입력 영역에 기초한 소프트웨어 신뢰성 성장 모델

박 중 양[†] · 서 동 우[††] · 김 영 순[†††]

요 약

소프트웨어를 테스팅하는 동안 얻어지는 고장 데이터를 분석하여 소프트웨어의 신뢰성이 성장하는 과정을 평가하기 위해 여러 가지 소프트웨어 신뢰성 성장 모델들이 개발되었다 그러나 이들 신뢰성 성장 모델들은 소프트웨어 개발과 사용 환경에 관한 여러 가정에 기반하고 있기 때문에, 이 가정이 적합하지 않은 상황이나 결함이 드물게 발생되는 소프트웨어에 대해서는 적절하지 않다. 입력 영역에 기초한 소프트웨어 신뢰성 모델은 일반적으로 이러한 가정을 요구하지 않는데 디버깅 전의 소프트웨어와 디버깅 후의 소프트웨어를 별개의 것으로 다루어 많은 테스트 입력을 요하는 단점이 있다. 본 논문에서는 이러한 가정이 요구되지 않고 디버깅 전과 후의 소프트웨어를 동시에 테스트하는 방법에 기반을 둔 입력 영역 기반 소프트웨어 신뢰성 성장모델을 제안하고 그 통계적 특성을 조사한다. 이 모델은 모든 데이터를 다 활용하기 때문에 기존 입력영역 소프트웨어 신뢰성 모델에 비해 적은 테스트 입력을 필요로 할 것으로 기대된다 그리고 소프트웨어의 유지보수 단계에 적용하기 위해 개발된 유사한 방법들과 비교한다

An Input Domain-Based Software Reliability Growth Model

Joong-Yang Park[†] · Dong-Woo Seo[††] · Young Soon Kim[†††]

ABSTRACT

A number of software reliability growth models (SRGMs) have been developed for evaluating software reliability growth behavior by analyzing the failure data obtained during testing software systems However, since these SRGMs are based on several assumptions about software development and usage environments, SRGMs are inadequate for circumstances in which such assumptions do not hold or software systems rarely fail. The existing input domain-based reliability models, which do not require assumptions on software development and usage environment, deal the software system before debugging and the software system after debugging independently. Therefore many test inputs are usually demanded This paper thus suggests an input domain-based SRGM, which does not require such assumptions and is based on the testing procedure that tests concurrently both the software system before debugging and the software system after debugging The suggested model uses all the available data, the required number of test inputs can be possibly reduced This reduction may compensate for the excessive testing time caused by executing the software systems before and after debugging. Its statistical *characteristics are investigated and it is compared with similar approaches developed for the software maintenance phase*

1. Introduction

In recent years there is a growing use of computer systems and software systems have been most important parts of many complex and critical computer systems Since failures of a software system could produce severe consequences in terms of human life, environment impact or economical losses, software systems are required to be sufficiently reliable for their intended purposes Software reliability has become one of major issues in the software system development.

† 정 회 원 : 경상대학교 통계학과 교수
†† 준 회 원 : 경상대학교 대학원 통계학과(석사 수료)
††† 준 회 원 : 경상대학교 대학원 통계학과
논문접수 : 2000년 7월 18일, 심사완료 : 2000년 10월 27일

So far the only way to improve and evaluate software reliability is software testing. That is, software testing is a means of improving the software reliability by probing the software system for faults and removing them. A number of software reliability models have been developed for analyzing the data obtained during testing the software system. Ramamoorthy and Bastani [13] classify software reliability models according to the development phases of software life-cycle, while Goel [7] divides them according to the nature of failure process into time-between-failures models, failure-count models, fault seeding models and input domain-based models. Recently Park et al. [15, 16] suggest several neural network models for predicting software reliability. Henceforth we only consider reliability models which are applicable in the testing and debugging phase. Since fault corrections are necessary in the testing and debugging phase, software reliability growth models (SRGMs) taking fault corrections into account are mainly used in this phase. Most of time-between-failures models and failure-count models belong to the class of SRGMs. Several researches [2, 4, 5, 8] indicate drawbacks of SRGMs : The usual assumptions made for SRGMs are still questionable ; SRGMs do not sufficiently account for the characteristics of the software systems under testing ; SRGMs do not work well for the software systems which rarely fail during testing. We will thus consider the input domain-based models which do not require assumptions on the software failure and correction processes. Ramamoorthy and Bastani [13] classify the input domain-based models into the class of reliability models for the validation phase. However, the input domain-based models can also be applied in the testing and debugging phase by treating the software system after each fault correction as a new software system. Nelson [11] estimates the reliability change as more information becomes available under the assumption that the reliability function remains constant. Weiss and Weyuker [14] extended this approach by treating reliability as a generalization of the probability of correctness of the software system. Ramamoorthy and Bastani [13] proposed an input domain-based model that estimates correctness instead of reliability. They assume that faith in software correctness can grow with the number of tests correctly executed. However, the attribute correctness is less useful than reliability since it does not give information about the influence of residual faults. We may further refer to Brown and Lipow [1], MacWilliams [9], and Tsoukalas, Duran and Ntafos [6].

Software reliability has been defined as the probability that no failure occurs in a specified environment during a specified exposure period. The time unit of exposure period depends on the type of reliability model used. Specifically SRGMs define the software reliability as

$$R_T = \Pr(\text{ no failure occurs within time period } [0, T]), \qquad (1)$$

where T is the exposure period whose time unit is the calendar or execution time. It is usually assumed that R_T follows a certain probability distribution, for example, $R_T = \exp\left(-\int_0^T z(t)dt\right)$ where $z(t)$ is the failure rate function. The main concern of these SRGMs is to estimate the failure rate function. On the other hand, input domain-based reliability models define the software reliability as

$$R_N = \Pr(\text{no failure occurs over } N \text{ application runs}), \quad (2)$$

where N is the exposure period whose time unit is the number of application runs. Assuming that inputs are selected independently according to the operational profile, $R_N = R^N$ where $R = R_1$. Since R_N can be simply calculated from R, reliability R becomes the main objective to be estimated by input domain-based reliability models. Obviously R is closely related to the operational profile of the software system. The operational profile is the nature of the software usage, which is in general determined by the systems that interact with the software system. The operational profile can be expressed by a probability distribution over the input domain, which represents the frequency of occurrence of each input in the input domain. Musa

[10] described a step by step procedure to develop a practical operational profile. In many applications, the operational profile is very difficult to obtain and even a crude approximation requires considerable efforts. Operative usage of the software system can be unpredictable, unknown or different for different users. It is not easy or even impossible in some cases for testing to follow the actual operational profile. Therefore, the operational profile is usually unknown in the testing and debugging phase. In this case, the uniform operational profile is assumed.

This paper proposes an input domain-based SRGM to describe the reliability growth behavior of a software system. The proposed SRGM is based on a multi-stage testing procedure, in which fault corrections occur after each stage and two software systems, the software system before fault corrections and the software system after fault corrections, are tested at each stage. Random testing is used to generate inputs from the input domain. Section 2 describes the testing profile and the multi-stage testing procedure. The input domain-based SRGM is presented and its statistical characteristics are investigated in Section 3. Then similar approaches developed for the maintenance phase will be compared with the suggested SRGM in Section 4. Conclusions and remarks on future research are made in Section 5.

2. Testing Profile and Testing Procedure

Each software system has a specification which is an input-output relation. That is, the specification S is a set of ordered input output pairs describing functions of the software system. The collection of inputs in S defines the input domain ID of the software system. If the software system produces output o on input i such that $(i, o) \in S$ for any $i \in ID$, the software system is said to meet its specification. A software system with specification S fails on an input if the software system does not meet S at the input. When a software system fails, the event is called a failure. One or more defects in the software system related to the failure are called faults. The input responsible for the failure is referred to as a failure-causing input. Thus the software reliability is characterized by the failure probability when inputs are selected according to the operational profile.

A systematic testing method should include a criterion for selecting test inputs, a procedure for conducting the testing and a criterion for deciding when to stop testing. During testing we select inputs from the input domain according to an input selection strategy and execute the software system by applying the selected inputs. If failures occur, we detect and remove faults which caused the failures. Such an action is called a fault correction. That is, a fault correction is an action aimed at localizing and removing the faults discovered by testing. An input selection strategy is called a testing profile. Therefore, a testing profile describes how to select inputs from the input domain. In general two types of input selection strategy, random testing and partition testing, are commonly used. The random testing selects inputs from the input domain by sampling randomly according to the testing profile, while the partition testing partitions the input domain into classes and forces at least one input to come from each class. Adopting the random testing, we will develop an input domain-based SRGM in this paper. However, design of a testing profile should be guided by the operational profile of the software system under testing to ensure that most frequently used operations receive most frequent tests. In the rest of this paper we assume that the operational profile is given and the testing profile is the same as the operational profile.

Next we consider the testing procedure which will be used in this paper. A test run is an execution of the software system by applying an input selected according to the given testing profile. The term "test stage" refers to some predetermined number of consecutive test runs. The testing is performed stage by stage. If failures occur during a testing stage, testing will not be interrupted for fault corrections. Fault corrections occur at the end of each testing stage. We denote by P_i the software system after fault corrections of $(i-1)$st testing stage (equivalently at the beginning of ith testing stage). The failure probability of P_i is denoted by θ_i. Moreover we denote the number of test inputs for ith testing stage by n_i. The existing input domain-based reliability models treat the software system after each test stage as a new software system. Thus they estimate θ_i from the failure data obtained during ith testing stage. Two software systems P_{i-1} and P_i are identical except for the part debugged after $(i-1)$st testing stage. The debugged part is likely to be a small portion of the software system. This implies that the failure data obtained before ith testing stage contains information on the undebugged part of P_i. It is thus desirable to develop a testing procedure through which relationship among θ_i's can be derived. We thus propose the testing procedure in which n_i test inputs are applied to both P_{i-1} and P_i for $i \geq 2$. One exception is that only P_1 is executed at 1st testing stage. This procedure will provide us with a relationship between θ_{i-1} and θ_i. Consequently it enables us to derive an algorithm for updating the previous estimate of failure probability, which may be referred to as an input domain-based SRGM. Input domain-based reliability models usually require large test inputs for obtaining accurate reliability estimates. Taking advantage of information obtained previously, this procedure can reduce the required number of test inputs. But, since we must exercise both the previous and current software systems, it costs more testing time. Thus this procedure is more economical when the cost of test input generation is higher than the cost of test input execution. The availability of the specification S will make it easy to evaluate and classify the outcome of each test input execution.

3. An Input Domain-Based SRGM

We suppose that the input domain and the random testing profile are given and remain unchanged during the testing phase. Further suppose that the testing procedure suggested in Section 2 is used. Only P_1 is exercised in 1st testing stage. Execution of each input in 1st testing stage results in one of the two outcomes, success (S) and failure (F). The number of failures occurred in 1st testing stage will be denoted by x_{1F}. However, both P_{i-1} and P_i are exercised in ith testing stage for $i \geq 2$. Each test run in the ith testing stage results in one of the five outcomes, SS, SF, FS, FFD and FFS of which brief descriptions are presented in Table 1.

〈Table 1〉 Brief descriptions of outcomes of each test run

outcome	description
SS	both P_{i-1} and P_i succeed
SF	P_{i-1} succeeds and P_i fails
FS	P_{i-1} fails and P_i succeeds
FFD	both P_{i-1} and P_i fail and outputs of P_{i-1} and P_i are different
FFS	both P_{i-1} and P_i fail and outputs of P_{i-1} and P_i are same

We denote the number of test runs and the occurrence probability corresponding to each outcome by $x_{i\,\mathrm{outcome}}$ and $p_{i\,\mathrm{outcome}}$. Then x_{1F} follows a binomial distribution with parameters n_1 and θ_1 and the joint distribution of x_{iSS}, x_{iSF}, x_{iFS}, x_{iFFD} and x_{iFFS} for $i \geq 2$ is given by the following multinomial distribution :

$$f(x_{iSS}, x_{iSF}, x_{iFS}, x_{iFFD}, x_{iFFS}) = \binom{n_i}{x_{iSS}, x_{iSF}, x_{iFS}, x_{iFFD}, x_{iFFS}} p_{iSS}^{x_{iSS}} p_{iSF}^{x_{iSF}} p_{iFS}^{x_{iFS}} p_{iFFD}^{x_{iFFD}} p_{iFFS}^{x_{iFFS}}, \tag{3}$$

where

$$\binom{n_i}{x_{iSS}, x_{iSF}, x_{iFS}, x_{iFFD}, x_{iFFS}} = \frac{n_i!}{x_{iSS}!x_{iSF}!x_{iFS}!x_{iFFD}!x_{iFFS}!},$$

$x_{iSS}+x_{iSF}+x_{iFS}+x_{iFFD}+x_{iFFS}=n_i$ and $p_{iSS}+p_{iSF}+p_{iFS}+p_{iFFD}+p_{iFFS}=1$ The outcomes SF and FFD are in general due to the imperfect debugging, i.e., imperfect fault correction. Once a failure occurs, the corresponding fault corrections may introduce new faults. However, tracking the fault correction activities, faults responsible for the failure can be eventually removed. We henceforth assume the perfect debugging. Then only three outcomes, *SS*, *FS* and *FFS*, can occur and the joint distribution of x_{iSS}, x_{iFS} and x_{iFFS} are given by

$$f(x_{iSS}, x_{iFS}, x_{iFFS}) = \binom{n_i}{x_{iSS},\ x_{iFS},\ x_{iFFS}} p_{iSS}^{x_{iSS}} p_{iFS}^{x_{iFS}} p_{iFFS}^{x_{iFFS}}, \quad (4)$$

where $x_{iSS} + x_{iFS} + x_{iFFS} = n_i$ and $p_{iSS} + p_{iFS} + p_{iFFS} = 1$. In order to estimate reliability, we need to represent Expression (4) in terms of θ_i's. First note that the following relationships hold.

$$\begin{aligned}\theta_{i-1} &= \Pr(P_{i-1}\text{ fails}) \\ &= \Pr(P_{i-1}\text{ fails and } P_i\text{ succeeds}) \\ &\quad + \Pr(P_{i-1}\text{ and } P_i\text{ both fail}) \\ &= \Pr(P_{i-1}\text{ fails and } P_i\text{ succeeds}) \\ &\quad + \Pr(P_{i-1}\text{ and } P_i\text{ both fail and produce same outputs}) \\ &\quad + \Pr(P_{i-1}\text{ and } P_i\text{ both fail and produce different outputs}) \\ &= p_{iFS} + p_{iFFS} + p_{iFFD} \end{aligned} \quad (5)$$

and

$$\begin{aligned}\theta_i &= \Pr(P_i\text{ fails}) \\ &= \Pr(P_{i-1}\text{ succeeds and } P_i\text{ fails}) \\ &\quad + \Pr(P_{i-1}\text{ and } P_i\text{ both fail}) \\ &= \Pr(P_{i-1}\text{ succeeds and } P_i\text{ fails}) \\ &\quad + \Pr(P_{i-1}\text{ and } P_i\text{ both fail and produce same outputs}) \\ &\quad + \Pr(P_{i-1}\text{ and } P_i\text{ both fail and produce different outputs}) \\ &= p_{iSF} + p_{iFFS} + p_{iFFD}. \end{aligned} \quad (6)$$

Since $p_{iSF} = p_{iFFD} = 0$ under the perfect debugging assumption, Expressions (5) and (6) are simplified to $\theta_{i-1} = p_{iFS} + p_{iFFS}$ and $\theta_i = p_{iFFS}$. The probabilities p_{iFS} and p_{iFFS} are then expressed as $p_{iFS} = \theta_{i-1} - \theta_i$ and $p_{iFFS} = 1 - p_{iSS} - \theta_{i-1} + \theta_i$. Substituting these into Expression (4), the joint distribution of x_{iSS}, x_{iFS} and x_{iFFS} can be rewritten as

$$f(x_{iSS}, x_{iFS}, x_{iFFS}) = \binom{n_i}{x_{iSS},\ x_{iFS},\ x_{iFFS}} p_{iSS}^{x_{iSS}}(\theta_{i-1} - \theta_i)^{x_{iFS}}(1 - p_{iSS} - \theta_{i-1} + \theta_i)^{x_{iFFS}}. \quad (7)$$

We now consider the problem of estimating the failure probability of the software system. After completing i testing stage, we have to estimate parameters θ_j and p_{jSS} for $j \le i$, among which the most interesting parameter is θ_i. The maximum likelihood estimates (MLEs) of the parameters will be derived in this paper by maximizing the likelihood function L given as

$$L = \begin{cases} \binom{n_1}{x_1}\theta_1^{x_{1F}}(1-\theta_1)^{n_1 - x_{1F}}, & \text{for } i = 1 \\ \binom{n_1}{x_1}\theta_1^{x_{1F}}(1-\theta_1)^{n_1 - x_{1F}} \prod_{j=2}^{i} \binom{n_j}{x_{jSS},\ x_{jFS},\ x_{jFFS}} \\ \quad p_{jSS}^{x_{jSS}}(\theta_{j-1} - \theta_j)^{x_{jFS}}(1 - p_{jSS} - \theta_{j-1} + \theta_j)^{x_{jFFS}}, \\ \quad \text{for } i \ge 2. \end{cases}$$

MLEs are usually obtained by maximizing the log likelihood function $\ln L$ with respect to θ_j's and p_{jSS} for $j \le i$. For the case where $i = 1$, it can be easily shown that $\hat{\theta}_1 = x_{1F}/n_1$. If $i \ge 2$, we have to estimate θ_j and p_{jSS} for $j \le i$. Let us denote MLEs of θ_j and p_{jSS} obtained after ith stage by $\hat{\theta}_{j,i}$ and $\hat{p}_{jSS,i}$. Accordingly $\hat{\theta}_1 = x_{1F}/n_1$ obtained after 1st stage is denoted by $\hat{\theta}_{1,1}$. We will show by mathematical induction that

$$\hat{\theta}_{j,i} = \hat{\theta}_{j,(i-1)} \text{ and } \hat{p}_{jSS,i} = \hat{p}_{jSS,(i-1)}$$
$$\text{for } j \le (i-1),$$
$$\hat{\theta}_{i,i} = \frac{x_{1F}}{n_1} - \sum_{j=2}^{i}\frac{x_{jFS}}{n_j}, \quad (8)$$

and

$$\hat{p}_{iSS,i} = \frac{x_{iSS}}{n_i}.$$

When $i = 2$, the likelihood equations are obtained as

$$\frac{\partial \ln L}{\partial \theta_1} = \frac{x_{1F}}{\theta_1} - \frac{n_1 - x_{1F}}{1-\theta_1} + \frac{x_{2FS}}{\theta_1 - \theta_2} - \frac{x_{2FFS}}{1 - p_{2SS} - \theta_1 + \theta_2} = 0 \quad (9)$$

$$\frac{\partial \ln L}{\partial p_{2SS}} = \frac{x_{2SS}}{p_{2SS}} - \frac{x_{2FFS}}{1-p_{2SS}-\theta_1+\theta_2} = 0$$

$$\frac{\partial \ln L}{\partial \theta_2} = -\frac{x_{2FS}}{\theta_1-\theta_2} + \frac{x_{2FFS}}{1-p_{2SS}-\theta_1+\theta_2} = 0 .$$

Substituting $\partial \ln L/\partial\theta_2 = 0$ into $\partial \ln L/\partial\theta_1 = 0$, then $\partial \ln L/\partial\theta_1 = 0$ becomes the same with $\partial \ln L/\partial\theta_1 = 0$ for 1st stage. Solving $\partial \ln L/\partial p_{2SS} = 0$ and $\partial \ln L/\partial\theta_2 = 0$ simultaneously with θ_1 replaced by $\hat{\theta}_{1,1}$, we can verify that $\partial \ln L/\partial p_{2SS} = 0$ and $\partial \ln L/\partial\theta_2 = 0$ hold only when $\theta_2 = x_{1F}/n_1 - x_{2FS}/n_2$ and $p_{2SS} = x_{2SS}/n_2$ Therefore

$$\hat{\theta}_{1,2} = \hat{\theta}_{1,1}, \quad \hat{\theta}_{2,2} = \frac{x_{1F}}{n_1} - \frac{x_{2FS}}{n_2} \text{ and}$$
$$\hat{p}_{2SS,2} = \frac{x_{2SS}}{n_2}$$

are the unique solution of the likelihood equations (9), i.e., MLEs of θ_1, θ_2 and p_{2SS}. This is the desired result. The likelihood equations for $(i-1)$st stage are given by

$$\frac{\partial \ln L}{\partial \theta_1} = \frac{x_{1F}}{\theta_1} - \frac{n_1-x_{1F}}{1-\theta_1} + \frac{x_{2FS}}{\theta_1-\theta_2} - \frac{x_{2FFS}}{1-p_{2SS}-\theta_1+\theta_2} = 0$$

$$\frac{\partial \ln L}{\partial p_{2SS}} = \frac{x_{2SS}}{p_{2SS}} - \frac{x_{2FFS}}{1-p_{2SS}-\theta_1+\theta_2} = 0$$

$$\frac{\partial \ln L}{\partial \theta_2} = -\frac{x_{2FS}}{\theta_1-\theta_2} + \frac{x_{2FFS}}{1-p_{2SS}-\theta_1+\theta_2} = 0 \quad (10)$$

$$\vdots$$

$$\frac{\partial \ln L}{\partial \theta_{i-2}} = -\frac{x_{(i-2)FS}}{\theta_{i-3}-\theta_{i-2}} + \frac{x_{(i-2)FFS}}{1-p_{(i-2)SS}-\theta_{i-3}+\theta_{i-2}} + \frac{x_{(i-1)FS}}{\theta_{i-2}-\theta_{i-1}} - \frac{x_{(i-1)FFS}}{1-p_{(i-1)SS}-\theta_{i-2}+\theta_{i-1}} = 0$$

$$\frac{\partial \ln L}{\partial p_{(i-1)SS}} = \frac{x_{(i-1)SS}}{p_{(i-1)SS}} - \frac{x_{(i-1)FFS}}{1-p_{(i-1)SS}-\theta_{i-2}+\theta_{i-1}} = 0$$

$$\frac{\partial \ln L}{\partial \theta_{i-1}} = -\frac{x_{(i-1)FS}}{\theta_{i-2}-\theta_{i-1}} + \frac{x_{(i-1)FFS}}{1-p_{(i-1)SS}-\theta_{i-2}+\theta_{i-1}} = 0.$$

Suppose that the unique solution for the above likelihood equations is given by

$$\hat{\theta}_{j,(i-1)} = \hat{\theta}_{j,(i-2)} \text{ and } \hat{p}_{jSS,(i-1)} = \hat{p}_{jSS,(i-2)} \quad \text{for } j \le (i-2).$$

$$\hat{\theta}_{(i-1),(i-1)} = \frac{x_{1F}}{n_1} - \sum_{j=2}^{i-1} \frac{x_{jFS}}{n_j}$$

and

$$\hat{p}_{(i-1)SS,(i-1)} = \frac{x_{(i-1)SS}}{n_{i-1}} ,$$

that is, Expression (8) holds. Next consider the likelihood equations for ith stage, which consist of the likelihood equations for $(i-1)$st stage with one modification and the following two additional equations.

$$\frac{\partial \ln L}{\partial p_{iSS}} = \frac{x_{iSS}}{p_{iSS}} - \frac{x_{iFFS}}{1-p_{iSS}-\theta_{i-1}+\theta_i} = 0$$

$$\frac{\partial \ln L}{\partial \theta_i} = -\frac{x_{iFS}}{\theta_{i-1}-\theta_i} + \frac{x_{iFFS}}{1-p_{iSS}-\theta_{i-1}+\theta_i} = 0. \quad (11)$$

One modification is that $\partial \ln L/\partial\theta_{i-1} = 0$ for $(i-1)$st stage is changed to

$$\frac{\partial \ln L}{\partial \theta_{i-1}} = -\frac{x_{(i-1)FS}}{\theta_{i-2}-\theta_{i-1}} + \frac{x_{(i-1)FFS}}{1-p_{(i-1)SS}-\theta_{i-2}+\theta_{i-1}} + \frac{x_{iFS}}{\theta_{i-1}-\theta_i} - \frac{x_{iFFS}}{1-p_{iSS}-\theta_{i-1}+\theta_i} = 0 .$$

However, if we substitute $\partial \ln L/\partial\theta_i = 0$ into $\partial \ln L/\partial\theta_{i-1} = 0$, $\partial \ln L/\partial\theta_{i-1} = 0$ for ith stage becomes identical to $\partial \ln L/\partial\theta_{i-1} = 0$ for $(i-1)$st stage. Therefore the likelihood equations for ith stage are composed of the likelihood equations for $(i-1)$st stage and two additional equations given by Equations (11). Letting $\hat{\theta}_{j,i} = \hat{\theta}_{j,(i-1)}$ and $\hat{p}_{jSS,i} = \hat{p}_{jSS,(i-1)}$ for $j \le (i-1)$ and substituting them into Equations (11), it is easily shown that Equations (11) are satisfied only when $p_{iSS} = x_{iSS}/n_i$, and $\theta_i = (n_{1F}/n_1) - \sum_{j=2}^{i}(x_{jFS}/n_j)$ Therefore, the estimates given in Expression (8) are the unique solution of the likelihood equations for ith stage. Now proof is completed.

One noteworthy point is that the estimates obtained in the previous stages do not change as the testing proceeds. Thus we can simply rename $\hat{\theta}_{j,i}$ and $\hat{p}_{jSS,i}$ as $\hat{\theta}_j$ and $\hat{p}_{jSS}$, where $\hat{\theta}_1 = x_{1F}/n_1$, $\hat{\theta}_j = x_{1F}/n_1 - \left(\sum_{k=2}^{j} x_{kFS}/n_k\right)$ and $\hat{p}_{jSS} = x_{jSS}/n_j$. We can further derive a useful relationship

$$\hat{\theta}_i = \hat{\theta}_{i-1} - \frac{x_{iFS}}{n_i} \quad \text{for} \quad i \geq 2, \tag{12}$$

which enables us to obtain the new estimate of the failure probability by updating the estimate obtained in the previous testing stage. In order to characterize MLEs statistically, we compute the expected value and variance of $\hat{\theta}_i$. Since

$$\begin{aligned} E(\hat{\theta}_i) &= E\left(\hat{\theta}_1 - \sum_{j=2}^{i} \frac{x_{jFS}}{n_j}\right) \\ &= E\left(\frac{x_{1F}}{n_1}\right) - \sum_{j=2}^{i} E\left(\frac{x_{jFS}}{n_j}\right) \\ &= \frac{1}{n_1} n_1 \theta_1 - \sum_{j=2}^{i} \frac{1}{n_j} n_j p_{jFS} \\ &= \theta_1 - \sum_{j=2}^{i} p_{jFS} \\ &= \theta_1 - \sum_{j=2}^{i} (\theta_{j-1} - \theta_j) \\ &= \theta_i , \end{aligned} \tag{13}$$

$\hat{\theta}_i$ is an unbiased estimator for θ_i.

$$\begin{aligned} V(\hat{\theta}_i) &= V\left(\hat{\theta}_1 - \sum_{j=2}^{i} \frac{x_{jFS}}{n_j}\right) \\ &= V(\hat{\theta}_1) + \sum_{j=2}^{i} V\left(\frac{x_{jFS}}{n_j}\right) \\ &= \frac{1}{n_1} \theta_1 (1-\theta_1) + \sum_{j=2}^{i} \frac{p_{jFS}(1-p_{jFS})}{n_j} \\ &= \frac{1}{n_1} \theta_1 (1-\theta_1) + \sum_{j=2}^{i} \frac{(\theta_{j-1}-\theta_j)(1-\theta_{j-1}+\theta_j)}{n_j} \end{aligned} \tag{14}$$

and

$$Var(\hat{\theta}_i) = Var(\hat{\theta}_{i-1}) + \frac{(\theta_{i-1}-\theta_i)(1-\theta_{i-1}+\theta_i)}{n_i} .$$

Thus

$$\widehat{Var}(\hat{\theta}_i) = \widehat{Var}(\hat{\theta}_{i-1}) + \frac{x_{iFS}(n_i - x_{iFS})}{n_i^3} .$$

An estimate of $V(\hat{\theta}_i)$ is obtained by substituting θ_j's in Expression (14) with $\hat{\theta}_j$. Similarly we can show that $\hat{p}_{iSS}$ is also unbiased and its variance is $p_{iSS}(1-p_{iSS})/n_i$. Taking advantage of the asymptotic normality of MLEs, interval estimation and hypothesis testing on θ_i's and p_{iSS}'s can be performed.

4. Comparison with Similar Approaches for the Maintenance Phase

Recently Podgurski and Weyuker [12] and Dasu and Weyuker [3] proposed economical approaches for estimating reliabilities of successive software versions resulting from the maintenance of software. Considering only two successive software versions during the maintenance phase, they proposed a heuristic algorithm for updating the previous estimate of reliability. Since their testing procedure is the same with ours, that is, test inputs are applied to two successive software versions, it seems desirable to compare each other. For the sake of comparison, we use the same notations defined in the previous two sections. Podgurski and Weyuker [12] developed a method for estimating θ_i under the following assumptions.

(*A1*) A program P_{i-1} was modified to obtain a program P_i with the same requirements. That is, P_{i-1} was modified to correct a defect or to improve its efficiency, not to add or remove functionality.

(*A2*) P_{i-1} and P_i have the same interface, so that a valid input to P_{i-1} is also a valid input to P_i.

(*A3*) The requirements for P_{i-1} and P_i define a function from inputs to outputs.

(*A4*) It is reasonable to view usage of P_{i-1} and P_i as random sampling from a shared operational distribution, which will not change in the near future.

(*A5*) The reliability measures of interest are the probabilities θ_{i-1} and θ_i that P_{i-1} and P_i fail, respectively, when executed on a random operational input.

($A6$) We possess an estimate $\hat{\theta}_{i-1}$ of θ_{i-1} and seek estimate $\hat{\theta}_i$ of θ_i.

($A7$) The modifications made to P_{i-1} to yield P_i affect a small proportion of operational executions.

($A8$) The cost of executing P_{i-1} and P_i is small relative to the cost of checking whether an execution conforms to requirements.

We first note that assumptions ($A1$) ~ ($A3$) and ($A7$) hold in the testing and debugging phase It is because the specification S is available and generally remain unchanged during the testing and debugging phase. If we assume that the operational profile is given and the testing profile is the same with the operational profile, assumption ($A4$) also holds in the testing and debugging phase Therefore the method of Podgurski and Weyuker[12] is applicable to the testing and debugging phase. As in Equations (5) and (6), we have

$$\theta_{i-1} = \Pr(P_{i-1} \text{ and } P_i \text{ both fail}) + \Pr(P_{i-1} \text{ fails and } P_i \text{ succeeds}) = p_{iFF} + p_{iFS}$$

and

$$\theta_i = \Pr(P_{i-1} \text{ and } P_i \text{ both fails}) + \Pr(P_{i-1} \text{ succeeds and } P_i \text{ fails}) = p_{iFF} + p_{iSF}.$$

where FF is the union of outcomes FFS and FFD. Thus

$$\theta_i = \theta_{i-1} - p_{iFS} + p_{iSF}. \qquad (15)$$

Podgurski and Weyuker[12] suggested x_{iFS}/n_i and x_{iSF}/n_i as estimates of p_{iFS} and p_{iSF} respectively. Inserting the available estimate of θ_{i-1} and estimates of p_{iFS} and p_{iSF} into Equation (15), the failure probability of the new version is obtained as $\hat{\theta}_{i-1} - (x_{iFS}/n_i) + (x_{iSF}/n_i)$ If the debugging is assumed to be perfect, $\hat{\theta}_i$ is estimated as $\hat{\theta}_{i-1} - - (x_{iFS}/n_i)$, which is identical to the recursive algorithm given in Expression (12). However, they did not make any comment on the estimate $\hat{\theta}_{i-1}$. That is, it is assumed that $\hat{\theta}_{i-1}$ is available or given. And only heuristics for updating algorithms are given. Therefore the results of the previous section provide Podgurski and Weyuker[12] with statistical justification

On the other hand Dasu and Weyuker[3] considered the same problem and proposed a different estimation method. First note that

$$\begin{aligned}\theta_i &= \Pr(P_i \text{ fails}) \\ &= \Pr(P_i \text{ fails and } P_{i-1} = P_i) + \Pr(P_i \text{ fails and } P_{i-1} \neq P_i) \\ &= \Pr(P_i \text{ fails} \mid P_{i-1} = P_i)\Pr(P_{i-1} = P_i) + \Pr(P_i \text{ fails} \mid P_{i-1} \neq P_i)\Pr(P_{i-1} \neq P_i),\end{aligned}$$

where $P_{i-1} \neq P_i$ represents the part of the input domain where P_{i-1} and P_i behave differently. Defining $n_{id} = x_{iFS} + x_{iSF} + x_{iFFD}$ and $n_i - n_{id} = x_{iSS} + x_{iFFS}$, $\Pr(P_{i-1} = P_i)$, $\Pr(P_{i-1} \neq P_i)$ and $\Pr(P_i \text{ fails} \mid P_{i-1}\ P_i)$ can be estimated by $(n_i - n_{id})/n_i$, n_{id}/n_i and x_{iSF}/n_i, respectively Further arguing that $\Pr(P_i \text{ fails} \mid P_{i-1} = P_i) = \Pr(P_{i-1} \text{ fails})$, they suggested

$$\hat{\theta}_i = \hat{\theta}_{i-1} \frac{n_i - n_{id}}{n_i} + \frac{x_{iSF}}{n_i}. \qquad (16)$$

However, it should be noted that $\Pr(P_i \text{ fails} \mid P_{i-1} = P_i) = \Pr(P_{i-1} \text{ fails})$ does not in general hold. This is satisfied when the operational profile is uniform. Thus the updating algorithm given in Equation (16) is applicable to the case of the uniform operational profile.

5. Conclusions

A number of software reliability growth models have been developed so far, which are based on the assumptions about software development and usage environment. In order to overcome the dependency on the assumptions, we suggested an input domain-based SRGM. The suggested input domain-based SRGM is based on a multi-stage testing procedure in which both the software systems before and after fault corrections are tested simultaneously at each stage. The suggested model was then statistically characterized and compared with similar approaches developed for the maintenance phase. The useful algorithm for updating the previous estimate is shown to be identical to the updating algorithm of Podgurski and Weyuker [12] developed for the maintenance phase. That is, our study also provides the heuristic of Podgurski and Weyuker [12] with statistical justification However, its practicability should be validated and examined through applying to real software testing. Since the multi-stage testing procedure requires more testing time, the trade-off between testing time and number of required inputs should be investigated in order to determine the time to stop testing We also need to extend the input domain-based SRGM for imperfect debugging environment.

References

[1] J. R. Brown and M Lipow, "Testing of Software Reliability," in Proc Int Conf. Reliable Software, Los Angeles, CA, pp.518-527, April, 1975

[2] R. W. Butler and G. B. Finelli, "The Infeasibility of Quantifying the Reliability of Life-Critical Real-Time Software," IEEE Trans on Software Engineering, Vol.19, pp.3-12, 1993.

[3] T Dasu and E. J. Weyuker, "Updating Software Reliability Subject to Resource Constraints," Proc. American Statistical Association Proc. Joint Statistical Meeting, 1999.

[4] T. Downs and P. Garrone, "Some New Models of Software Testing with Performance Comparisons." IEEE Trans. on Reliability, Vol 40, pp.322 328, 1991

[5] J. R. Dunham, "Experiments in Software Reliability : Life-Critical Applications," IEEE Trans on Software Engineering, Vol, SE-12, pp.110-123, 1986.

[6] J W. Duran and S. C Ntafos, "An Evaluation of Random Testing," IEEE Trans Soft. Eng., SE-10. pp.438-444, July, 1984.

[7] A. L. Goel, "Software Reliability Models : Assumptions, Limitations, and Applicability," IEEE Trans. on Software Eng., pp.1411-1423, Dec. 1985.

[8] B. Littlewood, "Software Reliability Modelling : Achievements and Limitations," Proc COMPEURO. Bologna, 1991.

[9] W. H. MacWilliams, "Reliability of Large Real-Time Software," in Proc IEEE Symp. Computer Software Reliability, New York, pp.1-6, May, 1973.

[10] J. D. Musa, "Sensitivity of Field Failure Intensity to Operational Profile Errors," Proceedings of the 5th International Symposium on Software Reliability Engineering, Monterey, Calif., November 6-9, pp.334-337, 1994.

[11] E. Nelson, "Estimating software reliability from test data," Microelectronics Reliability, Vol 17, pp.67-74, 1978.

[12] A. Podgurski and E. J. Weyuker, "Re-estimation of Software Reliability After Maintenance." IEEE Trans. on Rel., Vol 1, pp.79-85, 1998

[13] C. V. Ramamoorthy and F. B. Bastani, "Software Reliability-Status and Perspectives," IEEE Trans. Soft Eng., SE-8, No 4. pp.354-371, July, 1982.

[14] S N Weiss and E J Weyuker, "An Extended Domain-Based Model of Software Reliability." IEEE Trans on Software Eng , Vol.SE-14, No. 10, pp 456-470, 1988

[15] J.-Y Park, S.-U. Lee and J.-H. Park, "Neural Network Modeling for Software Reliability Prediction from Failure Time Data," Journal of Electrical Engineering and Information Science, Vol 4, pp.533 538, 1999.

[16] J.-Y Park, S -U. Lee and J.-H. Park, "Software Reliability Prediction Using Predictive Filter," Transactions of Korean Information Processing Society, Vol.7, No. 7, pp.2076-2085, 2000.

박 중 양

e-mail : parkjy@nongae.gsnu.ac.kr

1982년 연세대학교 응용통계학과 (학사)

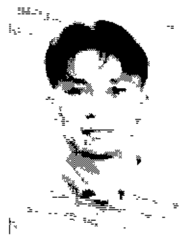
1984년 한국과학기술원 산업공학과 응용통계전공(석사)

1994년 한국과학기술원 산업공학과 응용통계전공(박사)

1984년~1989년 경상대학교 전산통계학과 교수

1989년~현재 경상대학교 통계학과 교수

관심분야 : 소프트웨어 신뢰성, 신경망, 선형 통계 모형, 실험계획법 등



서 동 우

e-mail : seo3129@netian.com

1997년 경상대학교 통계학과 (학사)

1999년~현재 경상대학교 대학원 통계학과(석사 수료)

관심분야 : 소프트웨어 신뢰성, 신경망, 선형 통계 모형, 데이터베이스 등

김 영 순

e-mail : coaskim@hanmail.net

1994년 경상대학교 통계학과 (학사)

1998년 경상대학교 통계학과 (석사)

2000년~경상대학교 통계학과 (박사 과정)

관심분야 : 소프트웨어 신뢰성, 신경망, 선형 통계 모형, 실험계획법 등