left[1-e^{-\left[\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{1+\beta}{\beta+e^{(b_n)(T+x)}}\right)^{(p_n-\alpha)}-\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{1+\beta}{\beta+e^{b_n T}}\right)^{(p_n-\alpha)}\right]\right]\right]}\right] \end{aligned} \tag{16}$$

(ii). Field operationalphase $(t \ge T)$

$$\begin{aligned} E_2C(T) = (C_1 - C_2)\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{\beta+1}{\beta+e^{b_n T}}\right)^{(p_n-\alpha)}\right] \\ + C_2\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{\beta+1}{\beta+e^{b_n T_c}}\right)^{(p_n-\alpha)}\left(\frac{\lambda}{\lambda+(p_n-\alpha)\log\left(\frac{\beta+e^{b_n T_c}}{\beta+e^{b_n T}}\right)}\right)^{\theta}\right] + C_3 T \\ + C_4\left[1-e^{-\left[\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{1+\beta}{\beta+e^{(b_n)(T+x)}}\right)^{(p_n-\alpha)}-\sum_{n=1}^{N}\frac{a_n}{(p_n-\alpha)}\left[1-\left(\frac{1+\beta}{\beta+e^{b_n T}}\right)^{(p_n-\alpha)}\right]\right]\right]}\right] \end{aligned} \tag{17}$$

4. NUMERICAL SIMULATION

A numerical illustration is presented to show the precision of the analytical results derived for the three modular software systemsto observe the effect of different factors on the anticipated maintenance cost during testing phase and software reliability.To determine ORT with a specified level of reliability, the parameters are set as follows: $x = 0.05$, $\alpha = 0.01$, $\beta = 0.1, \lambda = 0.61, \Theta = 1, T_c = 30, p_1 = 0.1$, $p_2 = 0.2$, $p_3 = 0.3, (a_1, a_2, a_3) = (10, 50, 50), (b_1, b_2, b_3) = (0.01, 0.2, 0.2)$. Here, we have parameters p_1, p_2 and p_3 denoting the probability of successful removalof faults from 1^{st}, 2^{nd}and3^{rd}software module respectively; a_1, a_2 and a_3 denoteinitial faults presented in 1^{st}, 2^{nd}and3^{rd}software module respectively; and b_1, b_2 and b_3 denote the fault detection/correction rate in 1^{st}, 2^{nd}and3^{rd}software module respectively. The numerical results are summarized in figures 2 - 7 and in table 1.The following are the cost parameters selected for the various sets:

Set I: C_1=320\$; C_2=400\$; C_3=50\$; C_4=3000\$.
Set II: C_1=330\$; C_2=400\$; C_3=50\$; C_4=3000\$.
Set III: C_1=320\$; C_2=415\$; C_3=50\$; C_4=3000\$.
Set IV: C_1=320\$; C_2=400\$; C_3=55\$; C_4=3000\$.
Set V: C_1=320\$; C_2=400\$; C_3=50\$; C_4=4000\$.

Figure 2(i-iii) and figure 3(i-iii) depict MVF of testing and operational phase on increasing time by varying the parameters p_1, p_2, p_3 respectively. We see that MVF for both testing and field operation phases increases rapidly with increase in time t in the beginning and then attains almost constant value with further increase in t. Figures 2(i) and 3(i) reflect the effect of probability$p_1$on MVF of testing and operational phases. It is noticed that MVF for both the cases decreases with respect to p_1. In this context, the MVF for both phases decrease on increasing parameter p_2 which could be observed in figures 2(ii) and 3(ii). In figures 2(iii) and 3(iii), a decreasing pattern of MVF can be seen on increasing probabilityp_3; however, the impact of p_3 in operational phase is less significant in comparison to the impact of p_1 and p_2.

Figures 4-7 reveal the impacts of parametersa_1, a_2, b_2 and b_3 on R(x|T) and maintenance costof testing phase (E_1CT) by varying testing time (T). The figures 4(i), 5(i), 6(i) and 7(i) demonstrate a sharp increase in software reliability around initial testing time span[0,80], then it attains its desired level of reliability. In the figures 4(ii), 5(ii), 6(ii) and 7(ii), we observe that E_1CT steadily decreases as testing time increases until it reaches its optimumand after that, it significantly growths and becomes asymptotically constant.

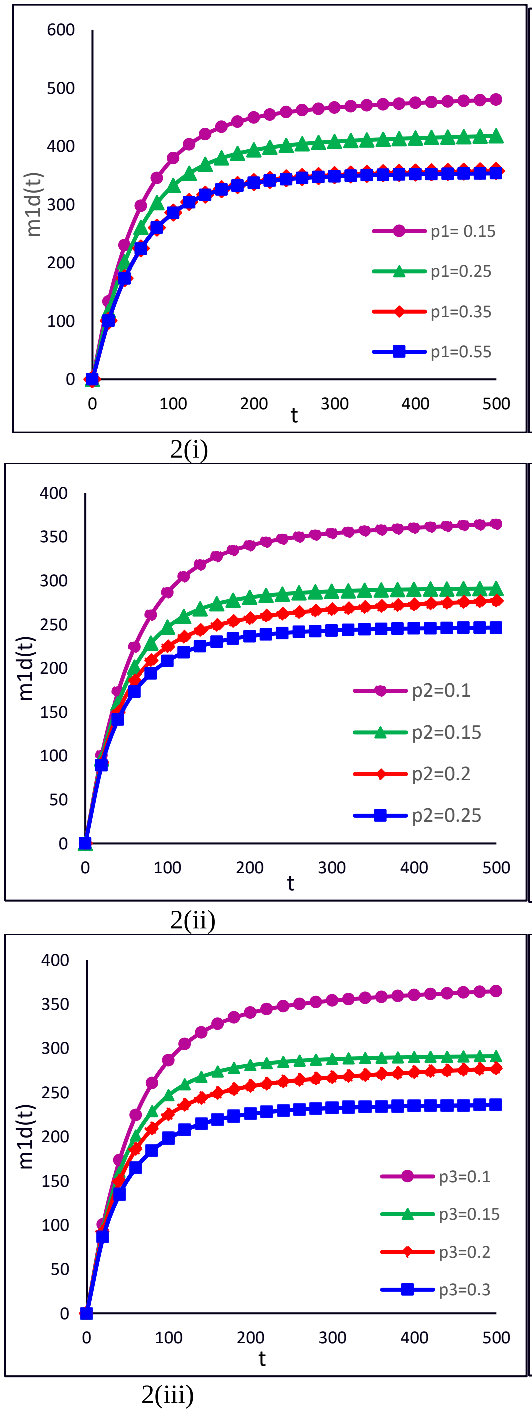
Figures 4(i-ii) illustrate the impact of reliability and expected maintenance cost in the testing phase (E_1CT) with respect to testing time, respectively by varying the parameter a_1. We observe the effect ofa_1 on R(x|T) in figure 4(i) which seems negligible till approximately $T = 60$, after which the reliability decreases on further increasing a_1. It is entirely normal for reliability to automatically decline as there are more faults. In figure 4(ii), E_1CT increases on increasing a_1, however, the impact of a_1 on the cost appears to be relatively less significant.

In figure 5(i), R(x|T) decreases on increasing a_2 initially till about $T = 100$ after which a_2 has minimum effect on R(x|T) and converge nearly to the same value. E_1CT increases with the increase ina_2 prominently which could be seen in figure 5(ii).

In figures 6(i) and 7(i), it appears that the fault detection rate in module 2 (b_2)and module 3(b_3), has relatively minor impact on R(x|T). But, on increasing b_2 and b_3, the overall maintenance cost increases significantly which can be visualized in figures 6(ii) and 7(ii).
A summary of the calculated ORT (*T**) along with minimal cost (*E_1CT**) and the reliability attained(*R(x/T)**) is provided in table 1.

Table 1. Optimal release time for their respective *E_1CT** and *R(x/T)**

Cost Set	*T**	*E_1CT** (in thousands)	*R(x/T)**
I	83.1	89.84977084	0.97102057
II	80.5	102.7174418	0.96831494
III	88.6	96.76123521	0.97588308
IV	81.3	98.83318178	0.96917814
V	84.5	98.44892177	0.97236315



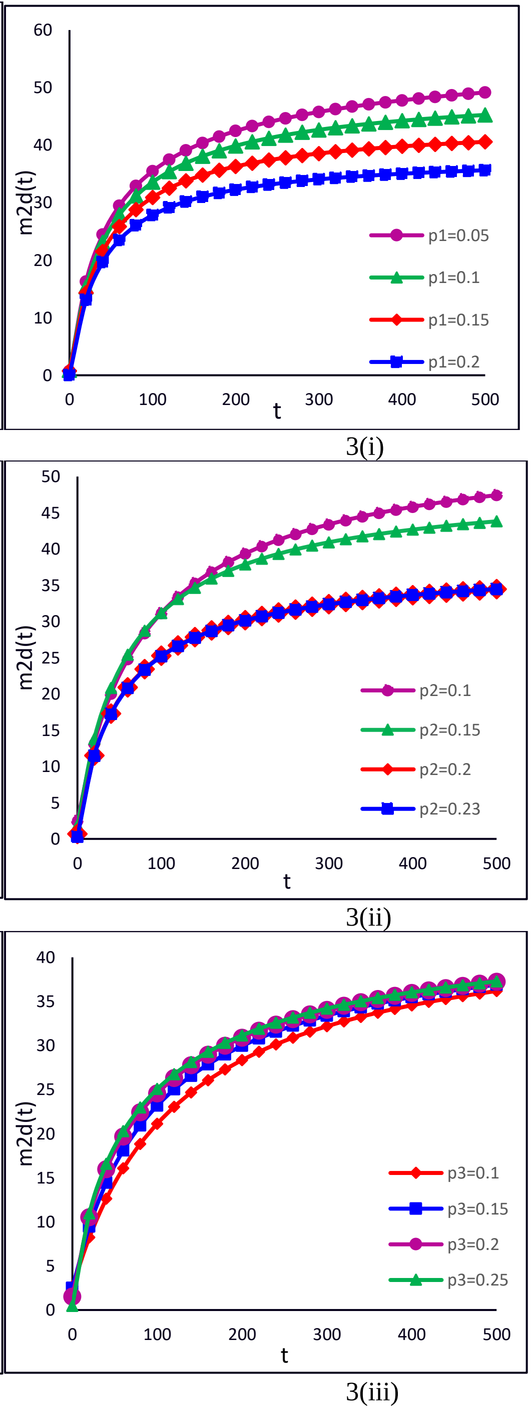
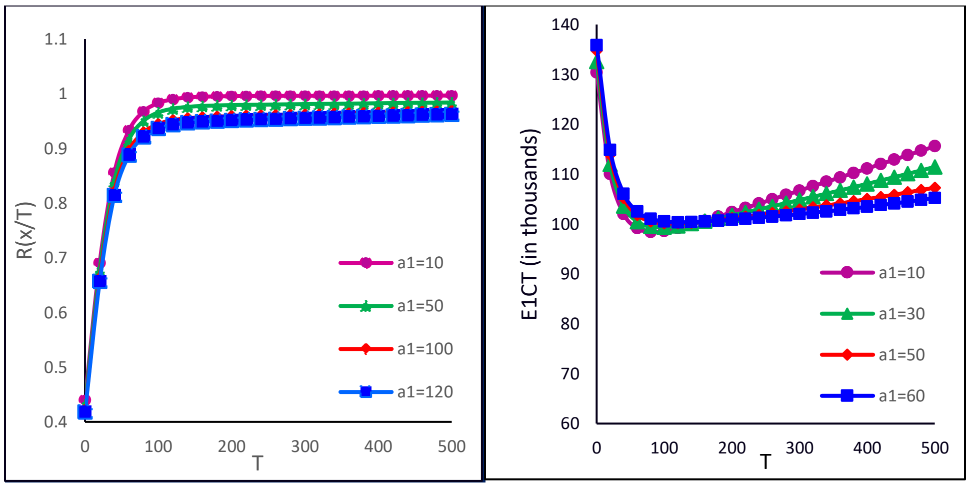


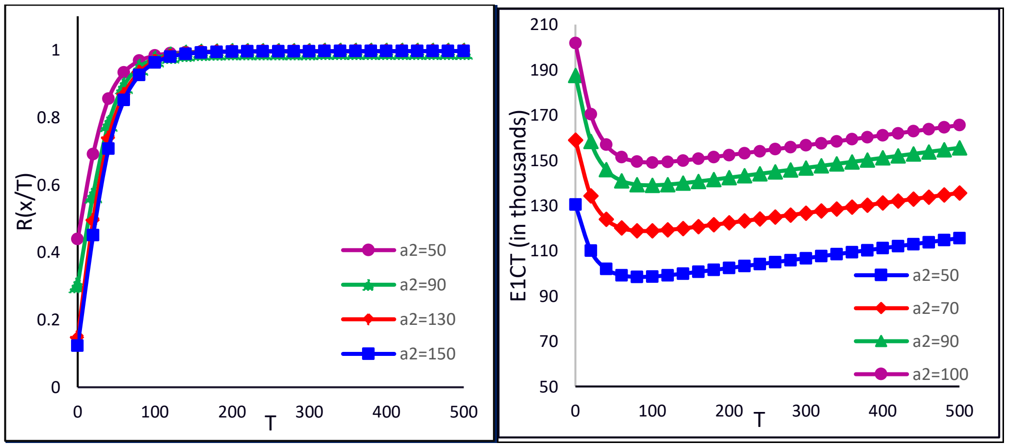
Figure 2: MVF of testing phase $(t \leq T)$for (i) p_1 (ii) p_2 (iii) p_3

Figure 3: MVF of operational phase $(t \geq T)$for (i) p_1 (ii) p_2 (iii)p_3



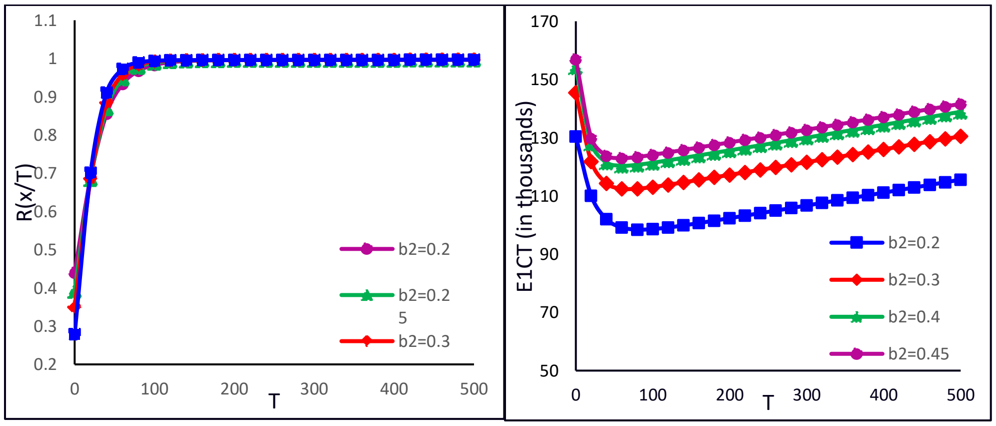
4(i) 4(ii)

Figure 4: Reliability and expected total maintenance cost vs testing time by varying a_1



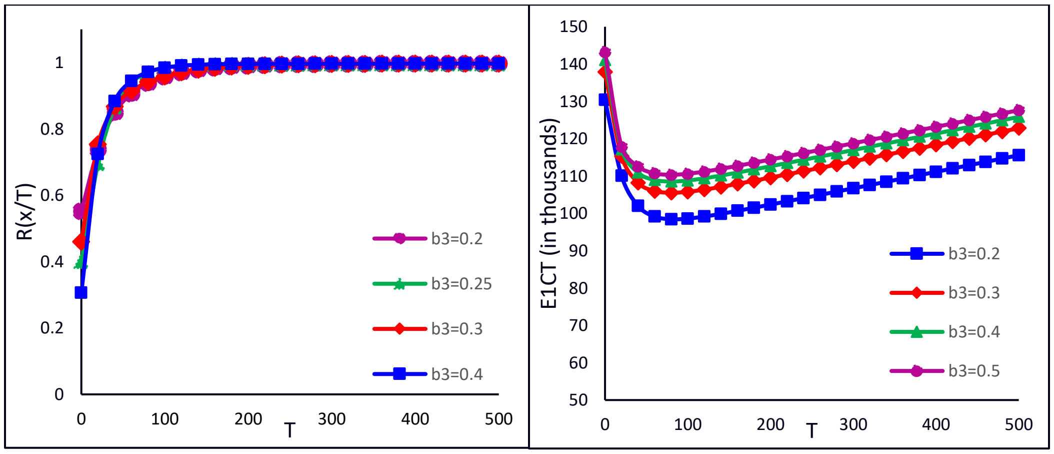
5(i) 5(ii)

Figure 5: Reliability and expected total maintenance cost vs testing time by varying a_2



6(i) 6(ii)

Figure 6: Reliability and expected total maintenance cost vs testing time by varying b_2



7(i) 7(ii)

Figure 7: Reliability and expected total maintenance cost vs testing time by varying b_3

Due to the complexity of the mathematical derivations, the numerical experiment presented in this study takes into account the scenario of two modules. The findings, even with this constrained scenario, have significant implications. The following inferences can be made in light of the numerical experiment's findings:

- As the probability of fault removal from the software module increases, there is a decrease in the number of detected faults for both in-house testing and the field operational phases.
- The total maintenance costs can be decreased by lowering the initial number of faults for the three modular software systems, whereas a rise in the fault detections leads to an earlier attainment of optimality at a lower testing time.
- A reduction in the values of a_1 and a_2 yields an improvement in software reliability.
- The impact of fault detection rates of modules 2 and 3 on the reliability is insignificant, but they result in increased maintenance costs.

5. CONCLUSIONS

In the present investigation, we have developed FDM & FCM models for module based SRGM that exhibits the ORT in a random field environment, assuming the RFE to be a gamma distribution function. Moreover, we have considered fault detection and correction rate to be inflection S-shaped function. To show the numerical analysis, a three modular system was chosen, and we have performed a numerical simulation using MATLAB to verify the analytical results. The suggested release times for testing, software reliability can be used to quickly spot any potential faults in the implementation and design of the software, allowing for their correction prior to release and guaranteeing that the software satisfies its reliability requirements at the lowest possible cost. This SRGM could prove to be an invaluable tool for software designers and system analysts to forecast the reliability of a complex modular software and arrive at knowledgeable determination about the optimal release time.

REFERENCES

Chang, I. H., Pham, H., Lee, S. W., & Song, K. Y. (2014). A testing-coverage software reliability model with the uncertainty of operating environments. *International Journal of Systems Science: Operations & Logistics*, 1(4), 220–227. https://doi.org/10.1080/23302674.2014.970244.

Chatterjee, S., Saha, D., & Sharma, A. (2021). Multi-upgradation software reliability growth model with dependency of faults under change point and imperfect debugging. *Journal of Software: Evolution and Process*, 33(6). https://doi.org/10.1002/smr.2344.

Goel, A. L. (1985). Software Reliability Models: Assumptions, Limitations, and Applicability. *IEEE Transactions on Software Engineering*, SE-11(12), 1411–1423. https://doi.org/10.1109/TSE.1985.232177.

Goel, A. L., & Okumoto, K. (1979). Time-dependent error-detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, R-28(3), 206–211. https://doi.org/10.1109/TR.1979.5220566.

Goševa-Popstojanova, K., & Trivedi, K. S. (2001). Architecture-based approach to reliability assessment of software systems. *Performance Evaluation*, 45(2–3), 179–204. https://doi.org/10.1016/S0166-5316(01)00034-7.

Gupta, R., Jain, M., & Jain, A. (2019). Software reliability growth model in distributed environment subject to debugging time lag (pp. 105–118). https://doi.org/10.1007/978-981-13-0857-4_7.

Gupta, R., Jain, M., Yadav, A., Chooramani, A., Kumar, S. (2021). Software reliability growth model in perfect and imperfect debugging environment. *Global Journal of Computer and Engineering Technology (GJCET)*, Vol. 1, No. 2, 25-33.

Inoue, S., & Yamada, S. (2011). Software reliability growth modeling frameworks with change of testing-environment. *International Journal of Reliability, Quality and Safety Engineering*, 18(04), 365–376. https://doi.org/10.1142/S0218539311004299.

Jain, M., & Gupta, R. (2011). Optimal release policy of module-based software. *Quality Technology & Quantitative Management*, 8(2), 147–165. https://doi.org/10.1080/16843703.2011.11673253.

Jain, M., Jain, A., & Gupta, R. (2018). Analysis of module-based software reliability growth model incorporating imperfect debugging and fault reduction factor (pp. 69–80). https://doi.org/10.1007/978-981-10-5577-5_6.

Jain, M., Manjula, T., & Gulati, T. R. (2012). Software reliability growth model (SRGM) with imperfect debugging, fault reduction factor and multiple change-point (pp. 1027–1037). https://doi.org/10.1007/978-81-322-0491-6_95.

Jelinski, Z., & Moranda, P. (1972). Software Reliability Research in Statistical Computer Performance Evaluation (pp. 465–484). *Elsevier*. https://doi.org/10.1016/B978-0-12-266950-7.50028-1.

Jones, C. (1996). New directions in software management. *Information Systems Management Group*, 5(12), 152–162.

Kumar, S., Aggarwal, A. G., & Gupta, R. (2023). Modeling the role of testing coverage in the software reliability assessment. *International Journal of Mathematical, Engineering and Management Sciences*, 8(3), 504–513. https://doi.org/10.33889/IJMEMS.2023.8.3.028.

Li, Q., & Pham, H. (2017). NHPP software reliability model considering the uncertainty of operating environments with imperfect debugging and testing coverage. *Applied Mathematical Modelling*, 51, 68–85. https://doi.org/10.1016/j.apm.2017.06.034.

Mishra, G., Kapur, P. K., & Aggarwal, A. G. (2023). A generalized multi-upgradation SRGM considering uncertainty of random field operating environments. *International Journal of System Assurance Engineering and Management*, *14*(S1), 328–336. https://doi.org/10.1007/s13198-023-01859-7.

Pachauri, B., Kumar, A., & Dhar, J. (2013). Modeling optimal release policy under fuzzy paradigm in imperfect debugging environment. *Information and Software Technology*, 55(11), 1974–1980. https://doi.org/10.1016/j.infsof.2013.06.001.

Pham, H. (2014). A new software reliability model with Vtub-shaped fault-detection rate and the uncertainty of operating environments. *Optimization*, 63(10), 1481–1490. https://doi.org/10.1080/02331934.2013.854787.

Pradhan, V., Dhar, J., Kumar, A., & Bhargava, A. (2020). An S-shaped fault detection and correction SRGM subject to gamma-distributed random field environment and release time optimization (pp. 285–300). https://doi.org/10.1007/978-981-15-3643-4_22.

Sgarbossa, F., & Pham, H. (2010). A cost analysis of systems subject to random field environments and reliability. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(4), 429–437. https://doi.org/10.1109/TSMCC.2010.2042713.

Teng, X. & Pham, H. (2004). A software cost model for quantifying the gain with considerations of random field environments. *IEEE Transactions on Computers*, 53(3), 380–384. https://doi.org/10.1109/TC.2004.1261844.

Verma, V., & Anand, S. (2020). Two-dimensional release policy for software systems incorporating FRF, opportunity cost and environment factor (pp. 879–885). https://doi.org/10.1007/978-981-15-1420-3_95.

Verma, V., Anand, S., Kapur, P. K., & Aggarwal, A. G. (2022). 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*, 13(5), 2429–2441. https://doi.org/10.1007/s13198-022-01653-x.

Zhang, C., Lv, W.G., Sheng, S., Wang, J.Y., Su, J.Y., & Meng, F.C. (2022). Default detection rate-dependent software reliability model with imperfect debugging. *Applied Sciences*, 12(21), 10736. https://doi.org/10.3390/app122110736.

Zhao, J., Liu, H.W., Cui, G., & Yang, X.Z. (2006). Software reliability growth model with change-point and environmental function. *Journal of Systems and Software*, 79(11), 1578–1587. https://doi.org/10.1016/j.jss.2006.02.030.

Zhu, M., & Pham, H. (2016). A software reliability model with time-dependent fault detection and fault removal. *Vietnam Journal of Computer Science*, 3(2), 71–79. https://doi.org/10.1007/s40595-016-0058-0.