Yugoslav Journal of Operations Research
31 (2021), Number 2, 249–264
DOI: https://doi.org/10.2298/YJOR200117005K

MEASURING SOFTWARE RELIABILITY UNDER THE INFLUENCE OF AN INFECTED PATCH

Jasmine KAUR
Department of Operational Research, University of Delhi, Delhi, 110007, India.
jasminekaur.du.aor@gmail.com

Adarsh ANAND
Department of Operational Research, University of Delhi, Delhi, 110007, India.
adarsh.anand86@gmail.com

Ompal SINGH
Department of Operational Research, University of Delhi, Delhi, 110007, India.
drompalsingh1@gmail.com

Vijay KUMAR
Department of Mathematics, Amity Institute of Applied Sciences, Amity University Uttar Pradesh,Noida, India.
corresponding author:vijay_parashar@yahoo.com

Received: January 2020 / Accepted: June 2020

Abstract: Patching service provides software firms an option to deal with the leftover bugs and is thereby helping them to keep a track of their product. More and more software firms are making use of this concept of prolonged testing. But this framework of releasing unprepared software in market involves a huge risk. The hastiness of vendors in releasing software patch at times can be dangerous as there are chances that firms release an infected patch. The infected patch (es) might lead to a hike in bug occurrence and error count and might make the software more vulnerable. The current work presents an understanding of such situation through mathematical modeling framework; wherein, the distinct behavior of testers (during in-house testing and field testing) and users is described. The proposed model has been validated on two software failure data sets of Tandem Computers and Brazilian Electronic Switching System, TROPICO R-1500.

Keywords: Infected Patch, Patching, Software Reliability.

MSC: 90B85, 90C26.

1. INTRODUCTION

An increase in the complexity of a program increases the probability of occurrence of a fault or an error. A simple internet search will give us numerous examples of a small or big bug causing great distress. For instance, BAE Automated Systems' software was supposed to handle baggage at the Denver International Airport. On its October 1993 launch date, it lost or misdirected so much amount of luggage, or delivered so much of it into the conveyor that the opening of the airport had to be delayed by 16 months. The cost overrun for the city was $1.1 million per day [10]. On 12 May, 2017 a massive ransomware attack left enumerable individuals and organizations across 150 countries locked out of their data. Their data was encrypted, and unless ransom was paid in form of crypto currency Bitcoin, the data was lost. On the first day out of 4 day attack phase, WannaCry affected 20,000 computers. It was caused due to vulnerability named Eternal Blue in Microsoft's operating system. Microsoft had discovered the vulnerability in March, 2017 and released a patch for it. The reason for the widespread havoc caused by the ransomware was that the systems had not been promptly patched [32].

Such examples make us realize the importance of precision. As the stakes get higher, one realizes the absolute necessity of having bugfree software. Bugfree software is a developer's prototypical creation. But with the amount of complexity that a software has these days and the number of code lines running into thousands; it is quite understandable to have faults at any point of time. This ideology makes the developers work even harder to remove the dormant bugs in the software as a lot of things like life, money and privacy comes at stake. For software to be considered a success, it must be capable to handle the job it was created for. Researchers have come up with many useful parameters to evaluate the worth of a software such as its functionality, usability, reliability, efficiency, maintainability, and portability [22]; among them, software reliability stands a class apart [2] [33].

Software Reliability is defined as the probability that the software system will perform its function without failure for a given period of time under specified operational environment [22] [23]. Software Reliability allows the user to know beforehand how the product is likely to perform. Numerous Software Reliability Growth Models (SRGMs) have been proposed to actually capture and portray the reliability growth of any software [22]. Software testers measure the reliability of software by quantifying the number of faults with the time spent on testing. Thus, it is important to know how long the testing process needs to be performed and when should the software be released for usage. The initial SRGMs such as the Jelinski-Morando Model [17], Musa Model [28], G-O Model [16], Yamada Model [36], Bittanti Model [13], Ohba Model [29], K-G Model [21] etc. captured the basic behavior of faults. Many concepts such as fault categorization, fault complexity, testing effort function, testing efficiency, testing environment, testing coverage,

change point, warranty, testing resource allocation, optimal control theoretic approach of optimal allocation of resources, etc. have been explored in detail in the software reliability literature [7] [8] [22] [25] [26] [27].

In the competitive world of software and technology, firms are forced to either keep up with the pace of the market or run out of business. This pressure results in firms releasing their product before it is fully prepared. In such cases, there is always a possibility that some bugs are still lying dormant in the software at its release time. To handle this precarious situation, a relatively new concept of patching has emerged, which is quite helpful in estimating software reliability and in providing after-sales support. Nowadays, all major software firms frequently release patches to handle issues arising in the operational phase. For instance, since October 2003 Microsoft has designated the second Tuesday of each month to release security patches for its products and coined it as "Patch Tuesday". Patches for any zero-day vulnerabilities like in the case of ransomware WannaCry may be also released in between and are termed as out-of-band patch [30].

In today's neck-to-neck competitive environment, every firm wants to bring out its offering as early as possible. In these circumstances, patching can help the software firms in releasing a pre-mature product which can undergo continual testing once the product is in operational phase. Patch is generally a small set of corrective code released during the operational phase of software to fix the bugs or to update them [1]. The earliest version of a patch was in the form of paper tape/punched cards. Later on, one could download such files from the vendor's site but now, automated software updates are available. Administrators also use patch management systems that handle the patch application process. Like, in the work by Arora *et al.* [9]; which has used utility theory to analyze the trade-off time to release a faulty product or to invest in post-release support. Their proposal gave a comparative study of the behavior of a social vendor, a software monopolist, and a tangible goods monopolist towards patching. In another work by Beattie *et al.* [12]; authors have suggested that when patches are available, the crucial question that arises is "When to patch?". It is a rational choice made by the administrator by finding a trade-off between the associated risk and time; for which they formulated a mathematical cost model.

The faults in software are discovered mainly by two groups of people: testers and users. Before the software release, the testers are the sole fault finders and this testing is known as in-house testing. When the testing team continues testing, the product even after its release, it is referred to as field testing. Field testing involves the role of both the testers as well as the product users. Once the software has reached the users, the users report a fault as and when they find it. In this way, both the testers and the user work simultaneously to identify faults and eventually the software is debugged by testers. To debug these newly discovered faults, the developers release a corrective code in the form of a patch. Das *et al.* [14] and Deepika *et al.* [15] have explored the joint role of tester and user in a patching based environment. Several practitioners have worked on this line of research viz, Jiang *et al.* [18] gave a release time scheduling policy where software is released a bit early and testing continues after the release; after the product is released, the

users also contribute in the testing process by reporting back faults to the vendor, which eventually leads to cost reduction. Anand *et al.* [3] have also studied the vulnerability patch modeling when the software vulnerabilities are reported by the vendors or reporters. Anand *et al.* [6] have also studied the role of tester and user in finding both faults and vulnerabilities, and hence proposed a patch release policy for the same. Anand *et al.* [1] proposed an economic cost analysis of software based on patching. They proposed a scheduling policy for a software product and showed the importance of patching in lowering the system outages and making the system more cost effective. Kumar *et al.* [24] discussed the reliability, which is a major attribute of the quality of a software, to address the issues of testing cost, release time of software, and a desirable reliability level. Kumar *et al.* [24] developed a reliability growth model implementing software patching to make the software system reliable and cost effective. Tickoo *et al.* [34] have proposed a testing effort based cost model to determine the optimal release time and patch time of a software. Kansal *et al.* [20] have proposed a generalized framework for finding the optimal release and patch time of a software under warranty. Anand *et al.* [5] have also worked on modeling the software patch/update release and software upgrade release phenomena simultaneously. Further, a novel concept of patching with consideration of error generation has been explored by Anand *et al.* [4].

When a users apply the patch on their systems, the problem is supposed to go away. But what if this hastily released patch has not been effectively tested? What if the patch is not able to fix the problem? What if the patch instead of removing problems can create some bugs? What if the problems that were lying dormant since the testing phase started causing unexpected behaviour in the system? There have been many real life situations that have exemplified such scenarios; like in order to keep themselves safe from the ransomware WannaCry, 5 hospitals in Queensland installed security patches. But the security patch was infected and instead, it incremented the problem at hand by denying access to the patient's medical records. It also caused system slowness in the affected hospitals, i.e., Brisbane's Princess Alexandra, Lady Cilento Children's hospital, the Cairns, Mackay, and Townsville hospitals [11]. According to the finding of a survey conducted by GFI software in 2011, 50% of the surveyed IT firms have faced, at least, one critical IT failure due to application of badly created security patches. Further, 43% of the legal sector and 40% of the healthcare sector suffer due to recurring problems caused by installation of bad security patches [37].

The idea of this paper is to consider the above questions and create a relevant modeling framework. Working on the lines of Anand *et al.* [4], it has been assumed that sometimes a patch can be destructive, i.e., rather than providing benefits it results into some failure. To keep up with the pace of the market, the developers release patches. Sometimes infectious or broken patches are released, which not only affects the process but also increases the number of faults due to error generation. Providing early patches might lead to the application of a faulty patch that eventually can lead to reduction in systems performance. In this paper, three different models have been proposed, which consider the different functional

forms of joint efforts of testers and users based on the fact that fixing bugs by issuing patches may sometimes result into increased fault count or it may happen that some bugs may not be perfectly debugged. Table 1 shows how our analysis is significant over the research done earlier in the field of software patch modeling.

Table 1: Comparison between Present and Prior Research

Concept	**Present Research**	**Anand *et al.* [1]**	**Jiang and Sarkar [18]**	**Kansal *et al.* [20]**	**Research on Concept of Patch***
Joint role of tester and user in fault debugging	*Yes*	Yes	Yes	Yes	No
Release time and Testing Stop Time As two Separate Scenario	*Yes*	Yes	Yes	Yes	No
Impact of an infected patch	*Yes*	No	No	No	No
Importance of patch testing before releasing for any fixation	*Yes*	No	No	No	No
Different Detection/Removal rate of testers before and after software release	*Yes*	Yes	Yes	Yes	No

*Research like: Arora *et al.* [9] and Beattie *et al.* [12].

The structure of the paper is as follows: Section 2 comprises of modeling framework with different set of notations; Section 3 covers the Model Validation and Data Analysis, followed by the Conclusion in section 4.

2. MODEL BUILDING

Consider the situation where the software is released in the market and the users are using it. When users face an issue with their products, they report the same to the producer. The software firms consider the issue and take an appropriate action. To handle the presence of bugs, software patches are then released by the vendors. The possibility that some of these patches being infected and eventually impacting the software reliability has been modeled here. We assume that the fault removal process is modeled as Non-Homogeneous Poisson Process (NHPP). Further, it is assumed that before the release of the software, the testing team is the sole fault finder. But once the software is in operational phase, both the tester and the user are detecting faults in the software with differing efficiencies. The model also considers the varying testing environment before and after the release of the software. Generally, before the software release, the testing team works very hard to make their product the best possible. But once the product has been released, the efficiency of the team tends to go down. The changing behavior of the tester, i.e., their seriousness to deal with the software, ultimately affects the software reliability. Thus, when we consider the impact of the field testing on software reliability growth, it is essential that we keep this changing testing environment into consideration.

Notations

The following notations have been used throughout this paper:

$m(t)$: Cumulative number of faults detected/corrected in the software.

a : Total number of faults in the software.

b_1 : Rate of fault removal per remaining fault before software release.

b_2 : Rate of fault removal per remaining fault after software release.

β_1 : Learning parameter before the software release.

β_2 : Learning parameter after the software release.

α : Bug generation due to infected patch.

p : Probability of perfect debugging of the software.

τ : Software release time.

$\otimes$: Steiltjes Convolution.

$F(t)$: Cumulative Distribution function(CDF) for fault debugging by the tester.

$G(t)$: Cumulative Distribution Function (CDF) for fault reporting by the user.

$g(t)$: Probability Distribution Function (PDF) for fault reporting by the user.

For the mathematical formulation, Software testing lifecycle $[0, T]$ has been divided into following two phases: $0 \leq t < \tau$, when the software is with the developer and in $\tau < t \leq T$, when the software is in operational phase and is being utilized by users. The timeline can be understood with the help of Figure 1.

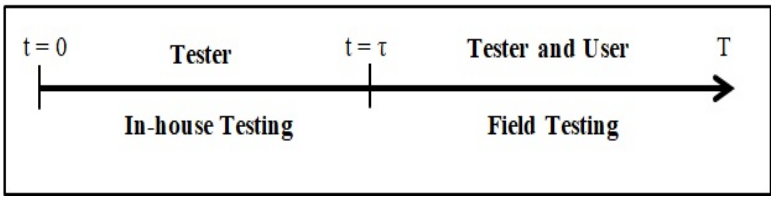

Figure 1: Software Testing Life Cycle

Phase 1:

For time period $0 \leq t < \tau$, in-house testing of the software takes place. Test cases are developed, executed and the testers try their best to debug the software based on these test cases. Extensive effort is being made to have the least number of faults possible at the release time of the software. Here, it has been assumed that fault removal process has an S-shaped distribution growth pattern as given by Kapur & Garg [21] which can be modeled as follows:

$$m(t) = a.F(t) = a\left(\frac{1 - e^{-b_1 t}}{1 + \beta_1 e^{-b_1 t}}\right) \tag{1}$$

Phase 2:

Phase 2 is the operational phase of the software. At time $t = \tau$, the software is released to the public. Thus, for time $\tau < t \leq T$, the users will also be contributing in finding faults in the software. It can be done in the form of feedback of the users taken by the vendors. The users also voluntarily report back issues and product aspirations to vendors. Thereby, both the testers and users shall contribute in the debugging process. To incorporate the combined effort of testers and users in bug finding, the following functional form of 2 distributions is being used from Steiltjes Convolution in Convolution theory:

$$(F \otimes G)t = \int_0^t F(t-x).g(x)dx \tag{2}$$

where $F(x)$ is the fault debugging rate of the tester throughout the software lifecycle, and $G(x)$ is the fault reporting rate of the user after the software release. As a part of maintenance activities, updates and patches are released in this phase to keep the software in pristine condition. While the updates deal with multiple issues simultaneously, patches are meant to handle singular issues as soon as possible. Multiple patches are released throughout the maintenance phase to quick-fix the issues. There is a possibility that among these numerous patches, a patch might be infected, i.e., it has not been effectively tested and might further increment the bug content of the software. With the help of unified modelling approach, this phenomenon can be mathematically expressed as follows [22]:

$$m(T-\tau) = \frac{a'}{(1-\alpha)}[1-(1-F \otimes G(T-\tau))^{p(1-\alpha)}] \tag{3}$$

where $a' = a.[1-F(\tau)]$

Using equation (1) and (3) the total number of faults debugged throughout the testing process can be modeled as follows:

$$\begin{aligned} m(T) &= m(\tau) + m(T-\tau) \\ &= a.F(\tau) + \frac{a}{(1-\alpha)}[1-F(\tau)].[1-(1-F \otimes G(T-\tau))^{p(1-\alpha)}] \end{aligned} \tag{4}$$

where $F(\tau) = \frac{1-e^{-b_1 \cdot \tau}}{1+\beta_1 . e^{-b_1 \cdot \tau}}$, i.e., the logistic distribution function for the fault debugging by the tester.

One important aspect to understand is that the users are not trained to find bugs. They report back the issue that they are facing. So their efficiency level cannot be the same as the testers. Similarly, all users cannot be equally efficient in detecting or reporting the issues. The users' bugs finding efficiency will depend on many factors like the amount of time they spend on operating the software, how well versed they are with the software's working, comfort while using the software, etc. So keeping these characteristics in mind, we have proposed three

different scenarios with different fault reporting rate for the user. Hence, user's fault reporting process can be taken to follow a constant, exponential, or logistic distribution whereas the tester's debugging rate is assumed to follow logistic distribution.

Model 1: Here it has been assumed that tester's debugging rate follows logistic distribution and user's fault reporting process is constant over the planning horizon Thus, the tester follows a learning pattern and his fault debugging abilities improve with time while the user's reports faults at the same rate. It can be represented as,
$F(T-\tau) \sim LogisticDistribution$ and $G(t-\tau) \sim Constant.$

Thus, using equation (2) the joint rate of user and tester can be given as:

$$(F \otimes G)(T-\tau) = \frac{1-e^{-b_2(t-\tau)}}{1+\beta_2 e^{-b_2(T-\tau)}} \tag{5}$$

Using equation (5) in equation (4), the overall fault removal phenomenon can be modeled as follows:

$$m(T) = a\left(\frac{1-e^{-b_1\tau}}{1+\beta_1 e^{-b_1\tau}}\right) + \frac{a'}{1-\alpha}\left[1-\left(1-\frac{1-e^{-b_2(T-\tau)}}{1+\beta_2 e^{-b_2(T-\tau)}}\right)^{p(1-\alpha)}\right] \tag{6}$$

where $a' = a.[1-F(\tau)]$

Model 2: The second model considers the tester's debugging rate to be following a logistic distribution and user's fault reporting rate to be following exponential distribution, i.e., the user's reporting rate increases exponentially with time. Thus, $F(T-\tau) \sim LogisticDistribution$ and $G(T-\tau) \sim ExponentialDistribution.$
$\implies F(T-\tau) = \frac{1-e^{-b_2(T-\tau)}}{1+\beta_2 e^{-b_2(T-\tau)}}$ and $G(T-\tau) = 1-e^{-b_2(T-\tau)}$

Thus, using equation (2) we have

$$(F\otimes G)(T-\tau) = \left(1-e^{-b_2(T-\tau)}+(1+\beta_2)e^{-b_2(T-\tau)}.log\left(\frac{(1+\beta_2)e^{-b_2(T-\tau)}}{1+\beta_2 e^{-b_2(T-\tau)}}\right)\right) \tag{7}$$

Using equation (7) in equation (4) we have the fault removal process for testing process inculcating patching as an attribute, which may sometimes lead to incorporation of infected patches.

$$m(T) = a\left(\frac{1-e^{-b_1\tau}}{1+\beta_1 e^{-b_1\tau}}\right) + \frac{a'}{1-\alpha}\left[1-\left(1-\left(1-e^{-b_2(T-\tau)}\right.\right.\right.$$
$$\left.\left.\left.+(1+\beta_2)e^{-b_2(T-\tau)}.log\left(\frac{(1+\beta_2)e^{-b_2(T-\tau)}}{1+\beta_2 e^{-b_2(T-\tau)}}\right)\right)\right)^{p(1-\alpha)}\right] \tag{8}$$

where $a' = a.[1 - F(\tau)]$

Model 3: The third model assumes that tester's fault detection rate is following logistic distribution i.e. $F(T-\tau) \sim LogisticDistribution$ and keeping the same distribution for users fault reporting phenomenon i.e.$G(T-\tau) \sim LogisticDistribution$ .We have taken a learning pattern for the user's reporting rate to demonstrate the role of those users who use the product very frequently and take active interest in the fault report process.

$\implies F(T-\tau) = \frac{1-e^{-b_2(T-\tau)}}{1+\beta_2.e^{-b_2(T-\tau)}}$ and $G(T-\tau) = \frac{1-e^{-b_2(T-\tau)}}{1+\beta_2.e^{-b_2(T-\tau)}}$

Using equation (2) we have:

$$(F \otimes G)(T-\tau) = \left(\frac{1-e^{-b_2(T-\tau)}}{1-\beta_2^2.e^{-b_2(T-\tau)}}\right) + \left(\frac{e^{-b_2(T-\tau)}(1+\beta_2)^2}{(1-\beta_2^2 e^{-b_2(T-\tau)})^2}\right). \left[b_2(T-\tau) + 2log\left(\frac{(1+\beta_2)e^{-b_2(T-\tau)}}{1+\beta_2 e^{-b_2(T-\tau)}}\right)\right] \quad (9)$$

Further, using equation (9) in equation (4), the mean value function for overall fault removal process can be given as follows:

$$m(T) = a.\left(\frac{1-e^{-b_1\tau}}{1+\beta_1 e^{-b_1\tau}}\right) + \frac{a'}{1-\alpha}\left[1-\left(1-\left(\left(\frac{1-e^{-b_2(T-\tau)}}{1-\beta_2^2.e^{-b_2(T-\tau)}}\right) + \left(\frac{e^{-b_2(T-\tau)}(1+\beta_2)^2}{(1-\beta_2^2 e^{-b_2(T-\tau)})^2}\right).\left[b_2(T-\tau) + 2log\left(\frac{(1+\beta_2)e^{-b_2(T-\tau)}}{1+\beta_2 e^{-b_2(T-\tau)}}\right)\right]\right)\right)^{p(1-\alpha)}\right] \quad (10)$$

where $a' = a.[1 - F(\tau)]$.

3. DATA ANALYSIS AND MODEL VALIDATION

The above models have been analyzed and validated on two fault count dataset. The first dataset ($DS{-}I$) consists of the fault count data of Tandem computers and contains 100 faults discovered over a period of 20 weeks [35]. The second dataset ($DS-II$) is the fault count data of a Brazilian Electronic Switching System, TROPICO R-1500, and contains 461 faults discovered over 81 time points (1 time point corresponds to 10 days) [19]. Parameter estimation has been done using the Statistical Analysis Software (SAS) [31]. The estimated parameter values for the two datasets are shown in Table 2. Table 3 shows the performance of the 3 models on different validation criteria like Sum of Squared Errors (SSE), Mean Square Error (MSE), Root Mean Square Error (R-MSE), R-Square (R^2) and Adjusted R-Square ($AdjR^2$).

The deviation between original fault data and the observed fault data for the three proposed models on $DS{-}I$ has been shown in Figures 2, 3, and 4 respectively.

Figures 5, 6, and 7 show the graphical representation of the deviation between original fault count and the observed fault data for three models on $DS-II$.

Figures 2-7 depict that the models are able to predict the data to a very good extent.

Table 2: Parameter Estimation for proposed models on DS-I and DS-II

Datasets	Models\Parameters	a	b_1	b_2	β_1	β_2	α	p
DS-I	**Model 1**	103.421	0.173	0.54	1.023	2.12	0.122	0.39
DS-I	**Model 2**	102.906	0.231	0.87	2.02	4.01	0.032	0.52
DS-I	**Model 3**	102.36	0.292	0.94	4.03	2.49	0.001	0.731
DS-II	**Model 1**	551.338	0.050	0.653	1.894	9.115	0.012	0.035
DS-II	**Model 2**	519.426	0.056	0.671	2.121	5.015	0.018	0.048
DS-II	**Model 3**	513.506	0.060	0.914	2.343	6.409	0.011	0.038

Table 3: Comparison Criteria for proposed models on $DS-I$ and $DS-II$

Datasets	Models\Parameters	SSE	MSE	R-MSE	R^2	$AdjR^2$
DS-I	**Model 1**	162.1	9.006	3.001	0.990	0.989
DS-I	**Model 2**	296.8	16.487	4.060	0.982	0.981
DS-I	**Model 3**	358.7	19.929	4.464	0.978	0.977
DS-II	**Model 1**	5664.1	73.560	8.577	0.996	0.996
DS-II	**Model 2**	6277.9	81.531	9.030	0.996	0.995
DS-II	**Model 3**	7161.4	93.005	9.644	0.995	0.995

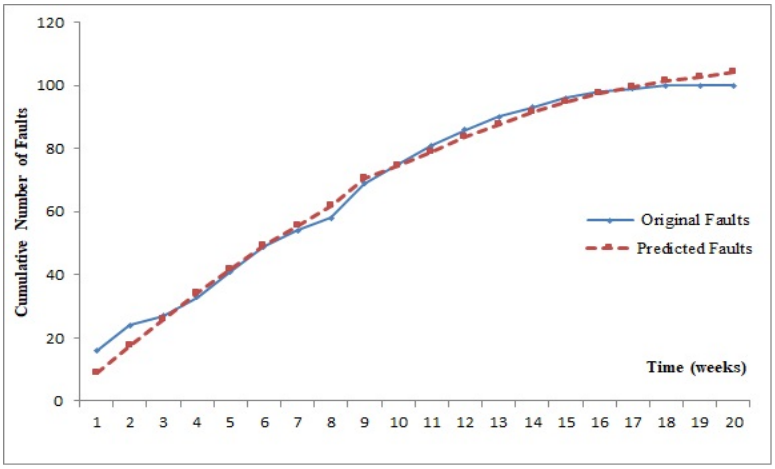

Figure 2: Goodness of fit curve for Model 1 on DS-I

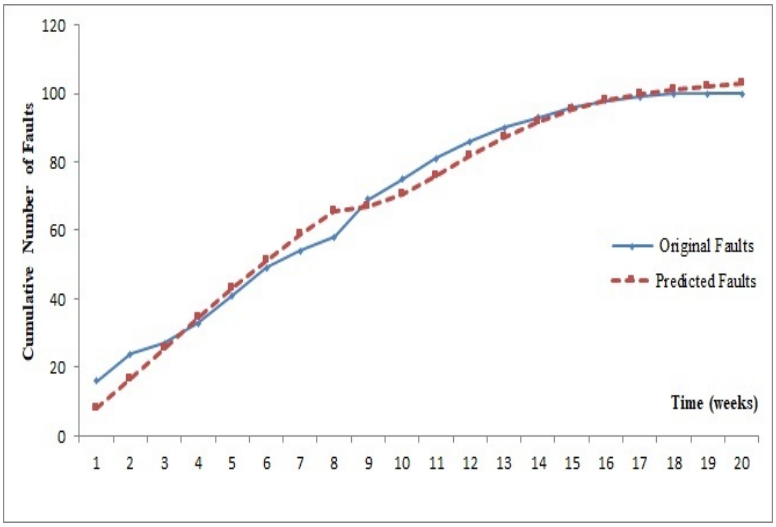

Figure 3: Goodness of fit curve for Model 2 on DS-I

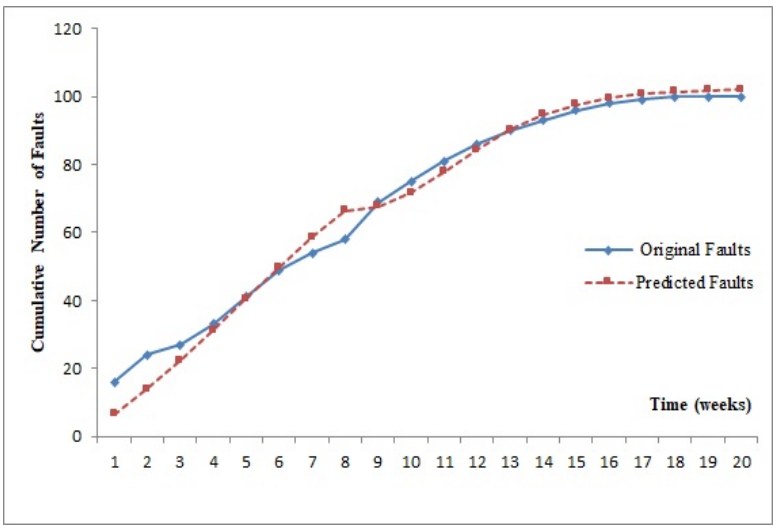

Figure 4: Goodness of fit curve for Model 3 on DS-I

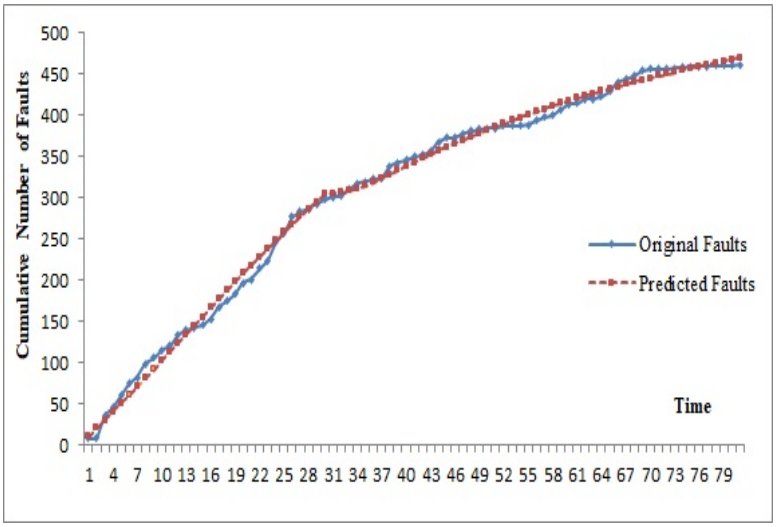

Figure 5: Goodness of fit curve for Model 1 on DS-II

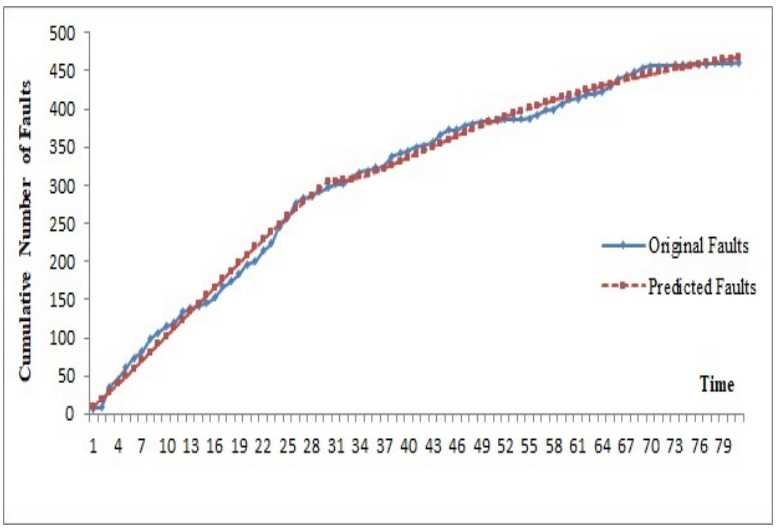

Figure 6: Goodness of fit curve for Model 2 on DS-II

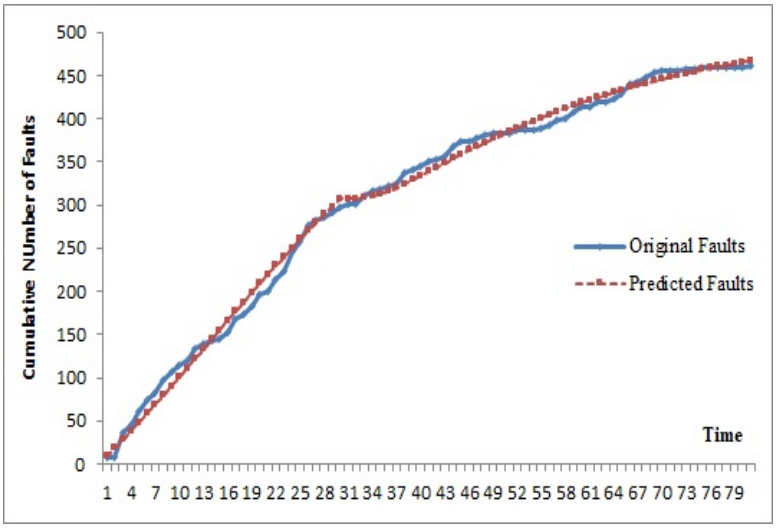

Figure 7: Goodness of fit curve for Model 3 on DS-II

4. CONCLUSIONS

This paper considers the effect of an infected patch on the software reliability growth. If the patches are not effectively tested, i.e., the faults are not completely removed at patch release time, then such infected patches can increase the fault content in the software. Before a software is released, testers are to detect bugs and remove them, but after their release, both testers and users are to find faults which are then corrected by the patches, issued by the developers. The users can have different levels of fault reporting rate depending upon their software use intensity and fault finding efficiency. To model the above concept, we proposed a generic model based on the concept of firms providing patching service wherein it is assumed that sometimes these corrective measures result in implementation of infected patch. Further, based on differing user efficiency, three models have been formulated and analyzed on two sets of software failure data. The results show that the predicted values obtained using the proposed models were very close to the actual values. Thus, the phenomenon of infected patching and the differing behaviour of testers and users in such a situation can be explained effectively through our models.

Acknowledgement: The research work presented in this paper is supported by grant to the second and the third author from Department of Science and Technology, India through DST PURSE PHASE II scheme.

REFERENCES

[1] Anand, A., Agarwal, M., Tamura, Y., and Yamada, S., " Economic impact of software patching and optimal release scheduling", *Quality and Reliability Engineering International*, 33 (1) (2017) 149–157.

[2] Anand, A., and Bansal, G., "Interpretive structural modelling for attributes of software quality", *Journal of Advances in Management Research*, 14 (3) (2017) 256-269.

[3] Anand, A., Bhatt, N., and Aggrawal, D., "Modeling Software Patch Management Based on Vulnerabilities Discovered", *International Journal of Reliability, Quality and Safety Engineering*, 27(2) (2019) 2040003.

[4] Anand, A., Das, S. and Singh, O., "Modeling software failures and reliability growth based on pre & post release testing", *5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO), Delhi, India*, (2016) 139–144.

[5] Anand, A., Das, S., Aggrawal, D., and Kapur, P. K., "Reliability analysis for upgraded software with updates", *Quality, IT and business operations, Springer Natur, Singapore*, (2018) 323–333.

[6] Anand, A., Gupta, P., Klochkov, Y., and Yadavalli, V. S. S., "Modeling Software Fault Removal and Vulnerability Detection and Related Patch Release Policy", *System Reliability Management: Solutions and Technologies, CRC Pess, Boca Raton, Florida*, (2018) 19-34.

[7] Anand, A., and Ram, M., "System Reliability Management: Solutions and Technologies", *CRC Press, Boca Raton, Florida*, (2018).

[8] Anand, A., and Ram, M., "Recent Advancements in Software Reliability Assurance", *CRC Press, Boca Raton, Florida*, (2019).

[9] Arora, A., Caulkins, J. P. and Telang, R., "Research note—Sell first, fix later: Impact of patching on software quality", *Management Science*, 52 (3) (2006) 465-471.

[10] Babcock, C., "What's The Greatest Software Ever Written?", `https://www.informationweek.com/whats-the-greatest-software-ever-written/d/d-id/1046033`, (2006).

[11] Bateman, D., "Cairns Hospital suffers software 'catastrophe' with possible loss of patient data", `https://www.cairnspost.com.au/news/cairns-hospital-suffers-software-catastrophe-with-possible-loss-of-patient-data/news-story/c828de3f4a0f73132ec3d19284cbae88`, (2017).

[12] Beattie, S., Arnold, S., Cowan, C., Wagle, P., Wright, C., and Shostack, A., "Timing the Application of Security Patches for Optimal Uptime", *LISA*, 2 (2002) 233-242.

[13] Bittanti, S., Bolzern, P., Pedrotti, E., Pozzi, M., and Scattolini, R., "A flexible modelling approach for software reliability growth", *Software Reliability Modelling and Identification, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, (1988) 101-140.

[14] Das, S., Anand, A., Singh, O., and Singh, J., "Influence of Patching on Optimal Planning for Software Release and Testing Time", *Communications in Dependability and Quality Management- An International Journal*, 18 (4) (2015) 81-92.

[15] Deepika, Anand, A., Singh, N., and Dutt, P., "Software reliability modeling based on in-house and field testing", *Communications in Dependability and Quality Management-An International Journal*, 19 (1) (2016) 74-84.

[16] Goel, A. L., and Okumoto, K., "Time-dependent error-detection rate model for software reliability and other performance measures", *IEEE transactions on Reliability*, 28 (3) (1979) 206-211.

[17] Jelinski, Z., and Moranda, P., "Software reliability research", *Statistical computer performance evaluation*, Academic Press, New York,United States (1972), 465-484.

[18] Jiang, Z., and Sarkar, S., "Optimal software release time with patching considered", *Proc. 13th Annual Workshop Information technologies and Systems, Seattle, Washington*, (2003) 61-66.

[19] Kanoun, K., de Bastos Martini, M. R., and de Souza, J. M., "A method for software reliability analysis and prediction application to the TROPICO-R switching system", *IEEE Transactions on Software Engineering*, 17 (4) (1991) 334-344.

[20] Kansal, Y., Singh, G., Kumar, U., and Kapur, P. K., "Optimal release and patching time of software with warranty", *International Journal of System Assurance Engineering and Management*, 7 (4) (2016) 462-468.

[21] Kapur, P. K., and Garg, R. B., "A software reliability growth model for an error-removal phenomenon", *Software Engineering Journal*, 7 (4) (1992) 291-294.

[22] Kapur, P. K., Pham, H., Gupta, A., and Jha, P. C., "Software reliability assessment with OR applications", *London: Springer*, (2011).

[23] Kapur, P. K., Pham, H., Chanda, U., and Kumar, V., "Optimal allocation of testing effort during testing and debugging phases: a control theoretic approach", *International Journal of Systems Science*, 44 (9) (2013) 1639-1650.

[24] Kumar, V., Singh, V. B., Dhamija, A., and Srivastav, S., "Cost-reliability-optimal release time of software with patching considered", *International Journal of Reliability, Quality and Safety Engineering*, 25 (4) (2018) 1850018.

[25] Kumar, V., Khatri, S. K., Dua, H., Sharma, M., and Mathur, P., " An assessment of testing cost with effort-dependent fdp and fcp under learning effect: a genetic algorithm approach", *International Journal of Reliability, Quality and Safety Engineering*, 21 (6) (2014) 1450027.

[26] Kumar,V., Kapur, P. K., Taneja, N., and Sahni, R., " On allocation of resources during testing phase incorporating flexible software reliability growth model with testing effort under dynamic environment", *International Journal of Operational Research*, 30 (4) (2017) 523-539.

[27] Kumar, V., and Sahni, R., " An effort allocation model considering different budgetary constraint on fault detection process and fault correction process", *Decision Science Letters*, 5 (1) (2016) 143-156.

[28] Musa, J. D., "A theory of software reliability and its application", *IEEE transactions on software engineering*, 13 (1975) 312-327.

[29] Ohba, M., "Inflection S-shaped software reliability growth model", *Stochastic models in reliability theory*, (1984) 144-162.
[30] Rouse, M., "Patch Tuesday", `https://searchsecurity.techtarget.com/definition/PatchTuesday`, (2017).
[31] "SAS/ETS 9.1 User's Guide", *SAS Institute*, (2004) 37–45.
[32] Schneier, B., "The Next Ransomware Attack Will Be Worse than WannaCry", `https://www.schneier.com/essays/archives/2017/05/the_next_ransomware_.html`, (2017).
[33] Singh, O., Anand, A., Aggrawal, D., and Agarwal, M., "Utility based assessment of attributes for software quality", *Proceedings of 5th International DQM conference on life cycle engineering and management(ICDQM-2014), Cacak, Serbia*, (2014) 95-110.
[34] Tickoo, A., Kapur, P. K., Shrivastava, A. K., and Khatri, S. K., "Testing effort based modeling to determine optimal release and patching time of software", *International Journal of System Assurance Engineering and Management*, 7 (4) (2016) 427-434.
[35] Wood, A., "Predicting software reliability", *Computer*, 29 (11) (1996) 69-77.
[36] Yamada, S., Ohba, M., and Osaki, S., "S-shaped reliability growth modeling for software error detection", *IEEE Transactions on reliability*, 32 (5) (1983) 475-484.
[37] GFI Software, "50% of Businesses Have Suffered IT Failures Due to Bad Software Updates", `https://www.gfi.com/company/press/press-releases/2011/06/50-of-businesses-have-suffered-it-failures-due-to-bad-software-updates`, (2011).