ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 8(34), DOI: 10.17485/ijst/2015/v8i34/85290, December 2015* ISSN (Online) : 0974-5645

The Property of Learning effect based on Delayed Software S-Shaped Reliability Model using Finite NHPP Software Cost Model

Hee-Cheul Kim*

Department of Industrial and Management Engineering, Namseoul University, Republic of Korea; kim1458@nsu.ac.kr

Abstract

Software testing in the debugging process is more efficient if it knows, in advance, to reduce costs in terms of changes in the software reliability and testing costs. In this process, essential items for software development are the reliability, cost, and consideration of release time. In this paper, the software reliability growth cost model based on Non-Homogenous Poisson Process (NHPP) about the property of learning effect for delayed software s-shaped reliability model was proposed, that was considered the actual number of faults removed in the software operation period after release time. Then discuss the relationship between release time and software testing-effort for determine the optimal release time so that can minimize the cost of software development and get more profit. Finally, according to some empirical studies, directly explain the effect of different parameters about the optimal software release time during testing process.

Keywords: Learning Effect, NHPP, Software Cost Model, S-Shaped Reliability Model

1. Introduction

The failure of computer systems from software failures may cause great losses. Therefore, an important issue during the software development process can be software reliability. In order to reflect user requirements about software reliability, the experiment for the cost of testing continue to be conducted. Software testing in the debugging process is more efficient if it knows, in advance, to reduce costs in terms of changes in the software reliability and testing costs. In this process, essential items for software development are the reliability, cost, and consideration of release time.

Eventually, the effort for predict the cause of a defect during the software product development is needed. Many software reliability models have been proposed considering the development costs. In the field, an excellent model[1,2] in terms of the error discovery process using Non-Homogenous Poisson Process (NHPP). In this model, if a fault occurs, immediately remove in the debugging process and have the assumption that no new fault has occurred.

In this field, enhanced non-homogenous Poisson Process model was presented by Gokhale and Trivedi[1]. Goel and Okumoto[2] were presented an exponential software reliability models. In this model, mean value function used to the total number of defects have S-shaped or exponential-shaped pattern. The generalized model relies about these models, delayed S-shaped reliability growth model and inflection S-shaped reliability growth model were proposed by Yamada and Ohba[3]. Zhao[4] proposed a software reliability problems in change point and Shyur[5] using the generalized reliability growth models proposed. Pham and Zhang[6] testing measured coverage, the stability of model from software stability can be evaluated.

* *Author for correspondence*

In this software reliability field, Huang[7], generalized logistic testing-effort function and the change-point parameter caused by incorporating efficient techniques to predict software reliability, were presented. Kuei-Chen[8] can be explained the learning process that software managers to become familiar with the software and test tools for S-type model. In addition, Kim[9] was studied about the comparative study of NHPP delayed S-shaped and extreme value distribution software reliability model caused by the perspective of learning effects and Shin and Kim[10] were studied on the comparative study of software optimal release time from NHPP software reliability model based on exponential and log shaped type from the perspective of learning effect.

In this paper, the software reliability growth cost model about the property of learning effect based on delayed software s-shaped reliability model was proposed caused by Non-Homogenous Poisson Process (NHPP).

The maximum likelihood estimation and bisection method used to estimate the parameters. Additionally, the model selection for the sake of efficient model used to mean square error and coefficient of determination was employed.

2. Assistance Works

2.1 NHPP Model

This is a classification of time category[1,8] for software reliability models have assumption which software failures induce the behavior based on Non-Homogeneous Poisson Process (NHPP). For parameter of the stochastic process, $\lambda(t)$ denotes the failure intensity at time t. Using $N(t)$ (denotes the cumulative number of faults) and $m(t)$ $m(t)$ (denotes its expectation), $m(t) = E[N(t)]$ and $\lambda(t)$ can be expressed as follows:

$$m(t) = \int_0^t \lambda(s)ds \tag{1}$$

And,

$$\frac{dm(t)}{dt} = \lambda(t) \tag{2}$$

Because $N(t)$ was known to have Poisson Probability Density Function (PDF) with parameter $m(t)$, can be expressed as follows[9]:

$$p(N(t) = n) = \frac{[m(t)]^n}{n!} e^{-m(t)}, n = 0,1,\cdots,\infty, \tag{3}$$

These time category models about the NHPP process can be described by the probability of failure. This model is consist of the failure intensity function (failure occurrence rates per fault) $\lambda(t)$ and mean value the function $m(t)$.

The NHPP models can be classified as finite failure and infinite failure categories.

The finite failure NHPP model have assumption that the expected number of faults detected given infinite amount of testing time will be finite, Also the infinite failures model have assumption that an infinite number of faults would be detected in infinite testing time[1]. Eventually, the failure NHPP[11] used to General Order Statistics (GOS) and infinite NHPP used to Record Value Statistics (RVS).

Let θ denotes the expected number of faults that would be detected given finite failure NHPP models. Then, the mean value function of the finite failure NHPP models can also be written as:

$$m(t) = \theta F(t) \tag{4}$$

In Equation (4), $F(t)$ denotes Cumulative Distribution Function (CDF). From the finite failure NHPP models, the failure intensity function $\lambda(t)$ is known to:

$$\lambda(t) = \theta F'(t) = \theta f(t) \tag{5}$$

Also, in Equation (5), $f(t)$ denotes Probability Density Function (PDF).

This can be materialized as:

$$\lambda(t) = [\theta - m(t)] \frac{F'(t)}{1 - F(t)} = [\theta - m(t)] h(t) \tag{6}$$

In equation (6), $h(t)$denotes the failure occurrence rate per fault of the software. The quantity [θ-$m(t)$], denotes the expected number of remaining faults in the software at time, t has pattern of monotonically non-increasing function[3]. For the property of the overall failure intensity, $\lambda(t)$ is caused by the property of failure occurrence rate per fault $h(t)$. The failure occurrence rate per fault ($h(t)$) has pattern constant or increasing, decreasing. In this section, describe some of the finite failure NHPP models using their hazard functions[1].

Let $\{t_n$, n = 1,2,...$\}$denote the sequence of times between successive software failures and t_n denotes the time between $(n\text{-}1)^{th}$and n^{th} failure.

Let $x_n$denotes failure time n, can be summarized as follows:

$$x_n = \sum_{i=1}^{n} t_i \tag{7}$$

The joint density or the likelihood function of x_1 , x_2 ,..., x_n is known as[1,10]:

$$f_{X_1, X_2, \cdots, X_n}(x_1, x_2, \ldots, x_n) = e^{-m(x_n)} \prod_{i=1}^{n} \lambda(x_i) \quad (8)$$

For a given sequence of software failure times (x_1, x_2 ,..., x_n) that are observations of the random variables (X_1 X_2 ,..., x_n), the parameter estimation of the software reliability models was performed using the Maximum Likelihood Method (MLE)[10].

2.2 Learning Factor using Cumulative and Intensity Function

In software testing implementation process, learning effects are testing by admin can be the same or manipulation of these effects is an important process.

The influential factors consist of autonomous errors-detected factor γ and learning factor η for the finding software errors.

As a result , if $f(t)$ is the intensity function that denotes the fraction of the errors detected at time t and $F(t)$ is t cumulative distribution function that denotes the fraction of the errors detected within time $(0,t]$, 1-$F(t)$ is the fraction number of the errors as yet undetected failure at time t.

The applying model using the influence factors is known as follows[8].

$$f(t) = (\gamma + \eta F(t))(1 - F(t)) \quad (9)$$

In Equation (9), note that $\gamma > 0$, $\eta > 0$.

The factor of testing staff/software developers spontaneously find software errors, which they was unaware is autonomous errors-detected factor. Also, the learning factor means that the testing staff/software developers deliberately set out to find software errors in software system for finding software errors from faults which were previously detected. Of course, both factors can be increased the efficiency in terms of software debugging. Specifically, in Equation (9), the hazard function can be derived as follows.

$$h(t) = \frac{f(t)}{1 - F(t)} = (\gamma + \eta F(t)) \quad (10)$$

In Equation (10), CDF (the Cumulative Probability Density) and PDF (Probability Density Function) can be derived as follows.

$$F(t) = \frac{h(t) - \gamma}{\eta}, f(t) = F'(t) = \frac{h'(t) - \gamma}{\eta} \quad (11)$$

2.3 Software Development Cost Model

Software cost model is defined as follows[12,13].

$$E = E_1 + E_2 + E_3 + E_4 = E_1 + C_2 \times t + C_3 \times m(t) + C_4 \times [m(t + t') - m(t)] \quad (12)$$

In Equation (12), all items means following contents

E: The expected total cost during all the software development cycle;

E_1: The cost of software design and initial software development rely on analyzing data, the amount of man power, CPU time and so on. Its' value is constant;

E_2: The cost of software testing; in other words, $E_2 = C_2$ xt, where C_2 denotes the cost per unit time and t denotes time.

E_3: The cost of removing a fault, that consists of activities like detecting the underlying fault and removing the fault. It is related to the software reliability modeling; in other words, $E_3 = C_3$ x $m(t)$,

Note that C_3 denotes the cost of removing a fault in the testing phase, $m(t)$ denotes the expected number of faults detected to the time t.

E_4: The cost of fixing a failure, during the operational phase that is also related to the software reliability modeling; ultimately,

$E_4 = C_4$ x[$m(t$+t') - $m(t)$], value of C_4 denotes the cost of fixing a fault which is observed by users in the software operational phase after the software release and t ' denotes the time during operating and maintaining the software after releasing the software system.

In reality, the value of C_4 is much larger than value of C_2 and C_3.

As a result, optimal software release time t can be get according to the following equation:

$$E' = (E_1 + E_2 + E_3 + E_4)' = C_2 + C_3 \times m'(t) + C_4 \times [m'(t + t') - m'(t)] = 0 \quad (13)$$

3. Proposed Software Reliability Model using the Perspective of Learning effects from Delayed Software S-Shaped Reliability Model

In this section, apply delayed S-shaped model. It is known,

as follows: a finite failure NHPP intensity function and mean value function[3,14].

$$\lambda(t) = \theta F'(t) = \theta \beta_1^2 t e^{-\beta_1 t} \ (\theta, \beta_1 > 0) \tag{14}$$

$$m(t) = \theta F(t) = \theta[(1-(1+\beta_1 t)e^{\beta_1 t}] \tag{15}$$

Note that $F(t)$ θ means the expected number of fault and β_1 is shape parameter in delayed S-shaped model. Using Equation (14) and (15), the hazard function is modified as follows.

$$h(t) = \frac{F'(t)}{1-F(t)} = \frac{f(t)}{1-F(t)} = \frac{\beta_1^2 t}{1+\beta_1 t} \tag{16}$$

Finally, the cumulative probability and density probability function using Equation (11) can be derived as follows.

$$F(t) = \frac{\left(\frac{\beta_1^2 t}{1+\beta_1 t}\right) - \gamma}{\eta} \quad f(t) = F'(t) = \frac{\beta_1^2}{\eta(1+\beta_1 t)^2} \tag{17}$$

The mean value function and intensity function about finite failure NHPP model considering learning effects, can be derived as follows using Equation (4), (5) and (17).

$$m(t) = \theta F(t) = \theta F(t) = \theta[\frac{\left(\frac{\beta_1^2 t}{1+\beta_1 t}\right) - \gamma}{\eta}] = \theta\left(\frac{\beta_1^2 t - \gamma - \beta_1 t}{\eta(1+\beta_1 t)}\right) \tag{18}$$

$$\lambda(t) = \theta F'(t) = \theta f(t) = \frac{\theta \beta_1^2}{\eta(1+\beta_1 t)^2} \tag{19}$$

In this case, the likelihood function, substituting (17) for (9) equation, is as follows.

$$L_{NHPP}(\Theta \mid D_{x_n}) = \left[\prod_{i=1}^{n} \theta\left(\frac{\beta_1^2}{\eta(1+\beta_1 x_i)^2}\right)\right] \bullet \exp\left[-\theta\left(\frac{\beta_1^2 x_n - \gamma - \beta_1 x_n}{\eta(1+\beta_1 x_n)}\right)\right] \tag{20}$$

The parameter estimation used to Maximum Likelihood Estimation (MLE) method. The log likelihood function, for the maximum likelihood estimation using Equation (20), is expressed as follows.

$$\ln L_{NHPP}(\Theta \mid D_{x_n}) = n\ln\theta + 2n\ln\beta_1 - n\ln\eta - 2\sum_{i=1}^{n}\ln(1+\beta_1 x_i) - \theta\left(\frac{\beta_1^2 x_n - \gamma - \beta_1 x_n)}{\eta(1+\beta_1 x_n)}\right) \tag{21}$$

Therefore, using Equation (21), MLE $\hat{\theta}_{MLE}$ *and* $\hat{\beta}_{MLE}$ can be estimated as follows:

$$\frac{\partial \ln L_{NHPP}(\Theta \mid \underline{x})}{\partial \theta} = \frac{n}{\theta} - \left(\frac{\beta_1^2 x_n - \gamma - \beta_1 x_n}{\eta(1+\beta_1 x_n)}\right) = 0 \tag{22}$$

This can be estimated as:

$$\hat{\theta} = \frac{\eta(1+\beta_1 x_n)}{\beta_1^2 x_n - \gamma - \beta_1 x_n}$$

$$\frac{\partial \ln L_{NHPP}(\Theta \mid \underline{x})}{\partial \beta_1} = \frac{2n}{\beta_1} - 2\sum_{i=1}^{n}\left(\frac{x_i}{1+\beta_1 x_i}\right) - \frac{\theta}{\eta}\left(\frac{x_n(\beta_1^2 x_n - 1 - \gamma + 2\beta_1)}{(1+\beta_1 x_n)^2}\right) = 0$$

$$Note.\ \underline{x} = (x_1, x_2, x_3, \ldots x_n) \tag{23}$$

4. Illustration

MLE considering influential factors In this chapter, using software failures time data[15], it is to analyze the characteristics of learning factors. This data set in Table 1 lists and in order to analyze the trust models presented in the first data set should be preceded by a trend test[15].

In this paper, the Laplace trend test analysis is used for reliability growth property. As a result of this test in this Figure 1, as indicated in the Laplace factor is between 2 and -2, data set shows the reliability growth property. Thus, using this data, it is possible to estimate the reliability[10,16]. According to modifying the value of different parameters, the effect of various parameters on the optimal software release time directly can be estimated.

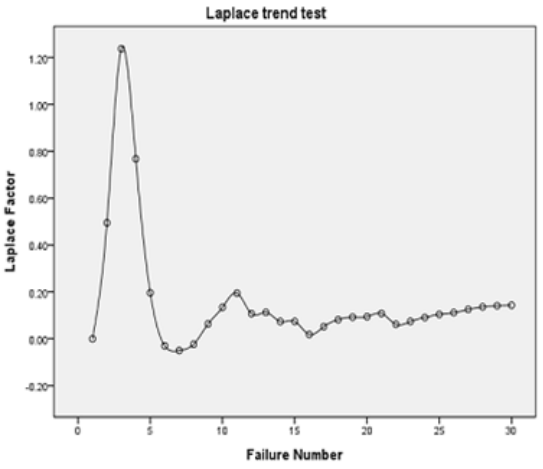

Figure 1. Laplace trend test.

According to the parameters[13], the following conditions[12] were applied:

From assumption (A), considering E_1 = 15, C_2 =0.1, C_3 =0.5, C_4 =1.5, t'=100, ɡ=0.07 and f=0.03, it is possible to obtain the following results:

Table 1. Software Failure time data

Failure Number	Failure Interval (second)	Failure Time (second)
1	0.479	0.479
2	0.266	0.745
3	0.277	1.022
4	0.554	1.576
5	1.034	2.610
6	0.949	3.559
7	0.693	4.252
8	0.597	4.849
9	0.117	4.966
10	0.170	5.136
11	0.117	5.253
12	1.274	6.527
13	0.469	6.996
14	1.174	8.170
15	0.693	8.863
16	1.908	10.771
17	0.135	10.906
18	0.277	11.183
19	0.596	11.779
20	0.757	12.536
21	0.437	12.973
22	2.230	15.203
23	0.437	15.64
24	0.340	15.98
25	0.405	16.385
26	0.575	16.96
27	0.277	17.237
28	0.363	17.600
29	0.522	18.122
30	0.613	18.735

Table 2. MLE considering influential factors

·	·	$\hat{\theta}_{MLE}$	$\hat{\beta}_{1MLE}$
0.03	0.07	31.6205	1.0361
0.05	0.05	32.2317	1.0260
0.07	0.03	32.5188	1.0158

As shown in Figure 2, the growth curve of the proposed model firstly decreases and then increases.

This situation means that the number of residual faults in software system is less and less in the process of fault removing. The reason is that probability of the remaining faults observed by users after software release is lower and lower. In the early phase of testing, there are still many faults in software which are easily detected and removed, and the cost of removing a fault in this phase is far lower than that of removing a fault in the operation phase. Therefore, the total cost of software decreases during the debugging process of faults. But in the latter phase the number of faults remaining in software is already less. Therefore, in this testing phase the time of detecting a fault is very long and the cost of removing a fault becomes higher than that in the operation phase, thus, the cost curve increases constantly with time. From the trend of the cost curve, the optimal software release time can be estimated and this situation is also the most realistic property. Most cases are consistent with this in the process of actual software development[12].

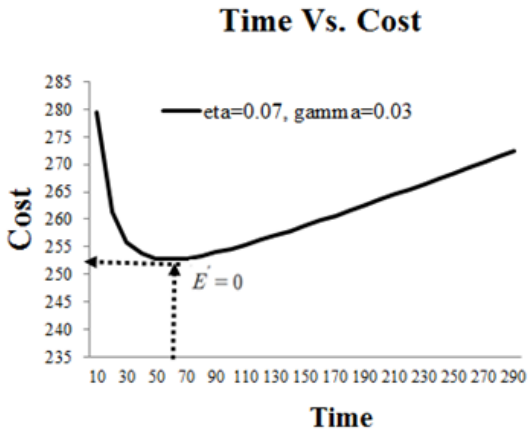

Figure 2. The curve under the condition of (A).

From assumption (B) under other parameters are the same conditions, Applying the changed value of influential factors from g=0.07 and f=0.03 to g=0.05 and f=0.05, get obtain the curve in Figure 3.

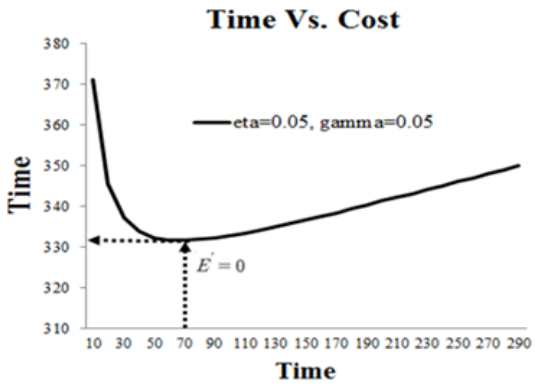

Figure 3. The curve under the condition of (B).

Comparing the different curve in Figure 2 and Figure 3, it can be interpreted that the optimal software release time has been postponed. This affairs happens because the value of influential factors, case of the increase in the autonomous errors-detected factor and reduce the learning factor. Thus, the learning factor should be increased in the operation phase. Under this situation, the testing time should be extended to make sure learning factor on a certain lever after the software release. It can be interpreted that if high learning factor can be efficient work in terms of cost.

From assumption (C), other parameters are the same conditions, case of value of influential factors with g=0.03 and f=0.07, get obtain the curve in Figure 4.

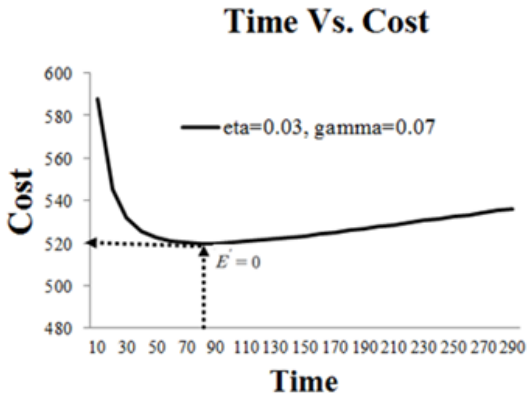

Figure 4. The curve under the condition of (C).

In terms of comparing the different curve in Figure 2 and Figure 4, it can be be interpreted that the optimal software release time has been postponed. This affairs happens because the value of influential factors, case of the increase from the autonomous errors-detected factor and leads to reduced impact for the learning factor. Thus, the learning factor should be increased during the operation phase. Under this affair, the testing time should be extended to make sure learning factor on a certain lever after the software release.

It can be interpreted that low autonomous errors-detected factor cannot be efficient work in terms of cost.

5. Conclusion

Software reliability growth model can be estimated the optimal software release time and the cost considering the testing efforts. More accurate model is needed to decrease the testing cost and increase the profit of releasing software. The use of software cost model can help predict the optimal software release time accurately. In terms of comparison, the proposed model takes into account the total number of faults discovered by users during the software operation period or software maintenance after its release. It can be interpreted that the cost of actual fault debugging is lower than the cost of removing all remaining faults in the operation phase can be effective. Thus, estimation of the optimal software release time is more realistic. In further studies, need to check the validity and effectiveness of our proposed software reliability growth model and the software cost model under the modeling framework by using much actual failure data. Generally, when learning factor is the highest and autonomous errors-detected factor is the lowest, model was effective depressing. Therefore, in this paper, the proposed model can be used as an alternative model in this field. These studies can be used as prior information to the software manager.

6. Acknowledgement

Funding for this paper was provided by Namseoul University.

7. References

1. Gokhale SS, Trivedi KS. A time/structure based software reliability model. Annals of Software Engineering. 1998; 8:85–12.
2. Goel AL, Okumoto K. Time dependent error - detection rate model for software reliability and other performance measure. IEEE Trans Reliability. 1979; 28(3):206–11.
3. Yamada S, Ohba H. S-shaped software reliability modeling for software error detection. IEEE Trans Reliability. 1983; 32:475–84.
4. Zhao M. Change-point problems in software and hardware reliability. Communication Stat Theory Methods. 1993; 22(3):757–68.
5. Shyur H-J. A stochastic software reliability model with imperfect debugging and change-point. J Syst Software. 2003; 66(2):135–41.
6. Pham H, Zhang X. NHPP software reliability and cost models with testing coverage. Eur J Oper Res. 2003; 145:445–54.
7. Huang C-Y. Performance analysis of software reliability growth models with testing-effort and change-point. J Syst Software. 2005; 76:181–94.
8. Kuei-Chen C, Yeu-Shiang H, Tzai-Zang L. A study of software reliability growth from the perspective of learning effects. Reliability Engineering and System Safety. 2008; 93:1410–21.

9. Kim H-C. The comparative study of NHPP delayed S-shaped and extreme value distribution software reliability model using the perspective of learning effects. International Journal of Advancements in Computing Technology (IJACT). 2013; 5(9):1210–8.
10. Shin H-D, Kim H-C. The comparative study of software optimal release time based on NHPP software reliability model using exponential and log shaped type for the perspective of learning effects. International Journal of Advancements in Computing Technology. 2013; 5(12):120–9.
11. Kuo L, Yang TY. Bayesian computation of software reliability. Journal of the American Statistical Association. 1996; 91:763–73.
12. Zhang Y, Wu K. Software cost model considering reliability and time of software in use. Journal of Convergence Information Technology . 2012; 7(13):135–42.
13. Satya Prasad R, Rao KRH, Kantha RRL. Software reliability measuring using modified maximum likelihood estimation and SPC. International Journal of Computer Applications (0975–8887). 2011; 21(7):1–5.
14. Alaa S. Parameter estimation of software reliability growth models by particle swarm optimization. AIML Journal. 2007; 7(1):55–61.
15. Hayakawa Y, Telfar G. Mixed poisson-type processes with application in software reliability. Mathematical and Computer Modelling. 2000; 31:151–6.
16. Kanoun K, Laprie JC, Lyu MR. Handbook of software reliability engineering. Chapter Trend Analysis. McGraw-Hill, New York: NY. 1996. p. 401–37.