Global Journal of Computer and Engineering Technology (GJCET)

Vol. 1, No. 2, July, 2021 ISSN: 2767-1933

SOFTWARE RELIABILITY GROWTH MODEL IN PERFECT AND IMPERFECT DEBUGGING ENVIRONMENT

Ritu Gupta, AIAS, Amity University, India (rgupta@amity.edu)
Madhu Jain, Indian Institute of Technology, Roorkee, India (madhu.jain@ma.iitr.ac.in)
Akshay Yadav, Indian Institute of Technology, Roorkee, India (ayadav@ma.iitr.ac.in)
Akshita Chooramani, AIAS, Amity University, India (achooramani@gmail.com)
Sudeep Kumar, AIAS, Amity University, India (sudeep.kumar2@s.amity.edu)

ABSTRACT

The stability or life of a software system with different capacities is referred to as software reliability. The software quality is the most important consideration while designing a software system. Software quality is determined by a variety of criteria, including software reliability, efficiency, testing abilities and cost considerations. In this paper, we deal with two different SRGMs (software reliability growth models) based on NHPP (Non-homogeneous Poisson process). We develop the models for inflection S-shaped context by considering time independent fault content factor for perfect debugging environment whereas another model differs in an imperfect debugging environment by involving time dependent total fault content factor. We suppose that once a software problem is identified (removed), immediate debugging begins, and that either the total number of faults is detected (removed) with probability $p_1(p_2)$ or the total number of faults remains constant with probability $(1-p_1)((1-p_2))$, i.e., p_1 % of the faults can be identified successfully and p_2 % of the faults may be removed as well during the software testing phase. The intended SRGMs' results are used to assess the software's reliability. We further compare the performance of both the software reliability growth models. The findings show that the model can perform better in terms of fitting and prediction.

Keywords: Software reliability, Perfect and imperfect debugging, Mean value function, Non-homogeneous Poisson process.

1. INTRODUCTION

Measuring and controlling the quality of the software before advertising are the key concerns for software developer and industry. Accordingly, software reliability growth models play a significant role in measuring satisfactory characterization of the software program and allocating testing resources. Over the past decades, several software reliability growth models have been purposed to estimate software reliability. Musa et al. (1987) studied software reliability measurement, prediction and applications. There are numerous reasons behind the growing importance of software program reliability. SRGMs (software reliability growth models) are models that attempt to forecast software reliability in phrases of the quantity of faults in the software program. Some SRGMs were evolved during the last thirty years as trendy elegance of properly advanced stochastic technique model in reliability engineering. Non-homogeneous Poisson Process (NHPP) has been successfully utilized to investigate software program reliability difficulties. For software program reliability estimation, Huang et al. (2003) proposed a unified scheme of Non-homogeneous Poisson systems. They are specifically handy to describe failure tactics which own satisfied tendencies such as the growth and deterioration of reliability. There are basically three varieties of SRGM: 1. Exponential SRGM 2. Delayed S- shaped SRGM 3. Inflection S- formed SRGM. An exponential SRGM absorbs a situation while a couple of failures are assigned to fault. Delayed S-shaped SRGM is formed to provide an explanation for the phenomenon of removing mistakes. An Inflection S-formed SRGM incorporates the mutual dependency of errors by way of modifying the logistic growth curve model. Kapur et al. (2006) categorized of faults in software and analyzed for the process of removal. When software program repair is imperfect, Boland et al. (2007) advocated top-of-the-line release times. They confirmed a generalized logistic function which defines the actual consumption of resources at some stage in development of software program. In figuring out the measuring enhancements, the time-duration of the software development process, attaining effect and debugging time-table, the software reliability version becomes easier. Lin et al. (2008) proposed a model which enhanced the predictive capabilities of software reliability growth models. This helps to assess the software program efficiently and to observe the changes that take vicinity in software every now and then. Chiu et al. (2008) studied of software program reliab ility increase model from the angle of gaining knowledge of results. Xiao et al. (2011) estimated software reliability using extreme valued distributions which shows relevance in communications in computers and information science. Chang et al. (2014) proposed a software reliability model incorporating operating environment uncertainty. Wang et al. (2016) gave optimized method for software reliability model based on NHPP. Byun (2017) discussed reliability prediction based on test duration. Mirchandani (2018) described a methodology that examines a software system throughout its life cycle and demonstrates the software system's software reliability as a function of its operational use and incremental changes. Malik et al. (2019) looked into how reliability modelling may be done at various stages of software development and offered reliability measuring tools and methodologies utilized in the modern software business.

Debugging is the method of finding and resolving defects or troubles within a program that save correct operation of software program or a system. There are two kinds of debugging - imperfect debugging and perfect debugging. In imperfect debugging, faults are not continually absolutely eliminated from a software or machine and while in perfect debugging, faults are totally removed from a software or system and no new faults are introduced, new faults can be provided as part of the fault removal process. In ideal debugging it is widely considered that faults are revised with composure which might be accountable for software failures and there are no new faults produced. Most of the earlier software reliability models assume the fault elimination manner to be ideal i.e. while an intention is made to remove a fault, it's far eliminated with actuality and there are no new faults produced or added. Software failure fee is taken to be proportional to the residual wide variety of bugs, and every worm has a constant failure fee contribution. Pham et al. (1999) proposed an S-shaped fault detection rate for a general imperfect software debugging model. The software operational reliability prediction was done by Huang et al. (2005). Perfect debugging is an impractical assumption because of the human detail associated in debugging a software program. The phenomenon of imperfect debugging takes vicinity because of introduction of latest faults all through the correction manner and additionally faults corrected are less than the faults responsible for failures. If a number of the detected faults are not eliminated with actuality or new faults tested during debugging manner then it's far known as imperfect debugging. There are type of imperfect debugging opportunities-first, on a failure the corresponding fault is recognized , but simply because of partial expertise of the software program, the detected fault isn't pulled out completely and subsequently the fault content material of the software program stays unaltered on the removal action, referred to as imperfect fault debugging, secondly, when a failure occurs, the relevant fault is identified and corrected in real time, however the removal process introduces additional flaws into the software program. This form of imperfect debugging resulted in an increase in the software's fault content, often known as mistakes generation. The Pham-Nordmann-Zhang (PNZ) model, which is often used in NHPP SRGMs, includes the imperfect debugging phenomenon by assuming that errors can be generated during the debugging section. This version assumes that the fault detection fee is a non-decreasing time-structured feature with an inflection S-Shaped model, and that the fault creation charge is a linear feature that is time-dependent. In imperfect fault removal debugging, detected faults aren't evacuated completely without bearing of new faults. The best distinction among imperfect fault removal and errors technology is that in imperfect fault elimination, while the software is generating errors, the fault information isn't always modified of the software program will increase as trying out progresses. But the mistake generation is mostly taken into consideration in imperfect debugging fashions in preference to imperfect fault removal. Goel (1985) has initially made recognized the idea of imperfect debugging. Jones (1996) has detected that usually fault elimination performance may be smaller than 100%. A NHPP software reliability growth model with imprecise debugging and error creation was proposed by Roy et al. (2014). A successive failure in a software program is indicated by using a fault or a bug inside the hardware that could motive the software program to malfunction. The most important metric for estimating software quality and reliability is fault prediction. To assess the software fault, a variety of approaches, metrics, aspects, and testing methodologies are available. Before the software is launched to the market, faults are identified and eradicated all through the lengthy – testing duration. On the basis of the remark of product disasters, faults and failures are estimated. The fault removal factor (FRF) performs a enormous role in overcoming the difficulty of failure incidence to enhance the increase of reliability of the software program. Zhu et al. (2016) suggested a software reliability model with a fault detection and removal method that is time dependent. Factors like fault removal and the severity of detection of faults; fault elimination performance (FRE) is every other thing which impacts the reliability of the software. It is a time consuming and a complicated method. Recent and notable developments of SRGMs with imperfect debugging environment have been discussed by Chaudhary et al. (2020) and Chatterjee et al. (2021).

2. MODEL DESCRIPTION

We provide two software reliability growth models in this section that may be helpful for software designers and developers for improving software quality in techno-economic constraint. We develop the two SRGM model in perfect and imperfect debugging environments by assuming an inflection S-shaped type fault detection and removal processes. The following are the assumptions for the suggested model:-

- The fault detection and removal processes are followed by the non-homogeneous Poisson process.
- At any point throughout the testing period, the rate of software fault detection and removal is proportional to the present fault content
- At any given time, the software failure rate is a function of the fault detection rates and the number of faults still present in the software
- The failure in the software system might occur because of the remaining errors.
- With probability p_1 (p_2), the total number of defects is lowered by one; with probability ($1- p_1$)(($1- p_2$)), the total number of faults remains the same. i.e, p_1 % of the faults can be detected successfully and p_2 % of the faults can be removed as well during the software testing phase, $0< p_1,p_2 <1$; $p_{1\neq} p_2$.

Following are the notations which are used for modelling purpose.

$m_d(t)$= expected number of faults detected in time interval (0, t]		
$m_r(t)$= expected number of faults removed in time interval (0, t]		
$r(t)$ = fault detection rate function		
$b(t)$= fault reduction factor		
$a(t)$= total number of faults i.e. fault content function		
a	=	initial number of faults
r	=	proportionality constant (detection rate)
α	=	shape parameter
β	=	scale parameter
ξ	=	constant fault introduction rate
x	=	mission time of the software
T	=	testing time of the software

2.1 Software Reliability Growth Model-1 (SRGM-1)

A SRGM with inflection S-shaped FRF is proposed in which we establish the mean value function which helps us to evaluate the total number of fault (of different types) removed in perfect debugging environment. The differential equations for the mean number of faults detected and removed are now constructed. These faults are treated in steps, beginning with detection and ending with removal. The following differential equations can be used to express the mean value function (MVF) for the NHPP model for fault detection and fault removal.

$$\frac{\mathrm{d}}{\mathrm{d}t}m_d(t) = r(t)[a - p_1 m_d(t)] \tag{1}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}m_r(t) = r(t)[m_d(t) - p_2 m_r(t)] \tag{2}$$

where

$$r(t) = rb(t) \tag{3}$$

$$b(t) = \frac{\alpha}{1+\beta e^{-\alpha t}} \tag{4}$$

The fault detection and fault removal processes are depicted in equations (1) and (2), respectively. Under the following initial conditions, solve the differential equations (1) and (2).

$$m_d(t) = 0 = m_r(t) \text{ when } t = 0, \tag{5}$$

We get

$$m_d(t) = \frac{a}{p_1}\left(1 - \left(\frac{1+\beta}{e^{\alpha t}+\beta}\right)^{rp_1}\right) \tag{6}$$

and,

$$m_r(t) = \frac{a}{p_1}\left(\frac{1}{p_1} - \frac{1}{(p_2-p_1)}\left(\frac{1+\beta}{e^{\alpha t}+\beta}\right)^{rp_1}\right) + \frac{a}{p_2(p_2-p_1)}\left(\frac{1+\beta}{e^{\alpha t}+\beta}\right)^{rp_2} \tag{7}$$

2.2 Software Reliability Growth Model-2 (SRGM-2)

This model is an extension of the above model in which we will establish the mean value function which helps us to evaluate total number of fault removed in imperfect debugging environment.
The differential equations for the mean number of faults detected and the number of faults removed are now constructed. These faults are treated in steps, beginning with detection and ending with removal. The following differential equations can be used to express the mean value function (MVF) for the NHPP model for fault detection and fault removal.

$$\frac{\mathrm{d}}{\mathrm{d}t} m_d(t) = r(t)[a(t) - p_1 m_d(t)] \tag{8}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} m_r(t) = r(t)[m_d(t) - p_2 m_r(t)] \tag{9}$$

where

$$a(t) = a + \xi m(t) \tag{10}$$

$$r(t) = rb(t) \tag{11}$$

$$b(t) = \frac{\alpha}{1+\beta e^{-\alpha t}} \tag{12}$$

Equations (8) and (9) show fault detection and fault removal processes, respectively. On solving these equations under the initial conditions,

$$m_d(t) = 0 = m_r(t) \text{ when } t = 0 \tag{13}$$

We obtain

$$m_d(t) = \frac{a}{p_1-\xi}\left(1 - \left(\frac{1+\beta}{e^{\alpha t}+\beta}\right)^{r(p_1-\xi)}\right) \tag{14}$$

$$m_r(t) = a\left(\frac{1}{p_2(p_1-\xi)} - \frac{1}{(p_2-p_1+\xi)}\left(\frac{1+\beta}{e^{\alpha t}+\beta}\right)^{r(p_1-\xi)}\right) + a(1+\beta)^{rp_2}\left(\frac{1}{p_2(p_1-\xi)} - \left(\frac{1}{p_2-p_1+\xi}\right)\right) \tag{15}$$

3. SOFTWARE RELIABILITY

In this section, we yield software reliability for both SRGMs. Software reliability can be defined as the likelihood of successful software performance over a certain time interval $(x, x+\Delta x)$ under certain conditions.
The software reliability can be given as follows:

$$R(T) = exp[-m_r(T+x) + m_r(T)] \tag{16}$$

3.1 Software Reliability of SRGM-1

Using equation (16) the software reliability for model 1 is given as:

$$R(T) = exp\left[-\left\{\frac{a}{p_1}\left(\frac{1}{p_2} - \frac{1}{(p_2-p_1)}\left(\frac{1+\beta}{e^{\alpha(T+x)}+\beta}\right)^{rp_1}\right) + \frac{a}{p_2(p_2-p_1)}\left(\frac{1+\beta}{e^{\alpha(T+x)}+\beta}\right)^{rp_2}\right\} + \frac{a}{p_1}\left(\frac{1}{p_2} - \frac{1}{(p_2-p_1)}\left(\frac{1+\beta}{e^{\alpha T}+\beta}\right)^{rp_1}\right) + \frac{a}{p_2(p_2-p_1)}\left(\frac{1+\beta}{e^{\alpha T}+\beta}\right)^{rp_2}\right] \tag{17}$$

3.2 Software Reliability of SRGM-2

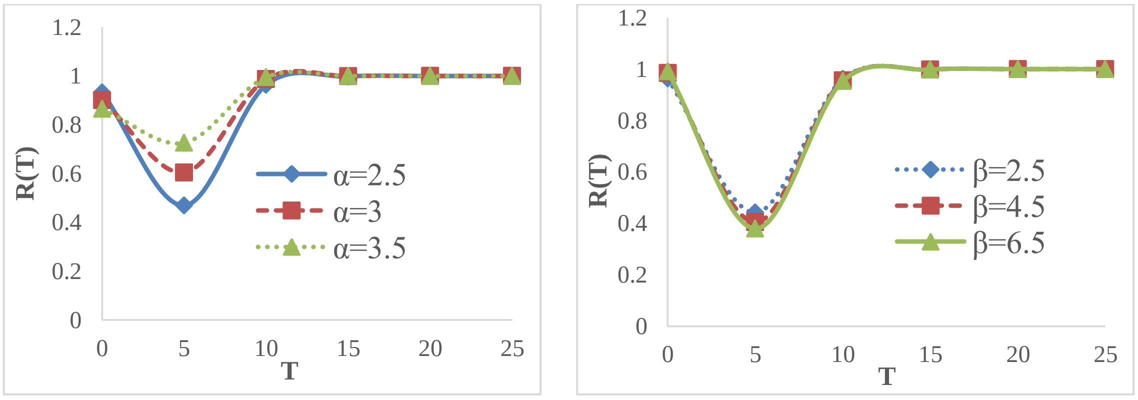
Using equation (16) the software reliability for model 2 is given as:

$$R(T) = exp\left[-a\left(\frac{1}{p_2(p_1-\xi)} - \frac{1}{(p_2-p_1+\xi)}\left(\frac{1+\beta}{e^{\alpha(T+x)}+\beta}\right)^{r(p_1-\xi)}\right) - a(1+\beta)^{rp_2}\left(\frac{1}{p_2(p_1-\xi)} - \left(\frac{1}{p_2-p_1+\xi}\right)\right) + a\left(\frac{1}{p_2(p_1-\xi)} - \frac{1}{(p_2-p_1+\xi)}\left(\frac{1+\beta}{e^{\alpha T}+\beta}\right)^{r(p_1-\xi)}\right) + a(1+\beta)^{rp_2}\left(\frac{1}{p_2(p_1-\xi)} - \left(\frac{1}{p_2-p_1+\xi}\right)\right)\right] \tag{18}$$

4. SENSITIVITY ANALYSIS

In this section, we use a numerical example to demonstrate software reliability for both models, allowing us to test the validity of the proposed models in both perfect and imperfect debugging conditions. To investigate the impact of different parameters on software reliability, the results are summarized in figures 1-3 and figures 4-6 by varying testing time for perfect and imperfect debugging environment, respectively. We set the default parameters as follows: α=2.5, β=2.5, a=50, r=0.6, p1=0.9, p2=0.5, ξ=0.05, *x*=0.1.

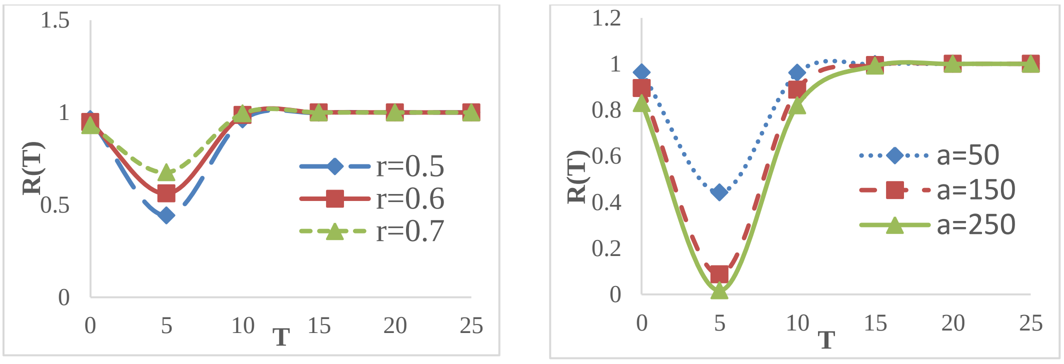
Figure 1 (i-ii) illustrates the software reliability by varying T for different values of α and β, respectively. It is cleared that in perfect debugging environment, initially the software reliability decreases for lower values of T and then it increases gradually then it becomes almost constant and reaches to desired level of reliability.



(i) (ii)

Figure 1: Software reliability v/s testing time in perfect debugging environment by varying (i) α and (ii) β

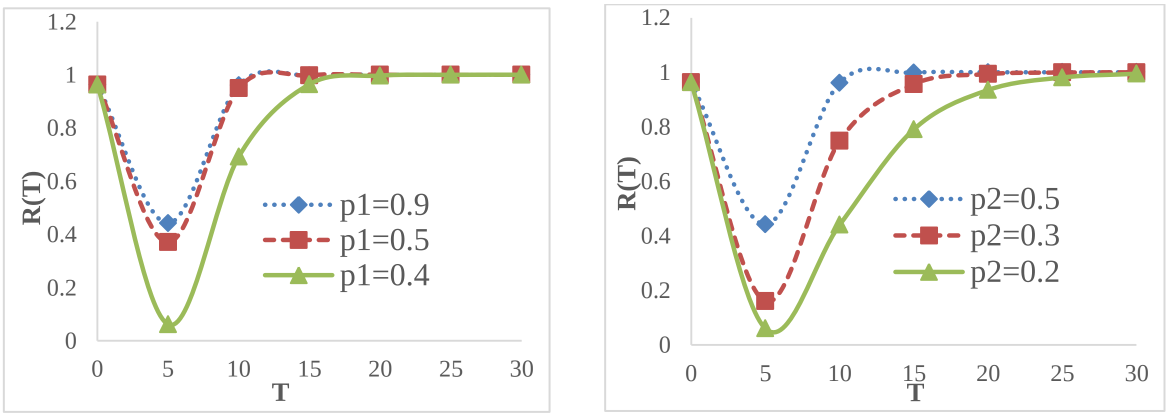
It is also noticed that with the increase in parameter (α), the reliability increases whereas with the increase in scale parameter (β), the software reliability decreases significantly.



(i) (ii)

Figure 2: Software reliability v/s testing time in perfect debugging environment by varying (i) a and (ii) r

Figure 2(i) shows the software reliability by changing the testing time for various initial fault counts. It is cleared that with the increase in parameter (a) i.e. the initial no. of faults in perfect debugging environment, the software reliability decreases. Figure 2(ii) shows the software reliability as a function of testing time (T) for various fault detection rates. We've noticed a downward trend in software reliability in perfect debugging environment with respect to the fault detection rate.



(i) (ii)

Figure 3: Software reliability v/s testing time in perfect debugging environment by varying (i) p_1 and (ii) p_2

In figure 3(i-ii), We demonstrate software reliability by varying (T) for various p_1 and $p_2$values. From figs, similar pattern of software reliability can be found with respect to testing time as we have seen in figures 1 and 2. It is also observed that, software reliability increases for increasing values of p_1 and p_2 in figure 3(i-ii).

Figures 4-6 depict the software reliability for SRGM-2 by varying testing time T in imperfect debugging environment The software reliability improves rapidly at first, then becomes nearly constant before reaching the ideal level of reliability.

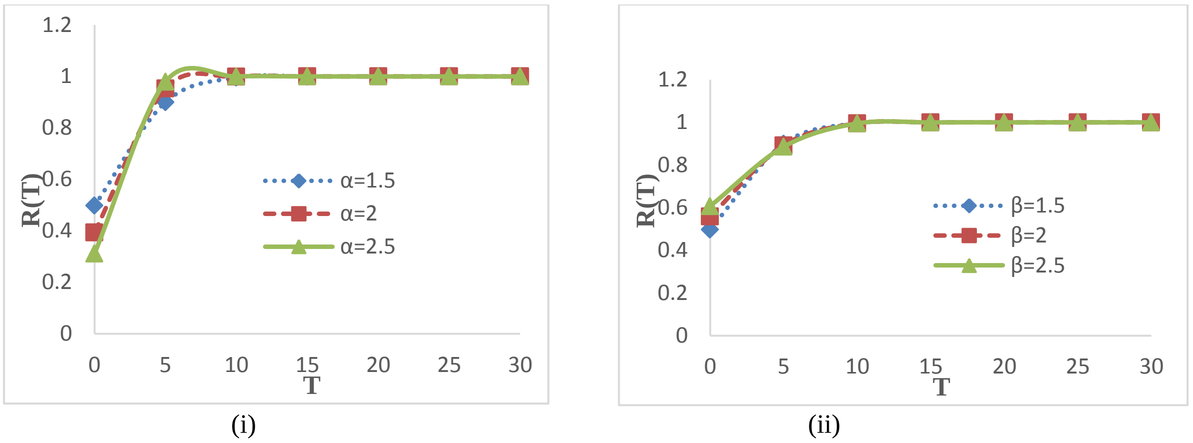


Figure 4: Software reliability v/s testing time in imperfect debugging environment, by varying (i) α and (ii) β

The software reliability is illustrated in Figure 4(i-ii) by varying (T) for various values of shape parameter (α) and scale parameter (β). It is cleared that with the increase in (α), the software reliability increases, whereas, with the increase in (β), the software reliability increases. A significant effect in software reliability with respect to α can be seen in figure 4(i).

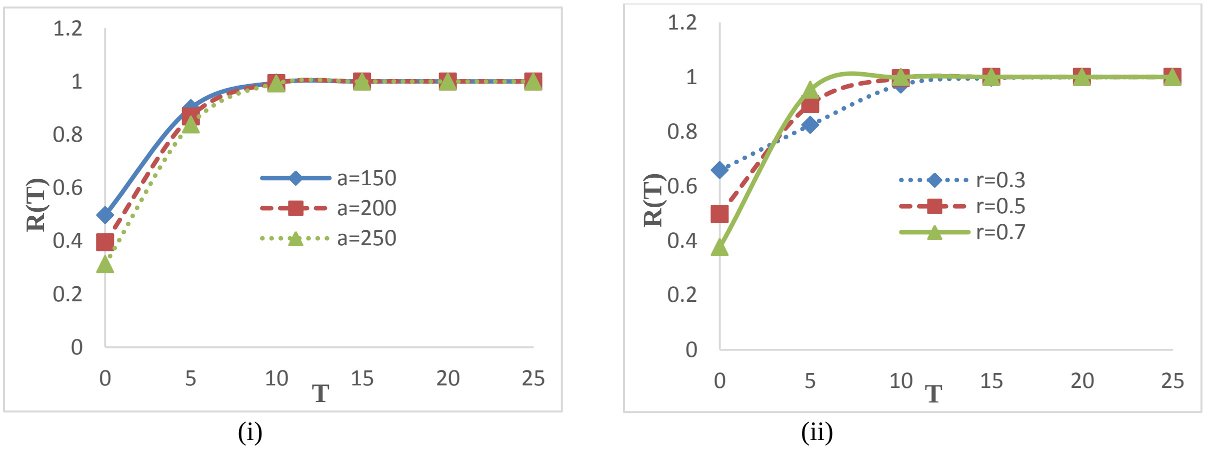
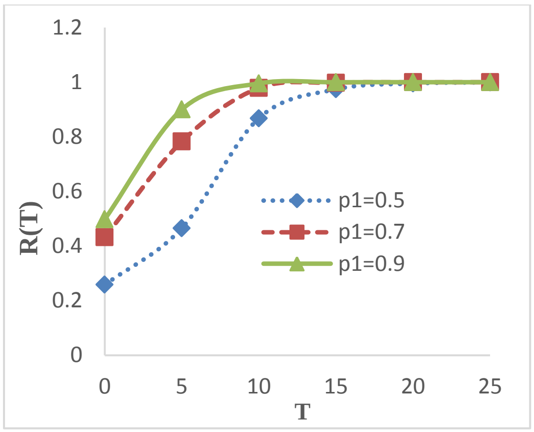


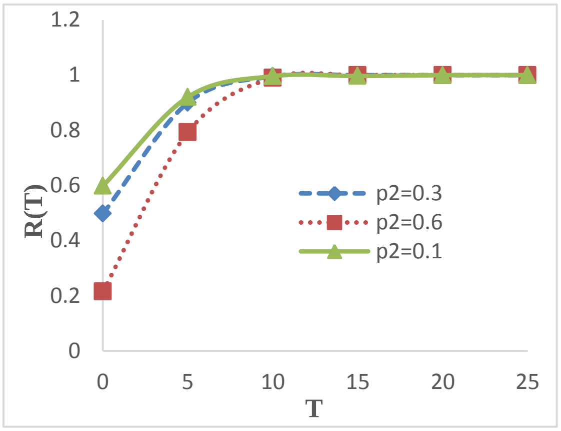
Figure 5: Software reliability v/s testing time in imperfect debugging environment, by varying (i) a and (ii) r

The software reliability is shown in Figure 5(i-ii) by varying (T) for various values of the initial number of faults and the fault detection rate respectively are shown. It is cleared that with the increase in parameter (a) i.e. the initial number of faults in imperfect debugging environment, the reliability decreases. A notable change in the software reliability can be seen in figure 5(ii) with respect to r.

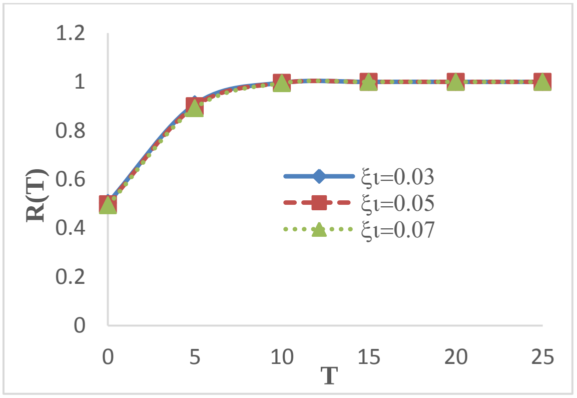
Figure 6(i-ii) depicts the software reliability pattern in an imperfect debugging environment in terms of testing time. For smaller values of T, we find a steady increase in software reliability, which eventually becomes asymptotically constant for larger values of testing time. Figure 6(i) depicts that software reliability increases as increasing values of p1 but, from figure 6(ii), it decreases for increasing values of p2. In figure 6(iii) no significant effect of software reliability can be seen while changing the values of ξ.



(i)



(ii)



(iii)

Figure 6: Software reliability v/s T in imperfect debugging environment by varying (i) p_1 (ii) p_2 and (iii) ξ

5. CONCLUSION

Software developer can better manage the quality of their software using estimates of reliability growth models. In project management, reliability estimation plays a crucial role. In this paper, we have developed the model for inflection S-shaped by considering time independent fault content factor for perfect debugging environment whereas other model differs by involving time dependent total fault content factor for imperfect debugging environment. The sensitivity analysis clearly shows that the parameters α, β, ξ, p_1, p_2, r, a are the influential parameters for software reliability in the models. It can be seen clearly with the help of graphs that by increasing/decreasing these parameters software reliability can be increased. Fault correction is being developed in parallel with fault corrector detection. Because the number of detected faults that need to be corrected varies greatly over time, it would be more cost-effective to adjust the number of correctors rather than fix it. Such an adaptive method aids in keeping the correction under control and detecting and correcting faults in a timely manner. However, such an analysis requires more field data from software development projects, thus the accompanying illustration will have to wait till more data becomes available.

REFERENCES

Boland, P.J. & Chuiv, N.N. (2007). Optimal times for software release when repair is imperfect. *Statistics & Probability Letters*, 77(12), 1176-1184.

Byun, S.K (2017). Some successful approaches to software reliability modelling in industry. *IEEE Transactions on Computers*, 74(5), 85-99.

Chang, H., Pham, H., Lee, S.W. & Song. K.Y. (2014). A testing coverage software reliability model with the uncertainty of operating environment. *International Journal of System Science, Operations & Logistics*, 1(4), 220-227.

Chatterjee, S., Saha, D. & Sharma, A. (2021). Multi-upgradation software reliability growth model with dependency of faults under change point & imperfect debugging. *Journal of Software Evolution and Process*, 33(2), 23-44.

Chaudhary, S., Singh, D. Sinha, A. & Mishra, S. (2020). A review on software reliability growth modelling. *International Journal of Engineering Research & Technology*, 8(10), 1-4.

Chiu, K.C., Huang, Y.S. & Lee, T.Z. (2008). A study of software reliability growth model from the perspective of learning effects. *Reliability Engineering and System Safety*, 93(10), 1410-1421.

Goel (1985). Time dependent error detection rate model for software reliability and other performance measures. *IEEE Transactions on Reliability*, 17(2) 1206-1211.

Huang, C.Y., Lin, C.T. & Su, Y.S. (2005). Modeling and prediction of software operational reliability. *International Journal of Technology, Engineering and Education*, 2, 91-100.

Huang, C.Y., Lyu & M.R. and Kuo, S.Y. (2003). A unified scheme of non-homogeneous Poisson process for software reliability estimation. *IEEE Transactions on Engineering*, 29, 261-269.

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

Kapur, P.K., Bardhan, A. & Kumar, K. (2006). On categorization of errors in a software. *International Journal of Management Software*, 16, 37-48.

Lin, C.T. & Huang, C.Y. (2008). Enhancing and measuring the predictive capabilities of testing effort dependent software reliability models. *Journal of Systems and Softwares*, 81, 1025-1038.

Malik P., Nautiyal L. &, Ram M., (2019). Tools and techniques in software reliability modeling. *Advances in System Reliability Engineering*, Academic press, 281-299.

Mirchandani, C. (2018). Adaptive software reliability growth. *Procedia Computer Science,* 140,122-132.

Musa, J.D., Iannio, A. & Okumoto, K. (1987). Software reliability measurement, prediction and application. New York, McGraw-Hill.

Pham, H., Nordmann, L., & Zhang, X.M (1999). A general imperfect software debugging model with S-shaped fault detection rate. *IEEE Transactions on Reliability*, 48, 169-175.

Roy, P., Mahapatra, G.S. & Dey, K.N. (2014). An NHPP software reliability growth model with imperfect debugging and error generation. *International Journal of Reliability, Quality and Safety Engineering*, 21, 1-32

Wang, J., Wu, Z., Shu, Y. & Zhang, Z. (2016). An optimized method for software reliability model based on nonhomogenous Poisson process. *Applied Mathematical Modelling*, 40(13-14), 6324-6339.

Xiao, X. & Dohi, T. (2011). Estimating software reliability using extreme value distribution. *Communications in Computer and Information Science*, 257, 399-406.

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.