Software Reliability Growth Models Incorporating Fault Dependency with Various Debugging Time Lags

Chin-Yu Huang[1], Chu-Ti Lin[1], Sy-Yen Kuo[2], Michael R. Lyu[3], and Chuan-Ching Sue[4]

[1]*Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan.*
[2]*Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan.*
[3]*Computer Science and Engineering Department, The Chinese University of Hong Kong, Shatin, Hong Kong.*
[4]*Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan.*

Abstract

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. Over the past 30 years, many software reliability growth models (SRGMs) have been proposed and most SRGMs assume that detected faults are immediately corrected. Actually, this assumption may not be realistic in practice. In this paper, we first give a review of fault detection and correction processes in software reliability modeling. Furthermore, we will show how several existing SRGMs based on NHPP models can be derived by applying the time-dependent delay function. On the other hand, it is generally observed that mutually independent software faults are on different program paths. Sometimes mutually dependent faults can be removed if and only if the leading faults were removed. Therefore, here we incorporate the ideas of fault dependency and time-dependent delay function into software reliability growth modeling. Some new SRGMs are proposed and several numerical examples are included to illustrate the results. Experimental results show that the proposed framework to incorporate both fault dependency and time-dependent delay function for SRGMs has a fairly accurate prediction capability.

1. Introduction

Dramatic advances in software technologies have greatly promoted the growth of computer applications. More and more critical applications, such as banking payment systems, credit card and shared ATM Systems, etc., are being developed. The software for these applications is becoming increasingly complex and sophisticated. Thus reliability will become the main goal for software developers. Software reliability is often defined as the probability of failure-free software operation for a specified period of time in a specified environment [1]. Over the past 30 years, many *Software Reliability Growth Models* (SRGMs) have been proposed for estimation of reliability growth of products during software development processes [2-6]. From our studies, we find that many papers consider an NHPP as a stochastic process to describe the fault process and reliability growth of most SRGMs is expressed as exponential curve [7].

On the other hand, Ohba [2, 8-9] proposed an inflected S-shaped model to describe the software failure-occurrence phenomenon with mutual dependency of detected faults. He thought that the exponential SRGM was sometimes insufficient and inaccurate to analyze actual software failure data for reliability assessment. Moreover, Yamada et al. [7, 10-11] also presented a delayed S-shaped SRGM incorporating the time delay between fault detection and fault correction. Actually, Ohba conceived that there were two types of faults in a software system: mutually independent faults and mutually dependent faults [8]. The mutually independent faults are on different program paths. Mutually dependent faults can be removed if and only if the leading faults are removed. Latter, Kapur et al. [12-13] proposed an SRGM that took care of the underlying fault dependency. They considered that in a software system, the fault removal depended on the previously removed faults and that would result in a delay of the fault removal process.

One common assumption of conventional SRGMs is that detected faults are immediately removed. In practice, this assumption may not be realistic in software development. We know that software testing and debugging are very complex and expensive processes. The time to remove a fault depends on the complexity of the detected faults, the skills of the debugging team, the available manpower, or the software development environment, etc. Therefore, the time delayed by the detection and/or correction process should not be negligible.

There are some papers that have addressed the problem of delayed fault correction time [14-24]. For example, Schneidewind [15-17] proposed an approach to model the fault-correction process by using a constant delayed fault-detection process. He assumed that the rate of fault correction was proportional to the rate of failure detection. However, if this assumption is not met in practice, the model will underestimate the remaining faults in the code [20]. Later, Xie and Zhao [18, 20] pointed out that this

Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)
0730-3157/04 $20.00 © 2004 **IEEE**

assumption was too restrictive. They extended the Schneidewind model to a continuous version by substituting a time-dependent delay function for the constant delay. Moreover, Goševa-Popstojanova and Trivedi [21] presented a software reliability modeling framework based on Markov renewal process, which incorporated the possible s-dependence among successive software runs, number of runs between failures and occurrence time of failure.

In this paper, we first give a review of fault detection and correction processes in software reliability growth models. Furthermore we show how several existing SRGMs based on NHPP models can be derived by applying the time-dependent delay function. On the other hand, it is probability that mutually independent software faults are on different program paths and mutually dependent faults can be removed if and only if the leading faults were removed. Thus we will incorporate the ideas of failure dependency and time-dependent delay function into software reliability growth modeling.

The rest of the paper is organized as follows. Section 2 gives a brief review of characteristics of the NHPP models with delayed correction process and shows how some existing NHPP models can be reinterpreted from a viewpoint of delayed correction process. We consider failure dependency in software reliability assessment in Section 3. Furthermore, we will introduce how to incorporate the ideas of failure dependency and time-dependent delay function into software reliability growth modeling. The experiments and numerical results are presented in Section 4. Finally, the concluding remarks are given in Section 5.

2. Reviews of fault detection and correction processes in software reliability growth models

Most SRGMs have some basic assumptions concerning the software error-detection process [2, 4-5, 7]:

(1) The fault removal process follows the Non-homogeneous Poisson Process (NHPP).
(2) The software system is subject to failures at random times caused by the manifestation of remaining faults in the system.
(3) All faults are independent and equally detectable.
(4) Each time a failure occurs, the fault that caused it is immediately and perfectly removed. A detected error is removed with certainty and correction of errors takes only negligible time. No new faults are introduced.

It is noted that the assumption (4) assumes that detected faults are immediately removed. In fact, this assumption may not be realistic in practice. In general, finding a fault during testing is one thing and fixing it is another, and often there is a considerable time delay between the two. Therefore, the time delayed by the correction process is not negligible. Schneidewind [15-17] ever modeled the fault-correction process by using a delayed fault-detection process. He assumes that the fault-detection process follows the NHPP and the rate of change of the mean value function (MVF) is exponentially decreasing. Under the above assumption, it is shown that the fault detection process can be modeled by an NHPP with exponentially decreasing intensity function $\rho(i)$, i.e.,

$$\rho(i) = \alpha \exp[-\beta\, i], \quad \alpha > 0, \quad \beta > 0, \tag{1}$$

where α and β are the parameters of the model [18]. Therefore, the MVF of fault detection process is given by

$$m_{\rho}(i) = (\alpha / \beta)(1 - \exp[-\beta\, i]). \tag{2}$$

Xie and Zhao [18, 20] explain that Schneidewind assume the rate of fault correction is proportional to the number of fault detected and it lags fault detection process by a constant delay ε_i. That is, the MVF is depicted as

$$m(i - \varepsilon_i) = (\alpha / \beta)(1 - \exp[-\beta(i - \varepsilon_i)]), \quad i \geq \varepsilon_i. \tag{3}$$

Obviously, the fault-detection process in the Schneidewind model is isomorphic to the Goel-Okumoto model, except the Goel-Okumoto model is viewed as a continuous-time process [20]. Xie and Zhao pointed out that this assumption is too restrictive and they extended the Schneidewind model to a continuous version by substituting a time-dependent delay function for the constant delay (ε_i) [18, 20]. That is, Eq. (2) and Eq. (3) can be changed as

$$m(t) = (\alpha / \beta)(1 - \exp[-\beta\, t]) \tag{4}$$

and

$$m(t - \varepsilon_t) = (\alpha / \beta)(1 - \exp[-\beta(t - \varepsilon_t)]), \quad t \geq \varepsilon_t. \tag{5}$$

In fact, most existing SRGMs can be reinterpreted as delayed fault-detection models that can model the software fault detection and correction processes. Therefore, we can remove the impractical assumption that the fault-correction process is perfect and establish a corresponding time-dependent delay function to fit the fault-correction process.

Definition 1: Given a fault-detection and fault-correction process, one defines the delay-effect factor, $\varphi(t)$, to be a time-dependent function that measures the expected delay in correcting a detected fault at any time.

Definition 2: An SRGM is called a delayed-time NHPP model if it obeys the following assumptions:

(1) The fault detection process follows the NHPP.
(2) The software system is subject to failures at random times caused by the manifestation of remaining faults in the system.
(3) All faults are independent and equally detectable.
(4) The rate of change of the MVF is exponentially decreasing.
(5) The detected faults are not immediately removed and it lags the fault detection process by a delay-effect factor $\varphi(t)$.

Based on the above assumptions (1)-(4), the original MVF of NHPP model is

$$m_{original}(t) = a(1 - \exp[-rt]), a > 0, r > 0, \tag{6}$$

where a is the expected number of initial faults, and r is the fault detection rate. From the assumption (5) in definition 2 and Eq. (6), the new MVF can be depicted as

$$m(t) = m_{original}(t - \varphi(t))$$
$$= a(1 - \exp[-rt]\exp[r\varphi(t)]), a > 0, r > 0. \quad (7)$$

We thus derive the following theorem.

Theorem 1: Given a delay-effect factor, $\varphi(t)$, we have [19]:

(a) The fault-detection intensity of the delayed-time NHPP SRGM is

$$\lambda(t) = dm(t)/dt$$
$$= ar\exp[-rt]\exp[r\varphi(t)] \times (1 - \frac{d\varphi(t)}{dt}), a > 0, r > 0. \quad (8)$$

(b) $d\varphi(t)/dt < 1$.

In the following, we will review three conventional SRGMs that can be directly derived from Definition 1, Definition 2, and Theorem 1. We can derive the fault-detection intensity from Eq. (8) and check the condition of Theorem 1.

- **Goel-Okumoto Model:** This model, first proposed by Goel and Okumoto [2, 4], is one of the most popular NHPP model in the field of software reliability modeling. If $\varphi(t) = 0$, then we have

$$d\varphi(t)/dt = 0 < 1 \quad (9)$$

and $m(t) = a(1 - \exp[-rt]), a > 0, r > 0$.

- **Yamada Delayed S-Shaped Model:** The Yamada Delayed S-Shaped model is a modification of the NHPP to obtain an S-shaped curve for the cumulative number of failures detected such that the failure rate initially increases and later decays [2, 4, 7, 10-11]. If $\varphi(t) = (\ln(1 + rt))/r$, then we have

$$d\varphi(t)/dt = 1/(1 + rt) < 1 \quad (10)$$

and $m(t) = a(1 - (1 + rt)\exp[-rt])$.

- **Yamada Weibull-Type Testing-Effort Function Model:** Yamada et al. [2, 7] proposed a software reliability model incorporating the amount of test-effort expended during the software testing phase. The testing-effort can be represented as the man power, number of CPU hours, or the number of executed test cases, etc. In general, the testing-effort during the testing phase and the time-dependent behavior of development effort in the software development process can be described by a Weibull curve. If $\varphi(t) = t + \alpha\exp[-\beta t^{\gamma}] - \alpha$, then we have

$$d\varphi(t)/dt = 1 - \alpha\beta\gamma t^{\gamma-1}\exp[-\beta t^{\gamma}] < 1 \quad (11)$$

and $m(t) = a\{1 - \exp[-r\alpha(1 - \exp[-\beta t^{\gamma}])]\}$.

Intuitively, the correction process can be viewed as a learning process since the software testing teams will familiar with the debugging environments and tools as time proceeds. These teams' skills can be gradually improved and thus the amount of time lag will be lesser. In other words, the delay-effect factor is non-increasing in the circumstances.

3. Considering failure dependency in software fault modeling

Assumptions [12-13]***:***

(1) The fault detection process follows the NHPP.
(2) The software system is subject to failures at random times caused by the manifestation of remaining faults in the system.
(3) The all detected faults can be categorized as leading faults and dependent faults. Besides, the total number of faults is finite.
(4) The mean number of leading faults detected in the time interval $(t, t+\Delta t]$ is proportional to the mean number of remaining leading faults in the system. Besides, the proportionality is a constant over time.
(5) The mean number of dependent faults detected in the time interval $(t, t+\Delta t)$ is proportional to the mean number of remaining dependent faults in the system and to the ratio of leading faults removed at time t and the total number of faults. Besides, the proportionality is a constant over time.
(6) The detected dependent fault may not be immediately removed and it lags the fault detection process by a delay-effect factor $\varphi(t)$. That is, $\varphi(t)$ is the time delay between the removal of the leading fault and the removal of the dependent fault(s).
(7) No new faults are introduced during the fault removal process.

Let a denotes the expected number of initial faults. Besides, a_1 is the total number of leading faults and a_2 is the total number of dependent faults detected in the software product. Therefore, from assumptions (3) & (4), we have

$$a = a_1 + a_2.$$

For the sake of convenience, in the following paragraph we will let $m(t)$ be the MVF of the expected number of faults detected in time $(0, t]$. Therefore, $m(t)$ is an increasing function of t and $m(0)=0$. Here we assume

$$m(t) = m_1(t) + m_2(t), \quad (12)$$

where $m_1(t)$ is the MVF of the expected number of leading faults detected in time $(0, t]$ and $m_2(t)$ is the MVF of the expected number of dependent faults detected in time $(0, t]$.

Consequently, if the number of detected leading faults is proportional to the number of remaining leading faults, then we obtain the following differential equation:

$$\frac{dm_1(t)}{dt} = r \times [a_1 - m_1(t)], \quad (13)$$

where a is the expected number of initial faults, and r is the fault detection rate. Solving the above differential equation under the boundary condition $m_1(t)=0$, we have

$$m_1(t) = a_1(1-\exp[-rt]).$$

Similarly, from assumptions (6) & (7), we have

$$\frac{dm_2(t)}{dt} = \theta \times [a_2 - m_2(t)] \times \frac{m_1(t-\varphi(t))}{a}. \quad (14)$$

Please note that the dependent faults can be removed only when the leading fault is perfectly removed. In the following, we will give a detailed description of possible behavior of $\varphi(t)$.

(Case 1) If $\varphi(t)=0$, Eq. (14) becomes

$$\frac{dm_2(t)}{dt} = \theta \times [a_2 - m_2(t)] \times \frac{a_1(1-\exp[-rt])}{a}. \quad (15)$$

Assuming the initial condition $m_2(0)=0$, we obtain

$$m_2(t) = a_2(1-\exp[\frac{a_1\theta(1-\exp[-rt]) - ra_1\theta\ t)}{ra}]), \quad (16)$$

where θ is the dependent fault removal rate. Here we let $a_1 = Pa$ & $a_2=(1-P)a$ (where P is the proportion of the leading faults). From Eq. (12), we obtain the MVF $m(t)$ as follows [12-13] :

$$m(t) = m_1(t) + m_2(t) = a(1-P\exp[-rt] - (1-P)\times \exp[\frac{P\theta}{r}(1-\exp[-rt]) - tP\theta]). \quad (17)$$

(Case 2) If $\varphi(t) = (\ln(1+rt))/r$, Eq. (14) becomes

$$\frac{dm_2(t)}{dt} = \theta \times [a_2 - m_2(t)] \times \frac{a_1(1-(1+rt)\exp[-rt])}{a}. \quad (18)$$

By solving the above equation under the boundary condition $m_2(0)=0$, the MVF is given by

$$m_2(t) = a_2(1-\exp[\frac{-a_1\theta(rt+2\exp[-rt]+rt\exp[-rt]-2)}{ra}]) \quad (19)$$

and $m(t) = a(1-P(1+rt)\exp[-rt] - (1-P)\times \exp[\frac{2P\theta}{r}(1-\exp[-rt]) - tP\theta(1+\exp[-rt])]). \quad (20)$

(Case 3) If $\varphi(t) = t + \alpha\exp(-\beta\ t^{\gamma}) - \alpha$, Eq. (14) becomes

$$\frac{dm_2(t)}{dt} = \theta \times [a_2 - m_2(t)] \times \frac{a_1\{1-\exp[-r\alpha(1-\exp[-\beta\ t^{\gamma}])]\}}{a}. \quad (21)$$

When γ=1 or γ=2 for Yamada's Weibull-type testing-Effort function model, we obtain the exponential or the Rayleigh curve respectively. Actually, they are special cases of the Weibull testing-effort function [12-13]. For example, if γ=1, Eq. (21) can be solved and is given by

$$m_2(t) = a_2(1- \exp[-\frac{a_1\theta\exp[-r\alpha](\exp[r\alpha]t\beta - \zeta[r\alpha] + \zeta[\exp[-t\beta]r\alpha])}{a\beta}]) \quad (22)$$

,where $\zeta[z] = -\int_{-z}^{\infty}\exp[-t]/t\ dt$.

Therefore,

$$m(t) = a(1-P\exp[-r\alpha(1-\exp[-\beta\ t])] - (1-P)\times \exp[-\frac{P\theta\exp[-r\alpha](\exp[r\alpha]\beta\ t - \zeta[r\alpha] + \zeta[r\alpha\exp[-\beta\ t]])}{\beta}]).$$

(23)

On the other hand, if γ=2, we have

$$m_2(t) = a_2\Big(1-\exp[-\int_x^0 P\theta\Big(-1+\exp[(-1+\exp[-y^2\tfrac{\beta}{2}])r\alpha]\Big)dy + \int_x^t P\theta(-1+\exp[(-1+\exp[-y^2\tfrac{\beta}{2}])r\alpha])dy]\Big), \quad (24)$$

and

$$m(t) = a\Big(1-P\exp[-r\alpha(1-\exp[-\tfrac{\beta}{2}t^2])] - (1-P)\times \exp[-\int_x^0 P\theta\Big(-1+\exp[(-1+\exp[-\tfrac{\beta}{2}y^2])r\alpha]\Big)dy + \int_x^t P\theta\Big(-1+\exp[(-1+\exp[-\tfrac{\beta}{2}y^2])r\alpha]\Big)dy\Big). \quad (25)$$

4. Numerical examples

4.1. Data description

We choose two real data sets as illustrations. The first data set (DS1) was from a study by Ohba [9]. The system was a PL/I database application software consisting of approximately 1,317,000 lines of code. During nineteen weeks, 47.65 CPU hours were consumed and about 328 software faults were removed.

The second data set (DS2) in this paper was from the technical report for the project of Reactor Vessel Level Indication System (RVLIS, a detection system used to monitor the level of water within the reactor vessel) [25]. The coding language is VersaPro 2.03 and the development platform is GE FANUC PLC 9030. It took 25 weeks to complete the test. During the test phase, 230 software faults were removed. The complete failure data is given in Table 1.

Table 1: Real software failure data set (RVLIS).

Week	CNF	Week	CNF	Week	CNF	Week	CNF
1	44	8	100	15	197	22	230
2	75	9	124	16	205	23	230
3	75	10	130	17	214	24	230
4	75	11	130	18	215	25	230
5	75	12	159	19	225		
6	75	13	175	20	227		
7	75	14	181	21	228		

CNF: Cumulative number of failures

4.2. Criteria for model's comparison

The comparison criteria we use to compare various models' performance are described as follows:

(1) The ***Noise*** is defined as [26]:

$$\sum_{i=1}^{n}\left|(r_i - r_{i-1})/r_{i-1}\right|, \quad (26)$$

where r_i is the predicted failure rate.

(2) The ***Mean Square of Fitting Error*** (MSE) is defined as [13]:

$$\sum_{i=1}^{k}[m(t_i) - m_i]^2/k, \quad (27)$$

where m_i is the observed number of faults by time t_i.

(3) The ***Mean Error of Prediction*** (MEOP) is defined as [27]:

$$\left\{\sum_{i=k}^{n}\left|n_i - m_i\right|\right\}/(n-k+1), \quad (28)$$

where n_i is the observed cumulative number of failures at time s_i and m_i is the predicted cumulative number of failures at time s_i, i=k, k+1,…, n.

4.3. Performance analysis

In this section, we will evaluate the proposed models and several existing NHPP models. Due to the limitation of space, here we only consider Eq. (20) as illustration.

4.3.1. Case 2–DS1. Firstly, all parameters of the proposed models are estimated by using the method of *least squares estimation* (LSE) or *maximum likelihood estimation* (MLE) [3-4, 6, 27]. Table 2 shows the estimated parameters of Eq. (20) and the performance comparisons of different SRGMs for DS1. It is noted that the proposed model (i.e., Eq. (20)) estimates P=0.72 for this data set. The result suggests that the software may contain two categories of faults, 72% are leading faults and 28% are dependent faults. Moreover, the possible values of P are also discussed and listed in Table 2. As seen from Table 2, the proposed model almost provides the lowest MEOP if compared to the Goel-Okumoto model and the Yamada delay S-shaped model. Overall, the MVF of proposed model provides a good fit to this data.

4.3.2. Case 2–DS2. Similarly, parameters of all selected models are estimated and the related MVFs are obtained. All selected models are compared with each other based on objective criteria. Table 3 shows the estimated parameters of Eq. (20) and the performance comparisons of different SRGMs for DS2. The proposed model estimates P=0.77 and it indicates that the software contains two categories of faults, 77% are leading faults and 23% are dependent faults. Moreover, the possible values of P are also listed in Table 3. On the other hand, we know that the inflection S-shaped model is based on the dependency of faults by postulating the assumption: some of the faults are not detectable before some other faults are removed [5]. Therefore, it may provide us some information for reference. After the simulation, we find that the estimated value of inflection rate (which indicates the ratio of the number of detectable faults to the total number of faults in the software) is 0.598 for DS2. It indicates that the growth curve is slightly S-shaped [12-13]. On the average, the proposed model performs well in this actual data.

5. Conclusions

In this paper, we incorporate both failure dependency and time-dependent delay function into software reliability assessment. Specifically, all detected faults can be categorized as leading faults and dependent faults. Moreover, the fault-correction process can be modeled as a delayed fault-detection process and it lags the detection process by a time-dependent delay. Thus the proposed delay-effect factor can be used to measure the expected time-lag in correcting the detected faults during software development. Some new SRGMs are proposed and several numerical illustrations based on two real data sets are presented. Experimental results show that the proposed framework to incorporate both failure dependency and time-dependent delay function for SRGM has a fairly accurate prediction capability.

6. Acknowledgments

This research was supported by the National Science Council, Taiwan, under Grant NSC 93-2213-E-007-088 and was also substantially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No.CUHK4360/02E). Moreover, we are thankful to Shian-Shing Shyu, Chung-Lin Lee, and Chi-Yuan Chang, Institute of Nuclear Energy Research, Atomic Energy Council, Executive Yuan, Taiwan, for providing the second data set. The authors also thank several anonymous referees for their constructive reviews and comments.

Table 2: Comparison results of different SRGMs for DS1.

Model	a	r	θ	P	MSE	MEOP	Noise
Eq. (20)	412.600	0.228869	0.090558	0.72	161.585	9.75609	2.16918
Eq. (20)	523.048	0.460176	0.283841	0.2	139.241	9.68111	1.50403
Eq. (20)	501.357	0.358750	0.192887	0.3	145.669	9.79057	1.69780
Eq. (20)	478.453	0.305156	0.147434	0.4	150.444	9.83572	1.84090
Eq. (20)	455.454	0.271896	0.121256	0.5	154.445	9.81603	1.96342
Eq. (20)	434.286	0.248901	0.104441	0.6	157.935	9.77504	2.06808
Eq. (20)	415.631	0.231742	0.092751	0.7	161.033	9.75492	2.15467
Eq. (20)	399.508	0.218213	0.084170	0.8	163.818	9.75748	2.22772
Eq. (20)	385.722	0.207095	0.077631	0.9	166.349	9.75588	2.29030
Goel-Okumoto model	760.534	0.032269	—	—	139.815	9.89065	0.60332
Yamada Delay S-shaped model	374.050	0.197651	—	—	168.673	9.78299	2.34455

Table 3: Comparison results of different SRGMs for DS2.

Model	*a*	*r*	*θ*	*P*	MSE	MEOP	Noise
Eq. (20)	264.181	0.218560	0.082480	0.77	402.515	13.2156	2.83969
Eq. (20)	330.303	0.683468	0.246725	0.2	334.808	14.6244	1.68513
Eq. (20)	313.562	0.409470	0.184855	0.3	370.289	14.2028	2.00071
Eq. (20)	301.219	0.325860	0.144944	0.4	383.379	13.9400	2.24942
Eq. (20)	290.181	0.280493	0.119105	0.5	390.808	13.7258	2.42998
Eq. (20)	279.858	0.251092	0.101601	0.6	396.008	13.5182	2.59620
Eq. (20)	270.325	0.230121	0.089176	0.7	400.082	13.3320	2.74551
Eq. (20)	261.688	0.214181	0.079991	0.8	403.478	13.1691	2.87787
Eq. (20)	253.989	0.201489	0.072978	0.9	406.418	13.0372	2.99694
Goel-Okumoto model	326.364	0.055693	—	—	253.217	12.7256	1.35427
Yamada Delay S-shaped model	247.221	0.191014	—	—	409.026	12.9325	3.10483

References

[1] American Institute of Aeronautics and Astronautics, *Recommended Practice for Software Reliability*, ANSI/AIAA R-013-1992, February 23, 1993

[2] M. Xie, *Software Reliability Modeling*, World Scientific Publishing Company, 1991.

[3] J. D. Musa, *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*, McGraw-Hill, 1999.

[4] M. R. Lyu, *Handbook of Software Reliability Engineering*, McGraw Hill, 1996.

[5] C. Y. Huang, M. R. Lyu, and S. Y. Kuo, "A Unified Scheme of Some Non-Homogenous Poisson Process Models for Software Reliability Estimation," *IEEE Trans. on Software Engineering*, Vol. 29, No. 3, pp. 261-269, March 2003.

[6] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability, Measurement, Prediction and Application*, McGraw Hill, 1987.

[7] S. Yamada, "Software Reliability Models and Their Applications: A Survey," *Proceedings of the International Seminar on Software Reliability of Man-Machine Systems*, pp. 56-80, Aug. 2000, Kyoto University, Kyoto, Japan.

[8] M. Ohba, "Infection S-Shaped Software Reliability Growth Model," *Stochastic Models in Reliability Theory*, Springer-Verlag, Berlin, pp. 144-162, 1984.

[9] M. Ohba, "Software Reliability Analysis Models," *IBM Journal of Research and Development*, Vol. 28, No. 4, pp. 428-443, 1984.

[10] S. Yamada, M. Ohba, and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection," *IEEE Trans. Reliability*, Vol. R-32, No. 5, pp. 475-478, 484, 1983.

[11] S. Yamada, M. Ohba, and S.Osaki, "S-Shaped Software Reliability Growth Models and Their Applications," *IEEE Trans. Reliability*, Vol. R-33, No. 4, pp. 289-292, 1984.

[12] P. K. Kapur and S. Younes, "Software Reliability Growth Model with Error Dependency," *Microelectronics and Reliability*, Vol. 35, No. 2, pp. 273-278, 1995.

[13] P. K. Kapur, R. B. Garg, and S. Kumar, *Contributions to Hardware and Software Reliability*, World Scientific Publishing Company, 1999.

[14] S. S. Gokhale, P. N. Marinos, M. R. Lyu, and K. S. Trivedi, "Effect of Repair Policies on Software Reliability", *Proceedings of Computer Assurance*, pp. 105-116, June 1997, Gatheirsburg, Maryland.

[15] N. F. Schneidewind, "Modeling the Fault Correction Process," *Proceedings of the 12th International Symposium on Software Reliability Engineering*, pp. 185-190, Nov. 2001, Hong Kong, China.

[16] N. F. Schneidewind, "An Integrated Failure Detection and Fault Correction Model," *Proceedings of 18th International Conference on Software Maintenance*, pp. 238-241, Oct. 2002, Montreal, Quebec, Canada.

[17] N. F. Schneidewind, "Fault Correction Profiles," *Proceedings of the 14th International Symposium on Software Reliability Engineering*, pp. 257-267, Nov. 2003, Denver, Colorado.

[18] M. Xie and M. Zhao, "The Schneidewind Software Reliability Model Revisited," *Proceedings of the 3rd International Symposium on Software Reliability Engineering*, pp. 184-192, Oct. 1992, Research Triangle Park, North Carolina.

[19] J. H. Lo, S. Y. Kuo, M. R. Lyu, and C. Y. Huang, "Modeling Fault Detection and Correction Processes in Software Reliability Analysis," *IEEE Trans. on Reliability*, in Revision.

[20] D. Wallace and C. Coleman, "Application and Improvement of Software Reliability Models," *Technical Report*, Software Assurance Technology Center, Oct. 2001.

[21] K. Goševa-Popstojanova and K. S. Trivedi, "Failure Correlation in Software Reliability Models," *IEEE Trans. Reliability*, Vol. 49, No. 1, pp. 37-48, March 2000.

[22] L. A. Tomek, J. K. Muppala, and K. S. Trivedi, "Modeling Correlation in Software Recovery Blocks," *IEEE Trans. Software Engineering*, Vol. 19, pp. 1071-1086, Nov. 1993.

[23] J. A. Morgan, G. J. Knafl, and W. E. Wong, "Predicting Fault Detection Effectiveness," *Proceedings of the 4th International Software Metrics Symposium*, pp. 82-89, Nov. 1997, Albuquerque, New Mexico.

[24] T. Dohi, N. Kaio, and S. Osaki, "Optimal Software Release Policies with Debugging Time Lag," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 4, No. 3, pp. 241-255, 1997.

[25] C. Y. Huang, C. T. Lin, H. K. Lo, Y. S. Su, and B. T. Lin, "Introduction to Software Reliability and Its Applications," *Technical Report*, NTHU EECS Industrial Affiliates Program (EECSIAP), Jan. 2004.

[26] M. R. Lyu and A. Nikora, "Applying Software Reliability Models More Effectively," *IEEE Software*, pp. 43-52, July 1992.

[27] M. Zhao and M. Xie, "On the Log-Power NHPP Software Reliability Model," *Proceedings of the 3rd International Symposium on Software Reliability Engineering*, pp.14-22, Oct. 1992, Research Triangle Park, North Carolina.