Scientific Research Publishing

Journal of Software Engineering and Applications, 2021, 14, 591-606
https://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Comparison of Hazard-Rates Considering Fault Severity Levels and Imperfect Debugging for OSS

Taku Yanagisawa[1], Yoshinobu Tamura[2], Adarsh Anand[3], Shigeru Yamada[4]

[1]Tokyo City University, Tokyo, Japan
[2]Yamaguchi University, Yamaguchi, Japan
[3]University of Delhi, Delhi, India
[4]Tottori University, Tottori, Japan
Email: g2081444@tcu.ac.jp, tamuray@yamaguchi-u.ac.jp, adarsh.anand86@gmail.com, yamada@tottori-u.ac.jp

How to cite this paper: Yanagisawa, T., Tamura, Y., Anand, A. and Yamada, S. (2021) Comparison of Hazard-Rates Considering Fault Severity Levels and Imperfect Debugging for OSS. *Journal of Software Engineering and Applications*, **14**, 591-606.
https://doi.org/10.4236/jsea.2021.1411035

Received: October 14, 2021
Accepted: November 16, 2021
Published: November 19, 2021

Abstract

Software reliability model is the tool to measure the software reliability quantitatively. Hazard-Rate model is one of the most popular ones. The purpose of our research is to propose the hazard-rate model considering fault level for Open Source Software (OSS). Moreover, we aim to adapt our proposed model to the hazard-rate considering the imperfect debugging environment. We have analyzed the trend of fault severity level by using fault data in Bug Tracking System (BTS) and proposed our model based on the result of analysis. Also, we have shown the numerical example for evaluating the performance of our proposed model. Furthermore, we have extended our proposed model to the hazard-rate considering the imperfect debugging environment and showed numerical example for evaluating the possibility of application. As the result, we found out that performance of our proposed model is better than typical hazard-rate models. Also, we verified the possibility of application of proposed model to hazard-rate model considering imperfect debugging.

Keywords

Open Source Software, Bug Tracking System, Software Reliability, Hazard-Rate Model, Imperfect Debugging

1. Introduction

Open source software (OSS) is freely available for use, reuse, fixing, and re-distribution by users and developers. OSS is used under various situations

because OSS is useful for many users to make cost reduction, standardization, and quick delivery. However, the quality of OSS is not good because of the unique development style. The quality of OSS is very important, which depends on the demand of users in the future. The faults latent in OSS are fixed by using database of bug tracking system (BTS). There is various information in terms of the faults recorded in BTS and the model to analyze big fault data in BTS based on deep learning has been proposed as a research [1]. Software reliability is one of the software characteristics factors in order to evaluate the quality of software. A software reliability model is the tool to measure the software reliability quantitatively and many various software reliability models have been proposed by many researchers [2]-[8]. Also, the software effort model based on the software reliability model has been proposed so far [9]. In particular, a hazard-rate model is one of the most popular ones [10] [11].

A lot of hazard-rate models have been proposed so far. However, the hazard-rate model based on fault levels has not been proposed as of today. The purpose of our research is to propose the hazard-rate model considering fault level for OSS. In this paper, we assume that there are different trends on each fault severity level in terms of mean time between software failures (MTBF). Based on the assumption, we analyze the fault big data in BTS and find the difference of trend on each fault severity level in terms of MTBF and we propose a hazard-rate model considering fault severity level for OSS from the result of analysis.

Moreover, we aim to adapt our proposed model to the hazard-rate considering the imperfect debugging environment. Most of the software reliability models are assumed that all detected faults in the software are fixed and removed perfectly and new faults are not introduced at the time when the fault is fixed and removed. However, that assumption is not practical one in actual situation. In other words, we assume that the testing phase and operating phase in software development are in imperfect debugging environment. There are some researches about debugging such as effectiveness of statistical debugging [12]. Also, the software reliability models considering the imperfect debugging environment have been proposed in the past. In this paper, we adapt our proposed model to the hazard-rate model considering the imperfect debugging environment. Then, we show several numerical examples based on the proposed model.

2. Bug Tracking System

The faults in OSS are fixed by using BTS. There are many information related to recorded faults in OSS, e.g., the recorded time of fault, the severity of fault and so on. As the severity of fault in BTS, we show the software fault severity levels [13] in Table 1. 7 kinds of levels in Table 1 are the fault levels of the severity in BTS.

3. Software Reliability Model

A software reliability model is the tool to measure the software reliability quan-

titatively and most of the models are proposed by the probability and statistical theory. The software reliability model is categorized according to analytical model and empirical model. Moreover, the empirical model is categorized into dynamic model and static model. Especially, the dynamic model is presenting the fault discovery event and software failure occurrence event in test phase or operation phase as a process of software reliability growth, which is described as a stochastic model *i.e.*, so-called software reliability growth model (SRGM). In this paper, we use the hazard-rate model which is one of the most popular ones in SRGM.

4. Hazard-Rate Model

In this section, we discuss the hazard-rate model. Firstly, we can express the probability related to the number of software faults and the time of occurrence of software failures in testing phase or operating phase as shown in Figure 1.

Table 1. The list of fault severity levels.

Software Fault Severity Level	Contents
Blocker	The fault is the most serious in all of fault levels.
Critical	This fault is more serious fault comparably but is less serious than blocker.
Major	This level shows that most of certain functions in software are malfunction.
Normal	This fault level is general one.
Minor	The fault happens in minor functions of software.
Trivial	The fault is minor one that has less effectiveness to the function in software.
Enhancement	This level is not fault itself generally but is requested to be revised.

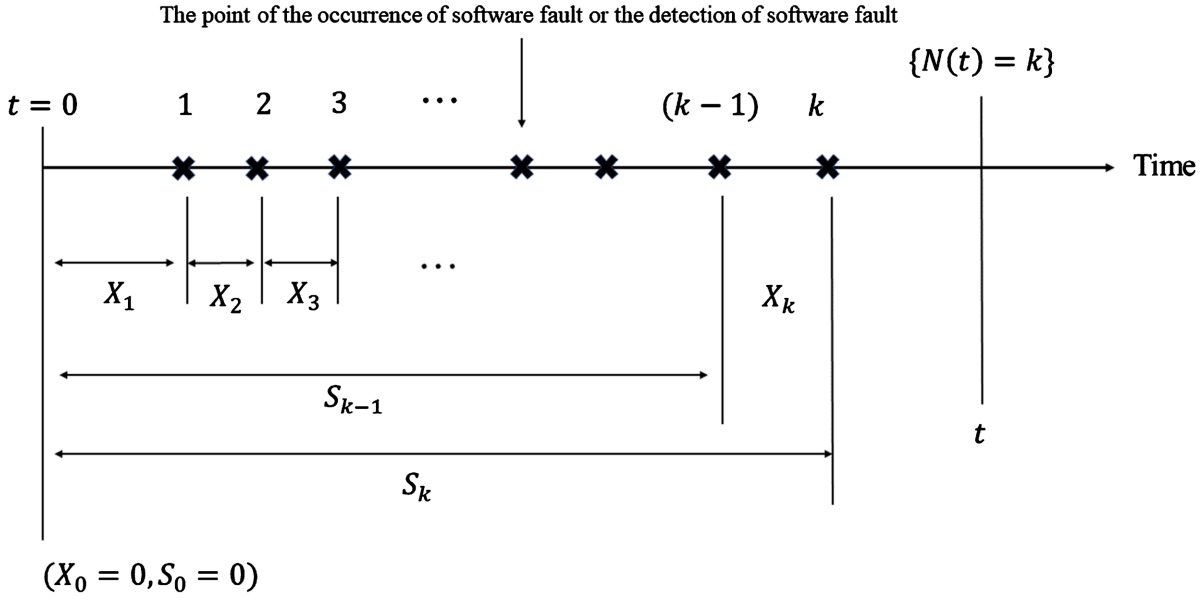

Figure 1. The variables of the software fault detection event and the software fault occurrence one.

The distribution function of $X_k\left(k=1,2,\cdots\right)$ representing the time-interval between successive detected faults of $\left(k-1\right)^{\text{th}}$ and k^{th} is defined as

$$F_k\left(x\right)\equiv \Pr\left\{X_k \le x\right\} \quad \left(x\ge 0\right), \tag{1}$$

where Pr{A} represents the occurrence probability of event A. Therefore, the following derived function means the probability density function of X_k:

$$f_k\left(x\right)\equiv \frac{\mathrm{d}F_k\left(x\right)}{\mathrm{d}x}. \tag{2}$$

Also, the software reliability can be defined as the probability that a software failure does not occur during the time-interval $\left(0,x\right]$. The software reliability is given by

$$R_k\left(x\right)\equiv \Pr\left\{X_k > x\right\} = 1-F_k\left(x\right). \tag{3}$$

From Equations (1)-(3), the hazard-rate is given by the following equation:

$$z_k\left(x\right)\equiv \frac{f_k\left(x\right)}{1-F_k\left(x\right)} = \frac{f_k\left(x\right)}{R_k\left(x\right)}, \tag{4}$$

where the hazard-rate means the software failure rate when the software failure does not occur during the time-interval $\left(0,x\right]$. A hazard-rate model is an SRGM representing the software failure-occurrence phenomenon by the hazard-rate.

Moreover, we discuss three hazard-rate model as follows:

4.1. Jelinski-Moranda Model

Jelinski-Moranda (J-M) model is one of the hazard-rate models. J-M model has the following assumptions:

1) The software failure rate during a failure interval is constant and is proportional to the number of faults remaining in the software.

2) The number of remaining faults in the software decreases by one each time a software failure occurs.

3) Any fault that remains in the software has the same probability of causing a software failure at any time.

From the above assumptions, the software hazard-rate in Equations (4) at k^{th} can be derived as

$$z_k\left(x\right) = \phi\left[N-\left(k-1\right)\right] \quad \left(N>0,\phi>0;k=1,2,\cdots,N\right), \tag{5}$$

where each parameter is defined as follows:

N: the number of latent software faults before the testing,

ϕ: the hazard-rate per inherent fault.

4.2. Moranda Model

Moranda model has the following assumptions:

1. The software failure rate per software fault is constant and is decreasing geometrically as a fault is discovered.

From the above assumptions, the software hazard-rate in Equations (4) at k^{th}

can be derived as

$$z_k(x) = D \cdot c^{k-1} \quad (D > 0, 0 < c < 1; k = 1, 2, \cdots), \tag{6}$$

where each parameter is defined as follows:

D: the initial hazard-rate for the software failure,

c: the decrease coefficient for hazard-rate.

4.3. Xie Model

Xie model has the following assumptions:

1. The software failure rate per software fault is constant and is decreasing exponentially with the number of faults remaining in the software.

From the above assumptions, the software hazard-rate in Equations (4) at k^{th} can be derived as

$$z_k(x) = \lambda_0 (N - k + 1)^{\alpha} \quad (N > 0, \lambda_0 > 0, \alpha \geq 1; k = 1, 2, \cdots, N), \tag{7}$$

where each parameter is defined as follows:

N: the number of latent software faults before the testing,

λ_0 : the hazard-rate per inherent fault,

α : the constant parameter.

4.4. MTBF

Three hazard-rate models above have the following assumption:

- Any fault that remains in the software had the same probability of causing s software failure at any time.

From the above assumption, MTBF by three hazard-rate models can be derived as

$$\mathrm{E}[X_k] = \int_0^{\infty} x f_k(x) \mathrm{d}x = \int_0^{\infty} R_k(x) \mathrm{d}x \equiv \frac{1}{z_k(x)}. \tag{8}$$

5. Observation and Analysis of Trend on Each Fault Level

We analyze the fault big data from the perspective of MTBF in Apache HTTP Server (The Apache Software Foundation) known as the OSS developed under Apache Software Foundation [14]. Especially, we use the data in terms of the fault severity. In this paper, we use 7 kinds of fault levels in severity as shown in the following items:

- Blocker
- Critical
- Major
- Normal
- Minor
- Trivial
- Enhancement

Figure 2 shows the estimation results of MTBF in each fault severity level. Table 2 shows the estimated variance in each fault severity level. In terms of

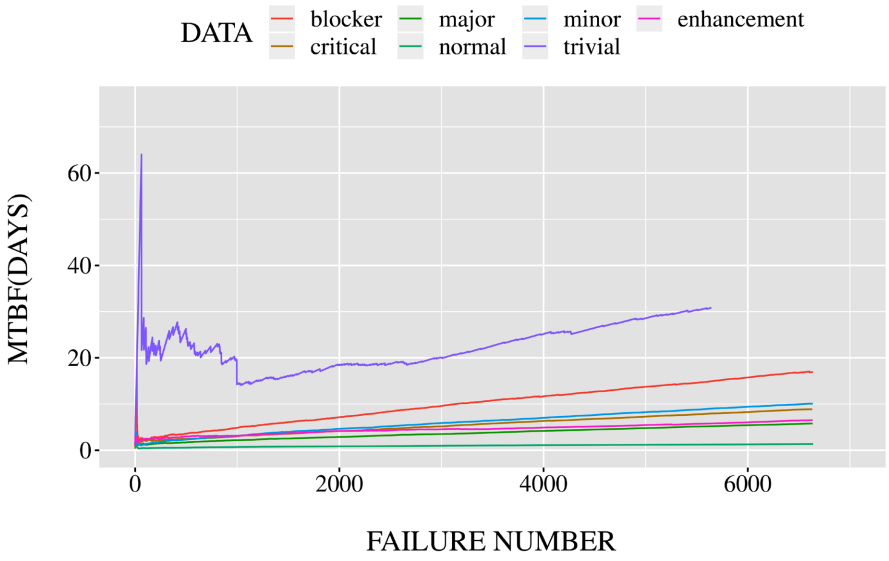

Figure 2. The mean time between software failures in each severity level of faults.

Table 2. The estimated variance in each fault severity level.

Software Fault Severity Level	Estimate of Variance
Blocker	17.76229
Critical	4.12128
Major	1.60268
Normal	0.05783
Minor	5.91591
Trivial	25.22615
Enhancement	1.17400

variance, we find that the value of normal fault is the smallest in all of fault severity levels. In other words, the normal faults occur at a constant frequency, while other fault severity levels occur less as time goes. From the results of analysis, we assume that the fault data is divided into normal fault and others.

6. Application of Hazard-Rate Model to the Actual Data

In this section, we apply typical hazard-rate models to 2 kinds of data sets which are normal fault and other fault in order to find out which hazard-rate models fit to normal fault and other fault in terms of MTBF. We apply the following 3 models to actual fault data.

- Jelinksi Moranda model (J-M)
- Moranda model
- Xie model

We use AIC (Akaike's Information based on the maximum likelihood estimation of model parameters Criterion) to measure the goodness-of-fit of these models to actual data. The result of AIC is shown in **Table 3**. **Figures 3-8** show

the estimated MTBF of each model for both normal fault and other fault data, respectively. From Table 3, we find that the value of AIC in Moranda model is the smallest in both normal fault and others fault.

Table 3. The values of AIC of each model in normal and others fault.

Models	AIC	
	Normal	Others
J-M	13,362.3	13,419.5
Moranda	13,336.5	13,351.5
Xie	13,357.7	13,407.3

Figure 3. The estimated MTBF for normal fault by using J-M model.

Figure 4. The estimated MTBF for normal fault by using Moranda model.

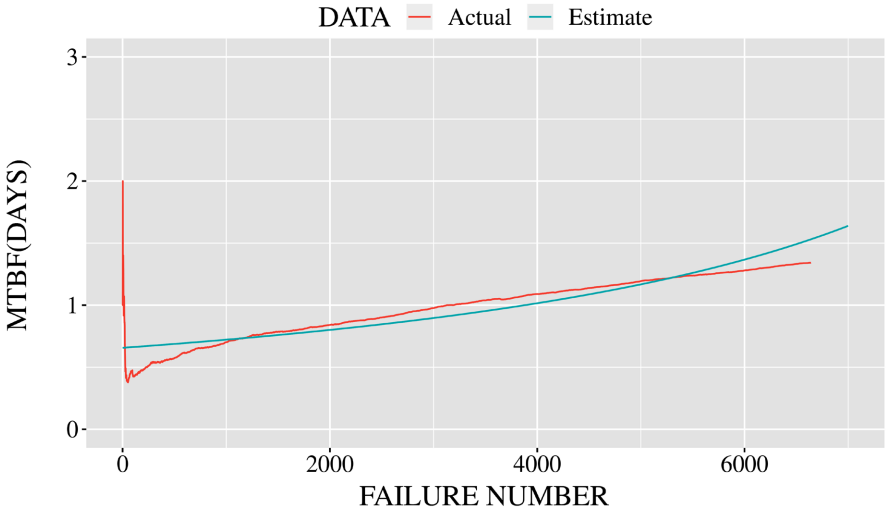

Figure 5. The estimated MTBF for normal fault by using Xie model.

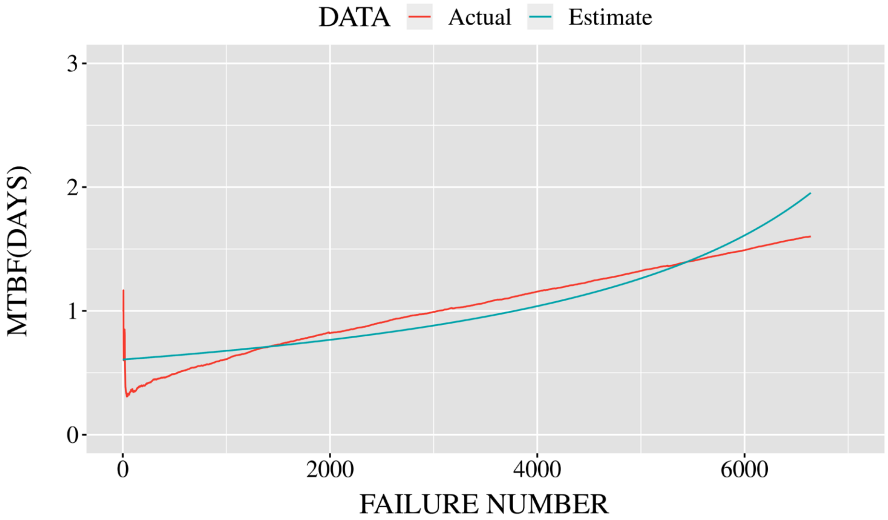

Figure 6. The estimated MTBF for others fault by using J-M model.

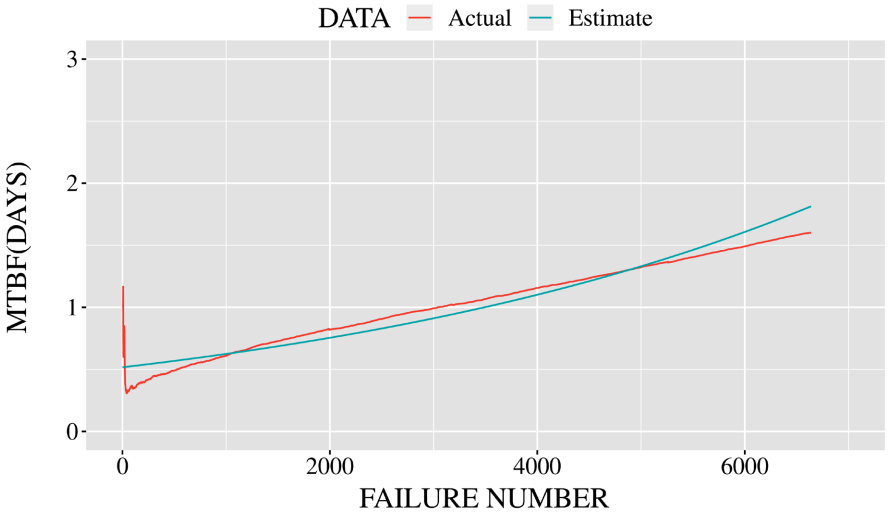

Figure 7. The estimated MTBF for others fault by using Moranda model.

Figure 8. The estimated MTBF for others fault by using Xie model.

7. Proposed Model

From the results of Section 2, we assume that fault data is divided into the following types:

A1. The normal fault

A2. The others fault

In the assumption above, A1 is the fault detected as a normal one, A2 is the fault detected as other one. Also, OSS manager cannot differentiate between assumptions A1 and A2 in terms of the software faults. The time interval between successive faults of $(k-1)^{\text{th}}$ and k^{th} is represented as the random variable $X_k\left(k=1,2,\cdots\right)$, Therefore, the integrated hazard-rate function $z_k\left(x\right)$ for X_k is defined as follows by using Moranda model:

$$z_k\left(x\right)=p\cdot z_k^1\left(x\right)+\left(1-p\right)\cdot z_k^2\left(x\right)\quad\left(k=1,2,\cdots;0\le p\le 1\right), \tag{9}$$

$$z_k^1\left(x\right)=D_1\cdot c_1^{k-1}\quad\left(k=1,2,\cdots;D_1\ge 0,0<c_1<1\right), \tag{10}$$

$$z_k^2\left(x\right)=D_2\cdot c_2^{k-1}\quad\left(k=1,2,\cdots;D_2\ge 0,0<c_2<1\right), \tag{11}$$

where each parameter is defined as follows:

$z_k^1\left(x\right)$: the hazard-rate for assumption A1,

D_1: the initial hazard-rate for the first software failure of A1,

c_1: the decrease coefficient for hazard-rate for assumption A1,

$z_k^2\left(x\right)$: the hazard-rate for assumption A2,

D_2: the initial hazard-rate for the first software failure of A2,

c_2: the decrease coefficient for hazard-rate for assumption A2,

p: the weight parameter for $z_k^1\left(x\right)$.

Equation (10) represents the hazard-rate for a software failure-occurrence phenomenon for the normal fault, On the other hand, Equation (11) represents the hazard-rate for a software failure-occurrence for the other one. Also, we show the diagram to describe the algorithm of proposed method in **Figure 9**.

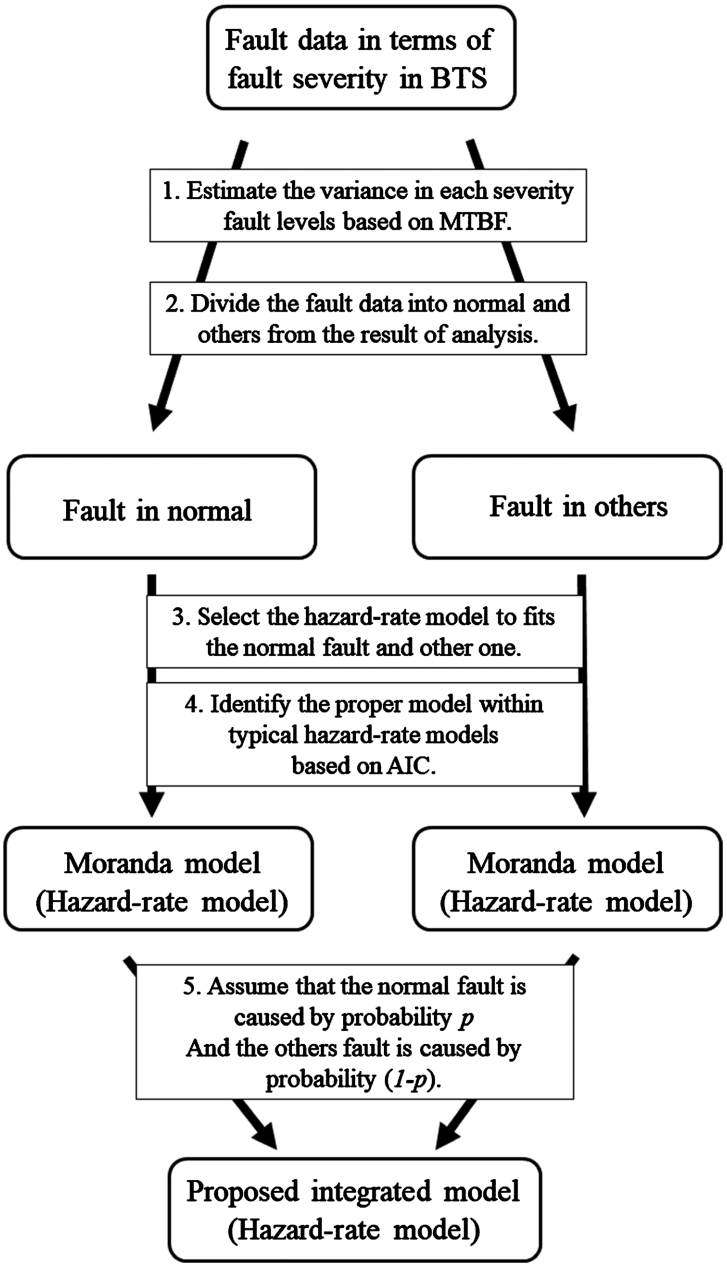

Figure 9. The description of the algorithm of proposed method.

8. Numerical Example

In order to evaluate the performance of the proposed model, we estimate the MTBF of fault big data in Apache HTTP server. The parameters of proposed integrated model have been estimated by MLE (Maximum Likelihood Estimation). The estimated value of parameters is shown as follows:

$$\widehat{w_1} = 2.58319, \quad \widehat{w_2} = 1.26026, \quad \widehat{c_1} = 0.99963, \quad \widehat{c_2} = 0.99999.$$

where $w_1 = pD$ and $w_2 = (1-p)D$ are assumed for the simplification technique. We compare the proposed integrated model with AIC of the typical hazard-rate model. **Figures 10-13** show the estimated MTBF for each model.

Figure 10. The estimated MTBF by using J-M model.

Figure 11. The estimated MTBF by using Moranda model.

Figure 12. The estimated MTBF by using Xie model.

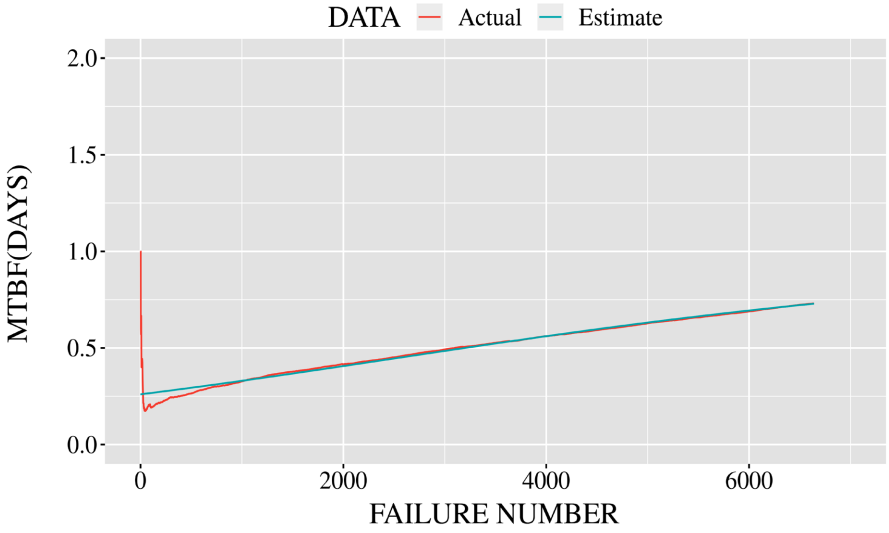

Figure 13. The estimated MTBF by using proposed integrated model.

Table 4. The values of AIC for each model.

Model	AIC
J-M model	13,392.8
Moranda model	13,350.1
Xie model	13,385.0
Proposed integrated model	13,326.5

From Figures 10-13, the typical hazard-rate models estimate MTBF higher than actual one and we found out these models estimate MTBF optimistically. On the other hand, the proposed integrated model estimates MTBF realistically. Table 4 shows the value of AIC for each model. From Table 4, the proposed integrated model fits better than the other model in terms of AIC. In other words, we can predict the MTBF of OSS more precisely with the proposed integrated model.

9. Imperfect Debugging Model

Most of the software reliability models are assumed that all faults found in software are fixed and removed perfectly and new faults is not introduced at the time when the fault is fixed and removed. However, that assumption is not practical one in actual situation. In other words, it insists the testing phase and operation phase in software development is in imperfect debugging environment. The software reliability models considering the imperfect debugging have been proposed [15]. In this paper, we adapt our proposed model to the hazard-rate model considering the imperfect debugging and verify the possibility of application of proposed model to it.

We assume that fault data is divided into the following types:

A3. The latent fault in software before the release

A4. The fault caused by imperfect debugging

In the assumption above, A3 is the latent fault before the release of software and A4 is the fault caused at the time when the latent fault is fixed and removed. Also, OSS manager cannot differentiate between assumptions A3 and A4 in terms of the software faults. The time interval between successive faults of $(k-1)^{\text{th}}$ and k^{th} is represented as the random variables $X_k\left(k=1,2,\cdots\right)$, Therefore, the hazard-rate function $z_k\left(x\right)$ for X_k is defined as follows:

$$z_k\left(x\right)=p\cdot z_k^3\left(x\right)+\left(1-p\right)\cdot z_k^4\left(x\right)\quad\left(k=1,2,\cdots;0\le p\le 1\right), \tag{12}$$

$$z_k^3\left(x\right)=p^1\cdot D_1\cdot c_1^{k-1}+\left(1-p^1\right)\cdot D_2\cdot c_2^{k-1}$$
$$\left(k=1,2,\cdots;D_1\ge 0,0<c_1<1,D_2\ge 0,0<c_2<1,0\le p^1\le 1\right), \tag{13}$$

$$z_k^4\left(x\right)=\lambda\quad\left(\lambda>0\right), \tag{14}$$

where each parameter is defined as follows:

$z_k^3\left(x\right)$: the hazard-rate for assumption A3,

D_1: the initial hazard-rate for the first software failure of the normal fault,

c_1: the decrease coefficient for hazard-rate for the normal fault,

D_2: the initial hazard-rate for the first software failure of the other fault,

c_2: the decrease coefficient for hazard-rate for the other fault,

p^1: the weight parameter for $z_k^1\left(x\right)$,

$z_k^4\left(x\right)$: the hazard-rate for assumption A4,

λ: the hazard-rate for the fault caused by imperfect debugging,

p: the weight parameter for $z_k^3\left(x\right)$.

Equation (13) represents the integrated hazard-rate for a software failure-occurrence phenomenon for the latent fault in software before the release, which is our proposed model. On the other hand, Equation (14) represents the hazard-rate for a software failure-occurrence for the fault caused by imperfect debugging. We assume that the fault caused by imperfect debugging is caused randomly. For that reason, we adapt exponential distribution to A4.

10. Numerical Example

In order to verify the possibility of application of proposed model to hazard-model considering imperfect debugging, we estimate the MTBF of fault big data in Apache HTTP Server as well. The parameters of proposed model considering the imperfect debugging have been estimated by MLE. The estimated value of parameters is shown as follows:

$$\hat{p}=0.96006,\quad \widehat{w_3}=0.24426,\quad \widehat{c_1^{k-1}}=0.99944,$$
$$\widehat{w_4}=2.77419,\quad \widehat{c_2^{k-1}}=0.99953,\quad \hat{\lambda}=32.77024.$$

where $w_3=p^1D_1$ and $w_4=\left(1-p^1\right)D_2$ are assumed for the simplification technique. We compare the proposed model adapted to imperfect debugging model with AIC of the proposed model. Figure 14 shows the estimated MTBF for the proposed model considering the imperfect debugging. Table 5 shows the value of AIC for each model.

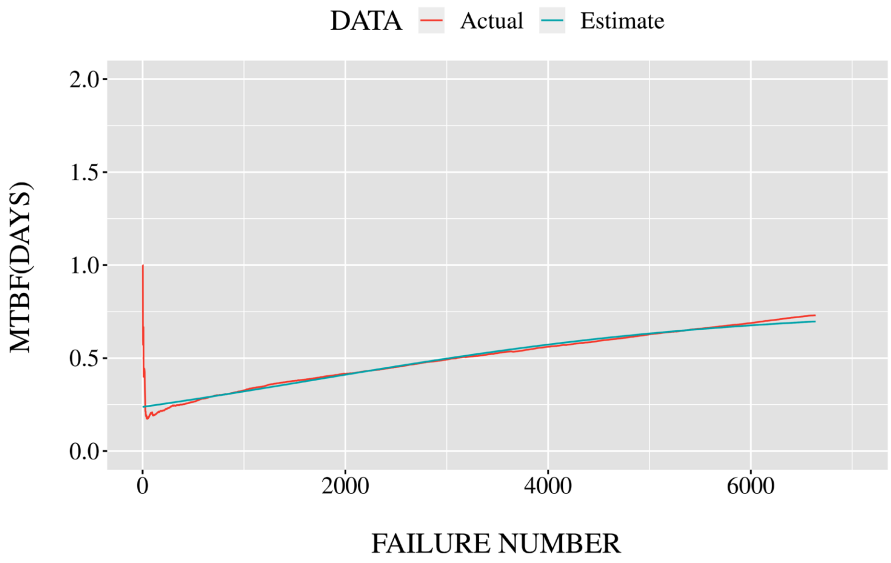

Figure 14. The estimated MTBF by using proposed model considering the imperfect debugging.

Table 5. The values of AIC for each model.

Model	AIC
Proposed model considering the imperfect debugging	13,324.8
Proposed integrated model	13,326.5

From Table 5, the proposed model for imperfect debugging fits better than proposed model in terms of AIC. In other words, it is possible to adapt the proposed model to the hazard-model considering imperfect debugging.

11. Concluding Remarks

In this paper, we have assumed that there are different trends on each fault severity level in terms of MTBF and analyzed the fault big data in BTS. We have found the difference of trend on each fault severity level in terms of MTBF and we proposed a hazard rate model considering fault severity level for OSS from the result of analysis. Proposed integrated model fits better than the typical hazard-rate models in terms of AIC. Moreover, we adapted our integrated proposed model to the hazard-rate model considering imperfect debugging. The proposed model considering imperfect debugging fits better than proposed integrated model in terms of AIC.

OSS is used by many organizations because of low cost, standardization and quick release. However, the quality of OSS is not good because of the unique development style. The quality of OSS is necessary to depend on the demand of users in the future. At the same time, it is very important to propose the software reliability model for OSS. Especially, software reliability models considering imperfect debugging are very practical for the actual situation in software development.

In the future, it is necessary to verify the applicability of proposed model because the data set in Apache HTTP server is the only one by which we evaluate the goodness-of-fit of proposed model. In this paper, we compared our proposed models to J-M model, Moranda model and Xie model. However, these models are very old ones. There are a lot of software reliability models that have ever proposed so far, therefore, we have to compare our proposed models to other hazard-rate models to evaluate the performance of our proposed models. Also, we consider the proposal of software reliability model for OSS from the other perspective.

Acknowledgments

This work was supported in part by the JSPS KAKENHI Grant No. 20K11799 in Japan.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Tamura, Y. and Yamada, S. (2016) Comparison of Big Data Analyses for Reliable Open Source Software. *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*, Bali, 4-7 December 2016, 1345-1349. https://doi.org/10.1109/IEEM.2016.7798097

[2] Yamada, S. and Tamura, Y. (2016) OSS Reliability Measurement and Assessment, Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-31818-9

[3] Tamura, Y. and Yamanda, S. (2010) Software Reliability Analysis with Optimal Release Problems Based on Hazard Rate Model for an Embedded OSS. 2010 *IEEE International Conference on Systems*, *Man and Cybernetics*, Istanbul, 10-13 October 2010, 720-726. https://doi.org/10.1109/ICSMC.2010.5641839

[4] Tamura, Y., Nobukawa, Y. and Yamada, S. (2015) A Method of Reliability Assessment Based on Neural Network and Fault Data Clustering for Cloud with Big Data. *Proceedings of the* 2*nd International Conference on Information Science and Security*, Seoul, 14-16 December 2015, 1-4. https://doi.org/10.1109/ICISSEC.2015.7370965

[5] Aljahdali, S.H., Sheta, A. and Rine, D. (2001) Prediction of Software Reliability: A Comparison between Regression and Neural Network Non-Parametric Models. *Proceedings ACS/IEEE International Conference on Computer Systems and Applications*, Beirut, 25-29 June 2001, 470-473. https://doi.org/10.1109/AICCSA.2001.934046

[6] Yamada, S. (2014) Software Reliability Modeling: Fundamentals and Applications. Springer-Verlag, Tokyo/Heidelberg.

[7] Tamura, Y., Matsumoto, M. and Yamada, S. (2016) Software Reliability Model Selection Based on Deep Learning. *Proceedings of the International Conference on Industrial Engineering, Management Science and Application* 2016, Jeju Island, 23-26 May 2016, 1-5. https://doi.org/10.1109/ICIMSA.2016.7504034

[8] Tamura, Y. and Yamada, S. (2005) Comparison of Software Reliability Assessment Methods for Open Source Software. *Proceedings of the* 11*th International Conference on Parallel and Distributed Systems*, Vol. II, Fukuoka, 20-22 July 2005, 488-492.

[9] Tamura, Y., Sone, H., Sugisaki, K. and Yamada, S. (2018) Effort Analysis of OSS Project Based on Deep Learning Considering UI/UX Design. *Proceedings of the IEEE International Conference on Reliability, Infocom Technology and Optimization*, Noida, 29-31 August 2018, 1-6. https://doi.org/10.1109/ICRITO.2018.8748408

[10] Schick, G.J. and Wolverton, R.W. (1978) An Analysis of Competing Software Reliability Models. *IEEE Transactions on Software Engineering*, **SE**-**4**, 104-120. https://doi.org/10.1109/TSE.1978.231481

[11] Jelinski, Z. and Moranda, P.B. (1972) Software Reliability Research. In: Freiberger, W., Ed., *Statistical Computer Performance Evaluation*, 465-484, Academic Press, New York, 465-484. https://doi.org/10.1016/B978-0-12-266950-7.50028-1

[12] Sandoqa, I., Alzghoul, F., Alsawalqah, H., Alzghoul, I., Alnemer, L. and Akour, M, (2016) Statistical Debugging Effectiveness as a Fault Localization Approach: Comparative Study. *Journal of Software Engineering and Applications*, **9**, 412-423. https://doi.org/10.4236/jsea.2016.98027

[13] MDN Web Docs (2021) BugDetails. https://developer.mozilla.org/

[14] The Apache Software Foundation (2021) The Apache HTTP Server Project. https://bz.apache.org/bugzilla/

[15] Yamada, S. and Sera, K. (1999) Imperfect Debugging Models with Two Kinds of Software Hazard Rate and Their Bayesian Formulation. *The IEICE Transactions*, **J82**-**A**, 1577-1584.