불완전 디버깅 환경에서의 신뢰성 보증 소프트웨어 양도 정책

박 중 양[†] · 김 영 순[††]

요 약

소프트웨어 시스템을 언제까지 테스팅해서 사용자에게 양도할 것인가를 결정하는 소프트웨어 양도 정책은 개발자가 해결해야 하는 중요한 문제의 하나이다. 이 양도 정책 문제에 대한 일반적인 접근법은 주어진 신뢰성 요구사항을 만족하기 위해 필요한 고장 발견수나 테스팅 시간을 결정하는 것이다. 소프트웨어 신뢰성은 잔존 고장수 또는 수정된 고장수에 의존하며, 불완전 디버깅 환경에서는 발견된 고장이 모두 수정되는 것이 아니므로 불완전 디버깅 환경에서의 양도 정책을 구하는 새로운 방법이 필요하다. 본 논문에서는 불완전 디버깅 환경에서 신뢰성 요구사항을 만족하였음을 원하는 신뢰수준으로 보증하는 양도 정책을 제안하고, 이를 구현하였다.

A Software Release Policy Assuring Reliability for Imperfect Debugging

Joong-Yang Park[†] · Young-Soon Kim[††]

ABSTRACT

An important issue for software developers is to determine when to stop testing the software system and release it to users. Generally the release time is specified by the number of detected faults or the testing time needed to meet the reliability requirement. Software reliability directly depends on the number of remaining or corrected faults. All the detected faults are not always corrected under imperfect debugging environment. We therefore need a new approach to software release policy for imperfect debugging. This paper suggests a software release policy, which guarantees that the reliability requirement has been achieved. The suggested policy is then implemented and illustrated for specific SRGMs.

1. Introduction

An important issue in developing a software system is to produce a high quality software system satisfying user requirements. There are many attributes of software quality. Software reliability, however, is generally accepted as the key factor in software quality since it quantifies software failures. Thus software reliability has been a primary concern for both users and software development organizations. Most software development organizations are adopting software reliability as a criterion for product release. A mathematical model called a software

† 정 회 원 : 경상대학교 자연과학대학 통계학과 교수, 경상대부설 정보통신연구센터 연구원
†† 준 회 원 : 경상대학교 대학원 통계학과 석사과정
논문접수 : 1997년 12월 22일, 심사완료 : 1998년 3월 23일

reliability growth model (SRGM) is a useful tool for grasping and assessing the degree of software reliability. A SRGM describes software fault-detection or software failure occurrence phenomena during the testing phase of software development and during the operation phase.

Another important issue for software developers is to determine when to stop testing the software system and release it to users. This problem is called an optimal software release problem. In this paper, we consider the optimal software release problem especially for imperfect debugging environment. The optimal software release policies are generally dependent on the employed SRGM and optimality criteria. Thus SRGMs for imperfect debugging and optimality criteria are briefly reviewed in Sections 2 and 3. A new approach to the optimal software release problem is suggested for the imperfect debugging environment in Section 4. The new approach allows us to declare that a software system has achieved its reliability requirement. The suggested approach is then implemented for two specific SRGMs in Sections 5.

2. SRGMs for Imperfect Dubugging

Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment. Due to design and other human errors, few software systems are completely free of faults. One way of reducing the number of faults in a software system is to perform an extensive test with the aim of detecting and removing as many faults as possible prior to its release. Such software testing also enables us to assess and improve software reliability. SRGMs are used to estimate quantitative measures such as the initial fault content, the mean time between software failures, and the software reliability and to monitor the reliability growth behavior of a software under testing or operation based on its failure history.

Most of earlier SRGMs assume that detected faults are corrected perfectly without introduction of any new faults. To make existing SRGMs more realistic, this assumption should be relaxed. It is therefore necessary to develop SRGMs which assume imperfect debugging in which the faults detected by testing are not always corrected/removed. Such imperfect debugging SRGMs are expected to estimate reliability assessment measures more accurately. SRGMs taking account of imperfect debugging were considered by Dalal and McIntosh [3], Goel and Okumoto [7,8], Kapur and Garg [11], Kapur and Younes [12], Xia, Zeephongsekul and Kumar [20], Yamada, Tokuno and Osaki [23] and Zeephongsekul, Xia and Kumar [26]. Goel and Okumoto [7,8] proposed an imperfect debugging model which is basically an extension of De-Eutrophication model advocated by Jelinski and Moranda [10]. Kapur and Garg [11] developed an imperfect debugging model based on the nonhomogeneous Poisson Process (NHPP) SRGM introduced earlier by Goel and Okumoto [9]. Dalal and McIntosh [3] considered a SRGM taking account of code changes, which implicitely include the concept of imperfect debugging. Xia, et al. [20] and Kapur and Younes [12] have further extended to the situation where there is learning in the debugging process. Zeephongsekul, et al. [26] proposed an NHPP SRGM in which primary-failures generate secondary-faults under imperfect debugging. Kapur and Younes [12] developed NHPP SRGM that describes the error removal phenomenon under imperfect debugging environment.

Yamada, Tokuno and Osaki [23] considered an imperfect debugging model based on Geometric De-Eutrophication model.

3. Optimality Criteria for Software Release Policies

Since Forman and Singpurwalla [4] addressed themselves to the optimal software release pro-blem, there have been a number of researches. The optimal software release policy is usually determined on the basis of two criteria described by Okumoto and Goel [17]. The two criteria are respectively :

1. When a measure for the quality of a software system (e.g. reliability, number of remaining faults and mean time to failure) reaches a given threshold.
2. When the cost (profit) is minimized (maximized).

Sometimes a mixture of the two criteria are employed. Specifically

3. When the cost (profit) is minimized (maximized) subject to a restriction that a measure for the quality of a software system reaches a given threshold.

Optimal software release policies for each of the three criteria are respectively referred to as reliability-optimal, cost-optimal and cost-reliability-optimal software release policies. Previ-ous studies on software release policy were classified in Table 1 with respect to types of criterion and debugging. Literatures for imperfect debugging are not many. This is because the concept of imperfect debugging was introdu-ced recently by Goel [6].

<Table 1> Classification of optimal software release policies.

	Type of debugging	
	Perfect debugging	Imperfect debugging
Reliability -optimal	[4], [5], [15], [16], [18]	[23]
Cost -optimal	[1], [2], [13], [14], [19], [21], [24], [25]	[3], [23], [26]
Cost -reliability -optimal	[19], [22]	[20], [23], [26]

4. A New Approach to Reliability- Optimal Software Release Policies

Suppose that R_0 is the software reliability objective for the operation time x_0. That is, the probability that the released software system operates without a failure for more than x_0 time should be greater than or equal to R_0. Let $R(t;T)$ and $R(t;m_d)$ respectively denote the reliability functions after T testing time and after detection of m_d faults. Previous studies on the reliability-optimal software release policy for imperfect debugging usually determine the optimal values of T and m_d subject to the given reliability objective. For example, Yamada, Tokuno and Osaki [23] obtained the minimum m_d, m_d^*, satisfying $R(x_0;m_d)\geq R_0$, whereas Xia, et al. [20] obtained the minimum T, T^*, such that $R(x_0;T)\geq R_0$. However, even though a software system is tested until m_d^* faults are detected or T^* testing time elapses, it is not assured that the software system achieves the reliability objective. Software reliability directly depends on the number of corrected faults or the number of remaining faults and we do not know how many faults are corrected until m_d^* fault detection or T^* testing time. Consequently we can not be sure that the reliability objective is accomplished. In the circumstances it is desirable to specify how confident we are that the software system attains the reliability objective. We now suggest an approach which explicitely employes the confidence level.

Consider a software release policy under which a software system is released when m_c faults are corrected. Let $R(x; m_c)$ be the reliability function when m_c faults are corrected. The minimum value of m_c satisfying $R(x_0; m_c) \geq R_0$ is the optimal value of m_c. In the imperfect debugging environment we can only recognize the detected faults and do not know whether each detected fault is corrected or not. Therefore determination of the value of m_c does not completely specify a software release policy. Thus we suggest that the software system be tested until the probability that m_c faults are corrected is greater than or equal to some acceptable value α. Then we can say $100\alpha\%$ confidently that reliability of the software system is at least R_0. The probability α is thus referred to as the confidence level. Denote by $N_c(t)$ the number of corrected faults up to t testing time. The procedure for obtaining an optimal policy is then described as :

(1) Determine the minimum integer m_c, m_c^*, satisfying the given reliability objective, i.e., $R(x_0; m_c) \geq R_0$.

(2) Determine the minimum T such that the probability that $N_c(T) \geq m_c^*$ is greater than or equal to the given confidence level α.

Section 5 applies the suggested approach to two specific imperfect debugging SRGMs recently advocated by Yamada, Tokuno and Osaki [23] and Xia, Zeephongsekul and Kumar [20]. The former belongs to the class of SRGMs describing times between failures, the latter belongs to the class of NHPP SRGMs.

5. Implementation for Two Imperfect Debu-gging SRGMs

5.1 A Reliability-Optimal Release Policy for Yamada, Tokuno and Osaki Model

Yamada, Tokuno and Osaki [23] developed a SRGM under the following assumptions.

(1) Each fault which causes a software failure is corrected perfectly with probability p.

(2) The hazard rate is constant between software failures caused by a fault in the software system and geometrically decreases whenever each detected fault is corrected.

(3) The probability that two or more software failures occur simultaneously is negligible.

(4) No new faults are introduced during the debugging. At most one fault is removed when it is corrected and the correction time is not considered.

Assumption (2) implies that when i faults have been corrected, the hazard rate for the next software failure occurrence is given by

$$z_i(t) = Dk^i, \quad i = 0, 1, 2, \cdots, \quad D > 0, \quad 0 < k < 1.$$

where D and k are the initial hazard rate and decreasing ratio, respectively. Distribution function for the next software failure occurrence time is then given by $F_i(t)=1-\exp(-Dk^i t)$. Several reliability measures were derived under these assum-ptions. Define the following random variables.

S_n : nth successful correction time of detected faults.

X_l : time interval between $(l-1)$st and l th software failures.

Let $G_n(t)$ and $\Phi_l(t)$ denote the distribution functions of S_n and X_l, respectively. It was shown that

$$G_n(t)=\sum_{i=0}^{n-1}A_{k,i,n}\{1-\exp(-pDk^i t)\}$$

and

$$\Phi_l(t)=\sum_{i=0}^{l-1}\binom{l-1}{i}p^i(1-p)^{l-1-i}\{1-\exp(-Dk^i t)\}$$

where $A_{k,0,1}=1$ and $A_{k,i,n}=k^{n(n-1)/2-i}\Big/\prod_{j=0,j\neq i}^{n-1}(k^j-k^i)$. Denoting by $P_n(t)$ the probability that $N_c(t)=n$, it is easily verified that $P_n(t)=G_n(t)-G_{n+1}(t)$.

Yamada, Tokuno and Osaki [23] then suggested a software release policy which releases a software system when m_d faults have been detected. The optimum value of m_d, m_d^* is the minimum integer m_d satisfying $R(x_0;m_d)\geq R_0$. Here $R(x_0;m_d)=1-\Phi_{m_d+1}(x_0)$. Since this release policy produces a random, not deterministic, release time, it was suggested that a software system be released at $T_d^*=\sum_{l=1}^{m_d^*}E(X_l)$ where $E(X_l)=(p/k+(1-p))^{l-1}/D$. Table 2 of Yamada, Tokuno and Osaki [23] shows the optimal values of m_d and corresponding T_d^* for $D=0.2$, $k=0.9$, $x_0=2.5$ and $R_0=0.95$. For the sake of comparison the optimal values are reproduced in Table 2. Note that $m_d^*=22$ for $p=1.0$ and $m_c^*=m_d^*$ when $p=1.0$. This implies that at least 22 faults should be corrected in order to attain the given reliability objective. We computed $\Pr(N_c(T_d^*)\geq m_c^*)$, the probability that at least $m_c^*=22$ faults are corrected during T_d^* testing time. The probabilities does not seem to be large enough, so the release policy of Yamada, Tokuno and Osaki does not assure the software reliability sufficiently.

〈Table 2〉 Values of m_d^* and T_d^* for $D=0.2$, $k=0.9$, $x_0=2.5$ and $R_0=0.95$.

p	m_d^*	T_d^*	$\Pr(N_c(T_d^*)\geq m_c^*)$
1.0	22	411.96	0.64
0.9	25	491.74	0.65
0.8	28	554.22	0.65
0.7	32	642.11	0.67
0.6	37	741.82	0.67
0.5	45	935.40	0.72
0.4	56	1171.99	0.73
0.3	75	1604.38	0.77
0.2	113	2471.[illegible]	0.80
0.1	226	5017.14	0.82

Let us now derive the optimal software release policy suggested in Section 4. First we obtain the minimum value of m_c satisfying given reliability objective. If m_c faults are corrected, the hazard rate for the next software failure is given by $z_{m_c}(t)$. Thus the corresponding reliability function is $R(t;m_c)=\exp(-Dk^{m_c}t)$. The minimum inte-ger satisfying $R(x_0;m_c)\geq R_0$ is then the optimum number of corrected faults. If $R(x_0;0)<R_0$, monotonicity of $R(x_0;m_c)$ ensures that there exists a unique m_c^* such that $R(x_0;m_c^*)\geq R_0$ and $R(x_0;m_c^*-1)<R_0$. Otherwise, $m_c^*=0$. That is,

$$m_c^*=\begin{cases}0, & \text{if } R(x_0;0)\geq R_0\\ \left[\dfrac{\ln(-\ln R_0)-\ln D-\ln x_0}{\ln k}\right]+1, & \text{otherwise}\end{cases}$$

where $[x]$ denote the largest integer less than or equal to x. The optimum software release time T_c^* is then the minimum value of T satisfying $\Pr(N_c(T)\geq m_c^*)\geq a$. Since $\Pr(N_c(T)\geq m_c^*)=G_{m_c^*}(T)$ is a monotone increasing function of T, T_c^* is the root of $G_{m_c}(T)=a$. For example, assume that $x_0=2.5$, $R_0=0.95$, $D=0.2$ and $k=0.9$ as in Table 2. The optimal value of m_c is obtained as 22 from equation (5.1). The corresponding values of T_c^* are numerically computed and presented in Table 3 for $a=0.90$ and 0.95. As expected, more testing time is required to assure the software reliability.

<Table 3> Values of T_c^* for $D=0.2$, $k=0.9$, $x_0=2.5$ and $R_0=0.95$.

	T_c^*	
p	$a=0.90$	$a=0.95$
1.0	550.05	599.41
0.9	611.17	666.01
0.8	687.57	749.26
0.7	785.79	856.30
0.6	916.76	999.02
0.5	1100.11	1198.82
0.4	1375.14	1498.82
0.3	1833.51	1998.04
0.2	2750.27	2997.06
0.1	5500.54	5994.12

5.2 A Reliability-Optimal Policy for Xia, Zeephongsekul and Kumar Model

Xia, Zeephongsekul and Kumar [20] developed an NHPP SRGM for imperfect debugging, in which the debugging process improves with experiences, that is, there is a learning factor involved. A cost-reliability-optimal release policy was then obtained. The model is based on the following assumptions.

(1) Software system is subjected to failures caused by faults remaining in the software system.
(2) Software failure rate is equally affected by faults remaining in the software system.
(3) The occurrences of software failure follow an NHPP.
(4) The software failure rate at any time is proportional to the number of faults remaining in the software system at that time.
(5) Failures are independent and each failure is caused by one fault.
(6) On the occurrence of $(i+1)$st failure, the following may occur:
  - fault content is reduced by one with probability p_i;
  - fault content is unchanged with probability $(1-p_i)$.
(7) The probability of fixing a failure is a linearly increasing function of the number of repaires carried out in the past, that is, $p_i=p_0[1+(i-1)l]$,

where p_0 is the probability of correcting the 1st fault and l is the learning factor.

Let $m_c(t)$ and $m_d(t)$ denote the expected values of $N_c(t)$ and $N_d(t)$, where $N_c(t)$ and $N_d(t)$ are the numbers of corrected faults and detected faults up to time t. It was also shown that

$$m_c(t) = \frac{1}{2} m_d^2(t) p_0 l + p_0 m_d(t)$$

and

$$m_d(t) = \frac{2a[1-\exp(-bp_0 rt)]}{p_0[(1+r)-(1-r)\exp(-bp_0 rt)]}$$

where a is the total number of faults in the software system, b is the proportionality constant (failure rate per fault) and $r = \sqrt{1+2al/p_0}$.

We now obtain the optimal release time for this SRGM. Assumptions (2) and (4) implies that the reliability function $R(t; m_c)$ is given by $\exp[-b(a-m_c)t]$. If $R(x_0; 0) \geq R_0$, then $m_c^* = 0$. Otherwise, m_c^* is the minimum integer such that $R(x_0; m_c) \geq R_0$. Therefore the optimal value of m_c is obtained as

$$m_c^* = \begin{cases} 0, & \text{if } R(x_0; 0) \geq R_0 \\ \left[a + \frac{\ln R_0}{bx_0}\right] + 1, & \text{otherwise.} \end{cases} \quad (6.1)$$

The corresponding optimal release time T^* is thus the minimum T satisfying $\Pr(N_c(T) \geq m_c^*) \geq \alpha$ for some acceptable α. Since $N_c(T)$ is a Poisson distributed random variable with mean $m_c(T)$, we can obtain T^* numerically. We illustrate by an example, which was considered in Example 1 of Xia, et al. [20].

Example Supposes that $a = 26.63$, $b = 0.0072$, $p_o = 0.6$, $l = 0.026$, $x_0 = 2.0$ and $R_0 = 0.8$. Suppose further that $\alpha = 0.9$. Using these parameters, we obtain $m_c^* = 12$ from equation (6.1) and $T^* = 170.498$ by a numerical method. The minimum value of T such that $R(x_0; T) \geq R_0$ was however obtained as 101.83 in Xia, et al. [20].

6. Discussion

It is important to determine an appropriate release time for a software system under development. In this paper we suggested a new approach to reliability-optimal software release policies for the imperfect debugging environment. The approach first determines the number of faults to be corrected and then the testing time required to correct the determined number of faults with some desirable probability. So we are provided with software release policies assuring that a software system attains the given reliability objective. The suggested approach is so general that it can be also applied to the cost-reliability-optimal software release policies.

References

[1] D. S. Bai and W. Y. Yun, "Optimum Number of Errors Corrected before Releasing A Software System," IEEE Trans. Rel., Vol. 1, pp. 41-44, 1988.
[2] S. R. Dalal and C. L. Mallows, "When

Should One Stop Testing Software," Journal of the American Statistical Association, Vol. 83, No. 403, pp. 872-879, 1988.

[3] S. R. Dalal and A. A. McIntosh, "When to Stop Testing for Large Software Systems with Changing Code," IEEE Trans. Software Eng., Vol. 20, No. 4, pp. 318-323, 1994.

[4] E. H. Forman and N. D. Singpurwalla, "An Empirical Stopping Rule for Debugging and Testing Computer Software," Journal of the American Statistical Association, Vol. 72, pp. 750-757, 1977.

[5] E. H. Forman and N. D. Singpurwalla, "Optimal Time Intervals for Testing Hypothesis on Computer Software Errors," IEEE Trans. Rel., Vol. R-28, pp. 250-253, 1979.

[6] A. L. Goel, "Software Reliability Models : Assumptions, Limitations and Applicability," IEEE Trans. Software Eng., Vol.Se-11, No. 12, pp. 1411-1423, 1985.

[7] A. L. Goel and K. Okumoto, "An Analysis of Recurrent Software Failures in A Real-time Control System," in Proc. ACM Annu. Tech. Conf., ACM. Washington. DC. pp.496-500, 1978.

[8] A. L. Goel and K. Okumoto, "A Markovian Model for Reliability and Other Performance Measures of Software Systems," in Proc. Nat. Comput. Conf., New York, Vol. 48, p p. 769-774, 1979.

[9] A. L. Goel and K. Okumoto, "Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," IEEE Trans. Rel., Vol.R-28, pp.206-211, 1979.

[10] Z. Jelinski and P. Moranda, "Software Reliability Research," in Statistical Computer Performance Evaluation, W. Freiberger, Ed. New York : Academic, pp. 465-484, 1972.

[11] P. K. Kapur and R. B. Garg, "Optimal Software Release Policies for Software Reliability Growth Models under Imperfect Debugging," Operations Research, Vol.24, pp.295-305, 1990.

[12] P. K. Kapur and S. Younes, "Modelling An Imperfect Debugging Phenomenon in Software Reliability," Microelectron. Reliab., Vol. 36, No. 5, pp. 645-650, 1996.

[13] H. S. Koch and P. Kubat, "Optimal Release Time of Computer Software," IEEE Trans. Software Eng., Vol. SE-9, No. 3, pp. 323-327, 1983.

[14] Y. Masuda, N. Miyawaki, U. Sumita and S. Yokoyama, "A Statistical Approcah for Determining Release Time of Software System with Modular Structure," IEEE Trans. Rel., Vol. 38, pp. 365-372, 1989.

[15] J. D. Musa and A. F. Ackerman, "Quantifying Software Validation : When to Stop Testing?," IEEE Software, May, pp. 19-27, 1989.

[16] H. Ohtera and S. Yamada, "Optimum Software-Release Time Considering an Error Detection Phenomenon During Operation," IEEE Trans. Rel., Vol. R-39, pp. 596-599, 1990.

[17] K. Okumoto and A. Goel, "Optimum Release Time for Software Systems Based on Reliability and Cost Criteria," The Journal of Systems and Software, Vol. 1, No. 4, pp. 315-318, 1980.

[18] S. M. Ross, "Software Reliability : The Stopping Rule Problem," IEEE Trans. Software Eng., Vol. SE-11, pp. 1472-1476, 1985.

[19] N. D. Singpurwalla, "Determining An Optimal Time Interval for Testing and Debugging Software," IEEE Trans. Software Eng., Vol. 17, No. 4, pp. 313-319, 1991.

[20] G. Xia, P. Zeephongsekul and S. Kumar, "Optimal Software Release Policies with Learning Factor for Imperfect Debugging," Microelectronics & Reliability, Vol. 33, pp.81-86, 1993.

[21] S. Yamada, J. Hishitani and S. Osaki, "Software Reliability Growth with A Weibull Test-effort : A Model & Application," IEEE Trans. Rel., Vol. 42, pp. 100-105, 1993.

[22] S. Yamada and S. Osaki, "Software Reliability Growth Modeling : Models and Assumptions," IEEE Trans. Software Eng., Vol. SE-11, No. 12, pp. 1431-1437, 1985.

[23] S. Yamada, K. Tokuno and S. Osaki, "Software Reliability Measurement in Imperfect Debugging Environment and Its Application," Reliability Eng. and System Safety, Vol.40, pp.139-147, 1993.

[24] M. C. K. Yang and A. Chao, "Reliability-Estimation & Stopping-Rules for Software Testing, Based on Repeated Appearances of Bugs," IEEE Trans. Rel., Vol. 44, No. 2, pp. 315-321, 1995.

[25] W. Y. Yun and D. S. Bai, "Optimum Software Release Policy with Random Life Cycle," IEEE Trans. Rel., Vol. 39, pp.167-170, 1990.

[26] P. Zeephogsekul, G. Xia and S. Kumar, "Software-Reliability Growth Model : Primary-Failures Generate Secondary-Faults under Imperfect Debugging," IEEE Trans. Rel., Vol.43, No. 3, pp.408-413, 1994.

[redacted]

박 중 양

1982년 연세대학교 응용 통계학과 졸업(학사)
1984년 한국과학기술원 산업공학과 응용통계전공(석사)
1994년 한국과학기술원 산업공학과 응용통계전공(박사)
1984년~1989년 경상대학교 전산통계학과 교수
1989년~현재 경상대학교 통계학과 교수
관심분야 : 소프트웨어 신뢰성, 신경망, 선형 통계 모형, 실험계획법

[redacted]

김 영 순

1994년 경상대학교 통계학과 졸업(이학사)
1996년~현재 경상대학교 통계학과(석사과정)
관심분야 : 소프트웨어 신뢰성, 신경망, 선형 통계 모형, 실험계획법