A Study on Change-Point Software Reliability Modeling and Optimization for Development Management

Dissertation submitted in partial fulfillment
for the degree of Doctor of Philosophy
(Engineering)

Yuka MINAMINO

Under the supervision of
Professor Shigeru Yamada

Doctoral Program of Graduate School of Engineering,
Tottori University, JAPAN

January 2017

ABSTRACT

Software development managers should conduct development management properly for achieving the quality required from the user. The major factors under the development planning are the quality, cost, and delivery (QCD), and they are needed to be satisfied enough. However, there are several problems for software development management. For example, when the software development managers make a decision by using the existing development management methods, the amount of information for the decision-making is not enough. That is, the existing development management methods are not realistic and applicable because they are not reflected the actual development environment enough. From these backgrounds, this doctoral thesis discusses the software development management methods based on software reliability growth models (SRGMs). Especially, we develop them in terms of the software development optimization. Concretely, we treat these problems: a change-point problem, an optimal software release problem and an optimal testing-resource allocation problem.

This thesis is composed of the following chapters: Chapter 2 discusses a change-point modeling framework and a detection method. Chapter 3 discusses an analytical optimal software release problem based on the change-point model. Chapter 4 and 5 discuss an optimal software release problem and an optimal testing-resource allocation problem based on the multi-attribute utility theory (MAUT).

In Chapter 2, the change-point modeling framework based on typical NHPP (nonhomogeneous Poisson process) models and the detection method are discussed. The testing-time when the stochastic characteristics of software failure-occurrence time-intervals change due to a change of testing-environment is called change-point. It is known that the occurrence of the change-point influences the accuracy of the SRGM-based software reliability assessment. The existing SRGMs are not introduced the concept although the change-point is known as a problem in the actual testing-environment. Furthermore, the change-point is caused by software development management or technical aspects of software development. In the former case, the change-point is determined sensuously by the software development managers. On the other hand, in the latter case, the change-point occurs as the natural phenomenon. Nevertheless, there is not the theoretical method for detecting the change-point. Therefore, we improve the accuracy of existing SRGMs by reflecting the actual testing-environment as the aim of this chapter. In this thesis, we develop specific NHPP models with the change-point, and confirm that the proposed models have better performance than the existing models. Also, we apply the Laplace trend test for developing the detection method. The Laplace trend test is a method for observing the time when the trend of software reliability growth process changes. Concretely, we confirm the effectiveness as a change-point detection method by comparing the goodness-of-fit for the existing models and our models applied the estimated change-points. Our change-point detection method enables us to evaluate whether the software development managers could determine the change-point properly or not.

In Chapter 3 and 4, the optimal software release problem is discussed. It is an optimization problem on estimating optimal shipping time of the software product. Normally, SRGMs are applied for discussing the optimal software release problem. However, the actual testing-environment is not reflected to this problem because the change-point is not considered. Therefore, we discuss the optimal software release problem based on the change-point model. We can estimate the optimal release time and optimal testing-time duration from the change-point

to the termination time of testing simultaneously by formulating a cost function based on a change-point model. Furthermore, normally, when the optimal release time is determined, an evaluation criterion tends to be focused. Since there is not the method for considering the multiple evaluation criteria simultaneously. Therefore, we apply the multi-attribute utility theory for solving them. MAUT is the utility theory on decision-making with considering multiple constraints. Additionally, the decision-making for the development management is evaluated by using the utility as an evaluation criterion. The optimal release time and optimal testing-time duration from the change-point to the termination time of testing are derived as the optimal solutions by solving the optimization problem.

In Chapter 5, we discuss an optimal testing-resource allocation problem based on MAUT. This problem is an optimization problem for allocating the testing-resource in a module testing. We need to develop the method for the optimal allocation with considering multiple constraints because an evaluation criterion tends to be focused as with Chapter 4. Therefore, we estimate the optimal testing-resource expenditures for each module and utility by using MAUT. Additionally, the expected number of remaining faults is estimated by using the optimal testing-resource expenditures. Each chapter shows numerical examples by using actual data sets. The final chapter summarizes the results obtained in this doctoral thesis, and remark the conclusion.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Professor Shigeru Yamada at Tottori University, the supervisor of my study and the chairman of this thesis reviewing committee, for his introduction to research in software reliability engineering, many invaluable comments, continual support, guidance, and warm encouragement.

I am indebted to Professor Akira Kitamura at Tottori University, Associate Professor Junji Koyanagi at Tottori University, and Associate Professor Yoshinobu Tamura at Yamaguchi University, the members of this thesis reviewing committee, for their helpful comments and checking the manuscript.

I would like to greatly acknowledge Assistant Professor Shinji Inoue at Tottori University, for his many valuable comments, continual support, and warm encouragement.

I would like to thank many professors and researchers who provide me with invaluable comments for my study and warm encouragement. I also acknowledge the encouragement of the professors of Department of Social Systems Engineering of Tottori University.

Special thanks are due to Mr. Mitsuho Matsumoto, Mr. Ryosuke Kii, Ms. Saki Taniguchi, Mr. Yasutaka Iwasaki, and the other past and present members of Information Systems Laboratory of Department of Social Systems Engineering at Tottori University. This work could not be achieved without their continual warm encouragement.

Finally and most importantly, I wish to thank my parents (Mr. Masato Minamino and Ms. Kayoko Minamino), my sister (Ms. Yuki Minamino/Shimizu), for their understanding and warm support. It is to them that this thesis is dedicated.

Yuka Minamino
January 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Software Quality/Reliability

Computer systems are necessary in our social life. For example, there are seat reservation systems, online transaction systems, hospital patient monitoring systems, air traffic control systems, and so forth. If a failure occurs in those systems, there is a potential to have negative effects on human life and social life. The environment for software development becomes more diversified, foreshorten delivery, large-scale, and complex. However, the users require high-quality software products.

"Software quality" is defined as the attribute measuring how well the software product meets stated user's functions and requirements in Japanese Industrial Standards Committee (JIS Z 8101). Quantitative measuring and evaluating of software quality systematically are important for accurate management. ISO/IEC (International Organization for Standardization/International Electrotechnical Commission) standardized the software quality (ISO/IEC 9126) gives us software quality characteristics and sub-characteristics. The quality characteristics are also called *"external characteristics"*. The external characteristics are the quality characteristics defined through the eyes of users to meet the users' quality requirements. Fig. 1.1 shows the six quality characteristics and sub-characteristics [36, 37, 38, 41]. *"Reliability"* in these quality characteristics is equivalent to the characteristic of *"must-be quality"*. That is, the reliability must be satisfied enough because incomplete reliability gives a feeling of dissatisfaction to the users.

"Software reliability" is defined as the ability of software to maintain the performance required under a predetermined time-period and conditions [36]. It is also defined as the probability of failure-free software operation for a specified period of time in a specified environment [19]. Therefore, it is important to detect and remove *"software failures"* and *"software faults"*

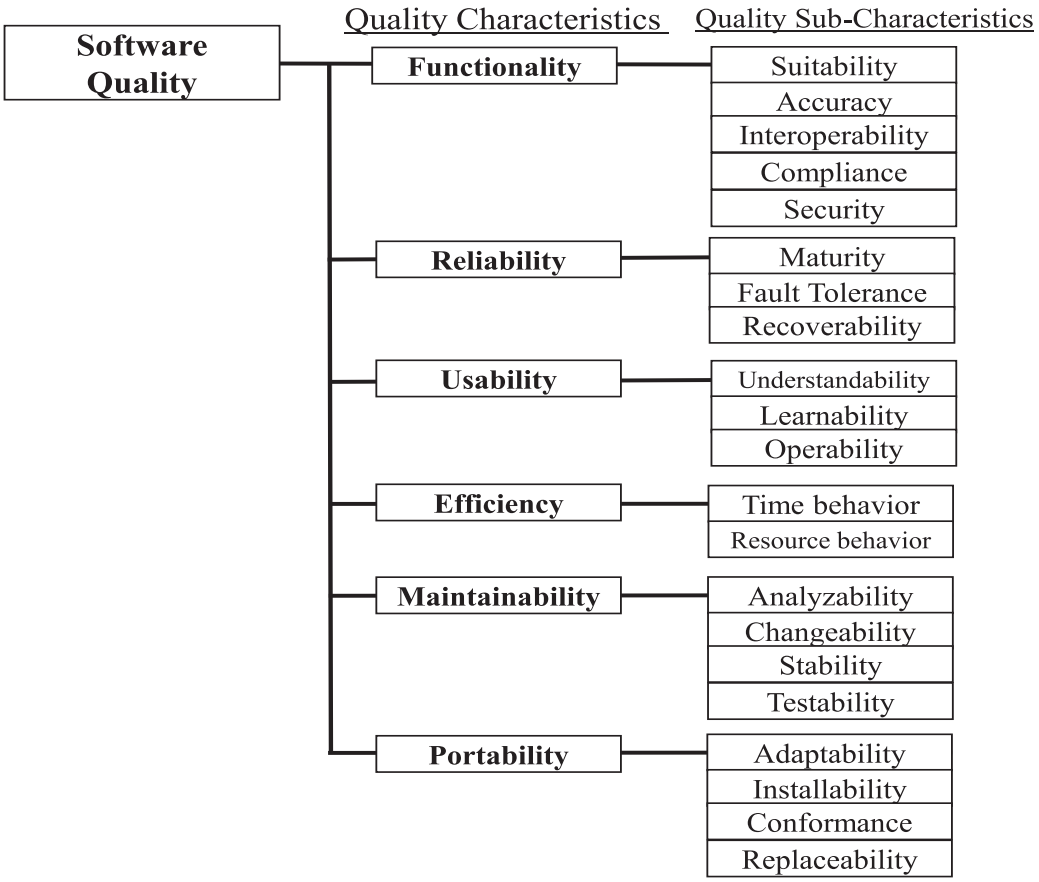

Fig. 1.1 : Software quality characteristics and sub-characteristics (ISO/IEC 9126) [36].

in the testing-phase for ensuring the software quality/reliability as far as possible. The software failures and software faults are defined as follows [37].

- *Software failure* : An unacceptable departure of program operation from the program requirements.

- *Software fault* : A defect in the program which causes a software failure. The software fault is usually called a *software bug*.

- *Software error* : Human action that results in the software system containing a software fault.

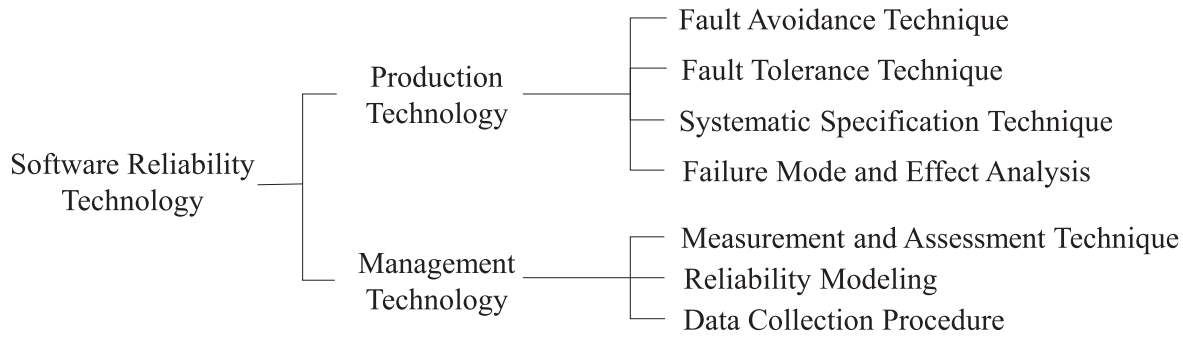

Fig. 1.2 : Software Reliability Technologies [35].

Various technologies, such as measurement, assessment, and reliability technologies, are needed for improving software reliability. They are called *"Software reliability Technology"* [35]. Software reliability technology is divided as the production technology and management technology. Fig. 1.2 shows these technologies and the classification. Furthermore, the software reliability technology in terms of engineering is defined as *"software reliability engineering (SRE)"*. The software reliability engineering is aimed at the measurement, prediction, and management of software reliability for maximizing the user satisfaction.

1.2 Software Development

Generally, software products are developed through the phases of specification, design, coding, and testing, as shown in Fig. 1.3. A general software development process of water-fall paradigm is shown in Fig. 1.4 [33, 36, 37, 38, 41]. The testing-phase is the most important activity because the final quality/reliability of software products is confirmed quantitatively by using software reliability assessment technologies.

Furthermore, when the software development managers want to know the problems for the software development management, as shown in Fig. 1.3, the software reliability assessment technologies are used. Especially, they need to know the following things [13].

- The failure rate of software released now.

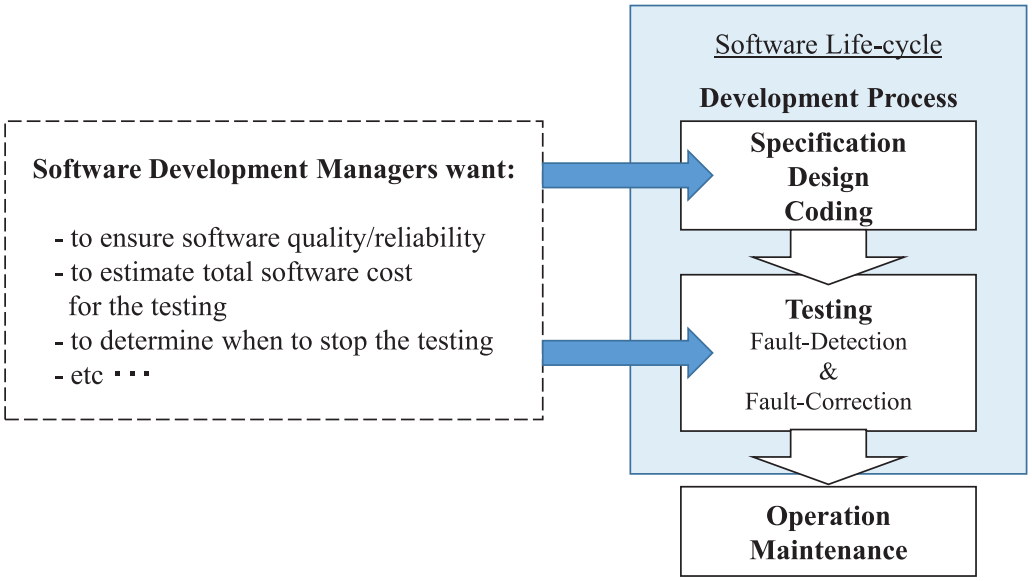

Fig. 1.3 : Aim of software quality/reliability measurement and assessment.

- The number of remaining faults in the software, the location of faults, and the priority of fault-removing activity.
- The testing-time to achieve software reliability targets.
- The schedule to spend testing-resources efficiently.

These above things should be answered with the help of software reliability models [8, 10, 13, 35, 36, 37, 38] as a technology of software reliability assessment. The software reliability model is a tool which is used to evaluate the software reliability quantitatively, and monitor the change in reliability performance [10]. The testing is advanced by the module testing, integration testing, and system testing. These tests are dynamic analysis, and they are conducted as follows [10, 36, 37, 38, 40, 41, 44]: First, the individual software modules are tested independently

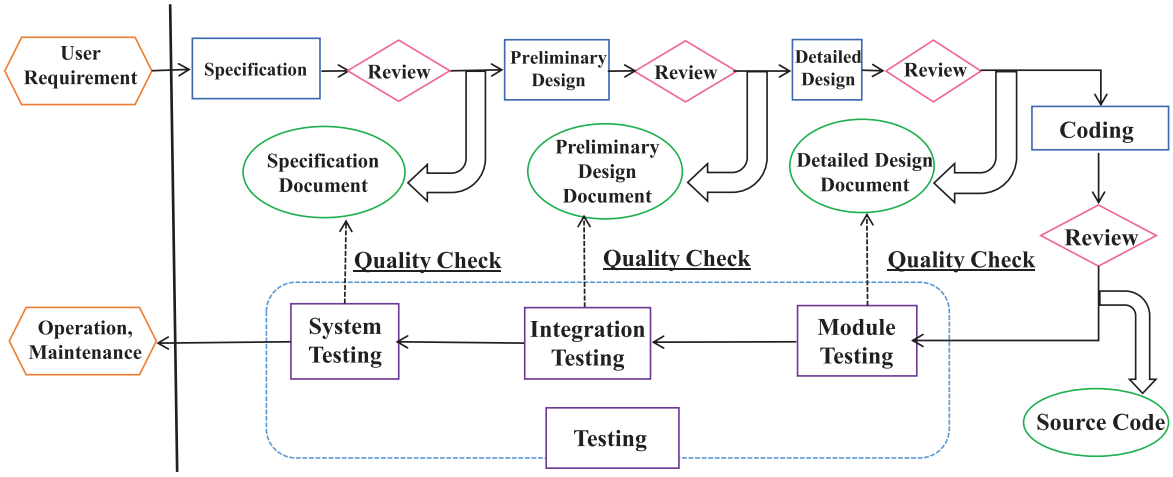

Fig. 1.4 : A general software development process (water-fall paradigm) [37].

in the module testing. Next, the interfaces of software modules are tested in the integration testing. Finally, the software system interconnected all software modules is evaluated in terms of quality/reliability in the system test. When we use software reliability assessment technologies to measure the software quality/reliability in these testing-phases, these results are utilized for not only software reliability assessment but also software development management in some cases. Notably, the software development management is important to be conducted from the perspective of QCD (Q: quality, C: cost, D: delivery). However, there is a variety of problems for software development management when we consider the QCD. For example, an *"optimal software release problem"* [3, 27, 28, 30, 33, 37, 38] and an *"optimal testing-resource allocation problem"* [10, 17, 33, 37, 38, 42, 43] are included as typical problems. Also, these problems are treated as optimization problems because the optimal solutions, such as the optimal software release time and the optimal testing-resource expenditures, are needed to obtain. Therefore, we solve these optimization problems under some constraints in terms of QCD, and derive the optimal solutions. We introduce these software development management problems concretely in the later section.

1.3 Software Reliability Modeling

The software reliability models as a base technology for quantitative software reliability assessment have been discussed well so far. Especially, software reliability growth models (SRGMs)

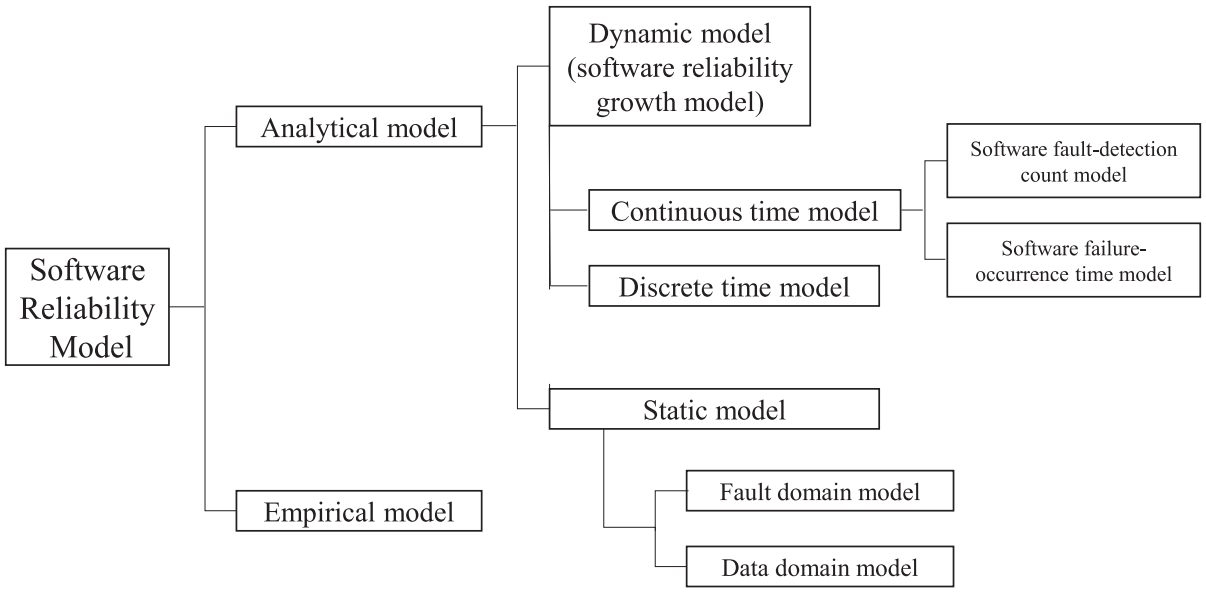

Fig. 1.5 : Hierarchical classification of software reliability models [35].

[8, 15, 37, 38] are mathematical models, and they are categorized as dynamic models in the software reliability models. Fig. 1.5 [37] shows the hierarchical classification of the software reliability models. The SRGMs describe the software failure-occurrence frequency or the software failure-occurrence time-intervals as random variables. Also, they treat the time-dependent behavior of the software failure-occurrence phenomenon as the software reliability growth process. The SRGMs are classified into the following three categories [36, 37, 39].

- *Software failure-occurrence time model*:
  The model which is based on the software failure-occurrence time or the software fault-detection time.

- *Software fault-detection count model*:
  The model which is based on the number of software failure-occurrence or the number of

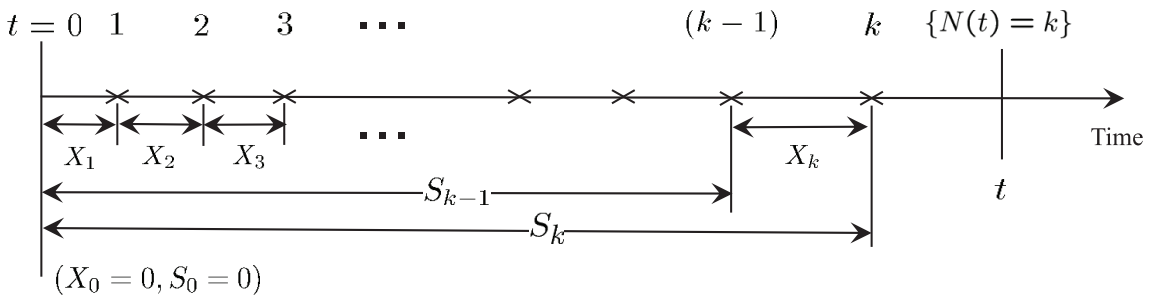

Fig. 1.6 : The stochastic quantities related to the software fault-detection and software-failure occurrence phenomenon [36].

detected faults.

- *Software availability model*:
  The model which describes the time-dependent behavior of software system alternating up (operation) and down (fault correction) states.

As the concept of the software reliability growth modeling, the following stochastic quantities related to the software fault-detection and the software failure-occurrence phenomenon are defined,as shown in Fig. 1.6.

$N(t)$: The cumulative number of faults detected up to time t
(or the cumulative number off software failures observed up to time t).
S_k: The k-th software failure-occurrence time ($k = 1, 2, \cdots$; $S_0 = 0$).
X_k: The time-interval between $(k-1)$-st and k-th software failures ($k = 1, 2, \cdots; X_0 = 0$).

Fig. 1.6 shows the occurrence of event $N(t) = k$ since i faults have been detected up to time t. Also, we treat $N(t)$, S_k, and X_k as the random variables. That is, we have the following relationship:

$$S_i = \sum_{k=1}^{i} X_i, \qquad X_k = S_k - S_{k-1}. \tag{1.1}$$

We can derive several software reliability assessment measures by assuming the probability distribution of $N(t)$, S_k, and X_k. The examples of software reliability assessment measures include the following:

- Expected number of remaining faults in the software.
- Software reliability function
- Mean time between software failures (MTBF)
- Hazard rate/failure rate

1.4 Background and Purposes of This Thesis

Generally, the SRGMs are constructed under assumptions that the stochastic characteristics for these quantities are same throughout the testing-phase. However, we often observe that the stochastic characteristics of software failure-occurrence time-intervals change due to a change of testing-environment. Testing-time observed such phenomenon is called change-point [1, 2, 5, 6, 20, 29, 32, 46], and is considered one of the factors which influences accuracy of the software reliability assessment.

Under the background, the software reliability growth modeling frameworks have discussed in the literatures [1, 2, 4, 5, 6, 20, 29, 31, 32, 46]. However, they are considered only the difference of software failure-occurrence rate before the change-point and those after change-point. It might be natural to consider that there is relationship of the software failure-occurrence time-interval before change-point and those after change-point, in the actual testing-phase. Since the same software product is tested even if the testing-environment changes during the testing-phase. Therefore, developing SRGMs with the effect of the change-point is expected to conduct the software reliability assessment with high accuracy. Concretely, we develop change-point models in cases that each mean value function with the change-point follows exponential, delayed S-shaped and inflection S-shaped SRGMs [9, 18, 37, 38]. Additionally, we introduce uncertainty of the testing-environment into the existing models in the cases that each mean value function follows the exponential and delayed S-shaped SRGMs, and develop more specific change-point models. Furthermore, change-point has not been determined by theoretical methods. For example, when a change-point is caused by software development management, it is determined by the software development managers based on their experience. Therefore, we need to discuss a theoretical change-point detection method.

The SRGMs are often used to discuss problems for software development management. Although ensuring the quality/reliability of software products is important, the evaluation of other factors, such as the stability and degree of progress, is also important. For example, as a

problem for software development management, there is an optimal software release problem. It is an optimization problem on estimating the optimal shipping time of software products. The existing methods to determine the optimal software release time are based on the reliability, cost, and delivery evaluation criteria. That is, the optimal software release time is determined by minimizing the total software cost when we use the existing method based on the cost evaluation criterion. However, the minimum cost is not necessarily important actually because it is more realistic that we consider not only the cost evaluation criterion but also many evaluation criteria simultaneously. From such backgrounds, the optimal software release problem with considering multiple evaluation criteria is beneficial.

In this thesis, we also discuss an optimal testing-resource allocation problem. It is an optimization problem for allocating testing-resources in a module testing. Most of existing optimal testing-resource allocation problems have been focused on the reliability because the improvement of quality increases the productivity by the following reasons: First, if the quality is high, the overhead work decreases and total software cost decreases. Next, high quality is ensured by the detection of software fault in the upstream process because the debugging cost for the upstream process is lower than the debugging cost for the downstream process. However, the testing-resources for the improvement of quality are not infinite, and there are some constraints for the evaluation criteria. That is, we need to expend the testing-resources for the module testing in consideration with the balance of many evaluation criteria.

In those optimization problems for software development management, it is noted that there is trade-off relationship of those attributes. Also, the management strategy for management aspects is not reflected the optimal solutions completely. The management strategy should differ by developed software products because they are diversified. That is, we introduce the multi-attribute utility theory (MAUT) [12, 21, 23, 24, 25, 26] into those problems, and estimate the optimal solutions in terms of the utility theory.

1.5 Organization of This Thesis

This thesis is composed of the following chapters. Chapter 2 discusses software reliability growth models with the change-point and a change-point detection method based on the Laplace trend test. Chapter 3 and Chapter 4 treat an optimal software release problem. Chapter 5 treats an optimal testing-resource allocation problem. Also, Chapter 3 and 5 are discussed based on the multi-attribute utility theory. Each content of these chapters is as follows:

- **Chapter 2:** *Change-Point Modeling and Detection Method*
  Chapter 2 discusses the software reliability growth modeling framework with a change of the testing-environment for improving the accuracy of the reliability assessment and development management based on SRGMs. Concretely, we develop specific NHPP (non-homogeneous Poisson process) models which the mean value functions before the change-point follow the exponential, the delayed S-shaped, and the inflection S-shaped models, respectively. Furthermore, we introduce the uncertainty of the testing-environment into the exponential and the delayed S-shaped change-point models by using the gamma distribution. We investigate the usefulness of our models by conducting goodness-of-fit comparisons with the existing models. Also, we discuss a change-point detection method based on the Laplace trend test [34, 23] as an application. The Laplace trend test enables us to detect change-points by regarding changing the trend of the software reliability. After that, we check the performances of the change-point models with the detected change-points, and confirm the effectiveness of our approach by the goodness-of-fit comparison.

- **Chapter 3:** *Optimal Software Release Problem Based on a Change-Point Model*
  Chapter 3 discusses an optimal software release problem for software development management. In this chapter, we apply our change-point model in Chapter 2, and define a cost function by using it. Next, we derive the optimal software release time and optimal testing-time duration from the change-point to the termination time of testing, analytically. Finally, we propose the cost-optimal and cost-reliability-optimal software release policies.

- **Chapter 4:** *Optimal Software Release Problem Based on MAUT*
  Chapter 4 discusses an optimal software release problem based on the multi-attribute utility theory. Especially, cost and reliability attributes are developed as evaluation criteria by using a change-point model. Also, we define a weighted multi-attribute utility function, and maximize it. Then, we can estimate the optimal software release time, optimal testing-time duration from the change-point to the termination time of testing, and utility, simultaneously. Additionally, the optimal occurrence-time of change-point and expected total software cost are estimated by using them. Finally, we check the behavior of the multi-attribute utility function and the cost function, and conduct the sensitive analysis.

- **Chapter 5:** *Optimal Testing-Resource Allocation Problem Based on MAUT*
  Chapter 5 discusses an optimal testing-resource allocation problem based on the multi-attribute utility theory. Especially, cost, testing-resource, and reliability attributes are developed as evaluation criteria by using the testing-effort dependent SRGM. As with Chapter 4, we define a multi-attribute utility function with introducing those attributes, and maximize it. We estimate the optimal testing-resource expenditures and utility for the module testing, simultaneously. Then, the expected number of remaining faults is estimated by using the optimal testing-resource expenditures for module testing. Finally, we conduct the sensitive analysis.

Each chapter provides numerical examples by using actual data-sets. The final chapter summarizes the results obtained in this thesis, and remark the conclusion.

The software reliability assessment in terms of SRGMs is discussed in Chapter 2. Also, the software development management in terms of optimization problems is discussed in Chapter 3, 4, and 5. Fig. 1.7 shows the construction of this thesis.

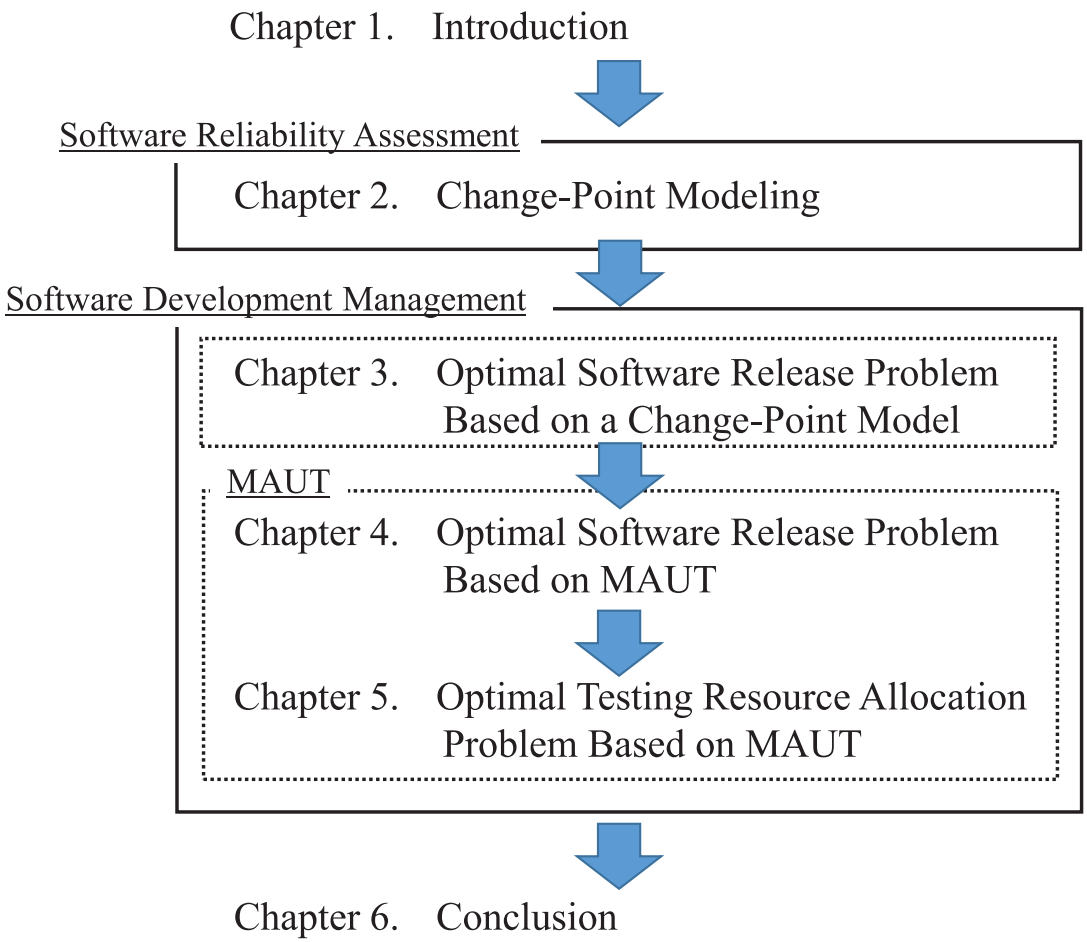

Fig. 1.7 : The construction of this thesis.

Chapter 2

Change-Point Modeling

2.1 Introduction

Generally, software reliability growth models (SRGMs) are developed by treating software failure-occurrence time- or fault detection time-interval as random variables. Also, they are assumed that the stochastic characteristics for these quantities are same throughout the testing-phase. However, this assumption does not enable us to reflect an actual software failure-occurrence phenomenon to software reliability growth modeling because we often observe a phenomenon that the stochastic behavior of software failure-occurrence time-interval notably changes due to a change of testing-environment. Testing-time when such phenomenon is observed is called *"change-point"*. It is known that the occurrence of change-point influences the accuracy of SRGM-based software reliability assessment. Therefore, in this chapter, we develop new NHPP (nonhomogeneous Poisson process) models by introducing the concept of change-point in typical NHPP models [11]. Then, we assume that the same change-points occur in the testing-phase.

Change-point occurs by these aspects in the actual testing-phase as follows [29, 45]:

- A change-point is caused by the aspect of software development management.
  - A case that the software development managers predict that delivery delay would be occurred and reliability requirement would not be achieved.

- A change-point is caused by the technical aspect of software development.
  - Due to difficulty of fault-detection, fault-independence, difference of fault-density of each module.
  - Due to a skill-up process of test-engineers.

In the former case, change-point is determined by the software development managers' decision. On the other hand, in the latter case, the change-point occurs as natural phenomenon without the software development managers' decision. That is, the software development managers don't have methods for detecting or determining change-point logically. For example, J. Zhao and J. Wang proposed a change-point detection method by using the hypothesis testing, and confirmed the certain effectiveness [14]. However, it cannot detect 4 or more change-points. Therefore, we discuss another change-point detection method which is constructed logically. In this chapter, we propose the change-point detection method based on a Laplace trend test. The Laplace trend test is a method for observing the time when the trend of change on the software reliability growth process. We detect 4 change-points by deriving a Laplace factor from actual data sets, and apply them into the EXP2-CP and DSS2-CP models which are proposed in this chapter, respectively. Then, we compare those change-point models by using MSE (mean squared errors), and check the effectiveness of our proposed method.

This chapter consists of the following sections. In Section 2.2, we introduce the existing software reliability growth modeling framework. In Section 2.3 and 2.4, we develop the change-point modeling frameworks. In Section 2.5-2.7, we explain the parameter estimation, the assessment criteria, and reliability assessment measures. In Section 2.8, we explain a Laplace trend test. In Section 2.9, we show numerical examples. In the final section, we conclude with summary.

2.2 Existing Software Reliability Growth Model

Basically, most of NHPP-based SRGMs, in which the total number of detectable faults is finite, can be developed under the following basic assumptions [21]:

(A1) Whenever a software failure is observed, the fault which caused it will be detected immediately and no new faults are introduced in the fault-removing activities.

(A2) Each software failure occurs at independently and identically distributed random times with the probability distribution, $F(t) \equiv \Pr\{T \leq t\} = \int_0^t f(x)dx$, where $\Pr\{A\}$ represents the probability of event A and $f(t)$ the probability density function.

(A3) The initial number of faults in the software, $N_0(> 0)$, is finite, and is treated as a random variable.

Now, let $\{N(t),\ t \geq 0\}$ denote a counting process representing the total number of faults detected up to testing-time t. From the basic assumptions above, the probability mass function that m faults are detected up to testing-time t is derived as

$$\Pr\{N(t)=m\} = \sum_{n} \left(\begin{array}{c} n \\ m \end{array} \right) \{F(t)\}^m \{1-F(t)\}^{n-m} \Pr\{N_0 = n\} \quad (m = 0, 1, 2, \cdots). \quad (2.1)$$

If we assume that the initial fault content, N_0, follows a Poisson distribution with mean ω. The Poisson distribution [16, 35] is given as

$$\Pr\{N_0 = n\} = \frac{\omega^n}{n!}\exp[-\omega]. \quad (2.2)$$

Therefore, we derive as follows:

$$\begin{aligned} \Pr\{N(t)=m\} &= \sum_{n=0}^{\infty} \left(\begin{array}{c} n \\ m \end{array} \right) \{F(t)\}^m \{1-F(t)\}^{n-m} \frac{\omega^n}{n!}\exp[-\omega] \\ &= \exp[-\omega]\frac{\{\omega F(t)\}^m}{m!} \sum_{n=0}^{\infty} \frac{\{\omega(1-F(t))\}^{n-m}}{(n-m)!} \\ &= \frac{\{\omega F(t)\}^m}{m!}\exp[-\omega F(t)] \qquad (m = 0, 1, 2, \cdots). \quad (2.3) \end{aligned}$$

Then, Eq. (2.3) is equivalent to an NHPP with mean value function $\mathrm{E}[N(t)] \equiv \Lambda(t) = \omega F(t)$. An NHPP model for software reliability assessment can be developed by assuming a suitable software failure-occurrence time distribution.

2.3 Software Reliability Growth Model with Change-Point

Now, we define the following stochastic quantities being related to our modeling approach:

X_i: the i-th software failure-occurrence time before change-point.

$(X_0 = 0, \quad i = 0, 1, 2, \cdots)$

S_i: the i-th software failure-occurrence time-interval before change-point.

$(S_i = X_i - X_{i-1}, \quad S_0 = 0, \quad i = 1, 2, \cdots)$

Y_i: the i-th software failure-occurrence time after change-point.

$(Y_0 = 0, \quad i = 0, 1, 2, \cdots)$

T_i: the i-th software failure-occurrence time-interval after change-point.

$(T_i = Y_i - Y_{i-1}, \quad T_0 = 0, \quad i = 1, 2, \cdots)$

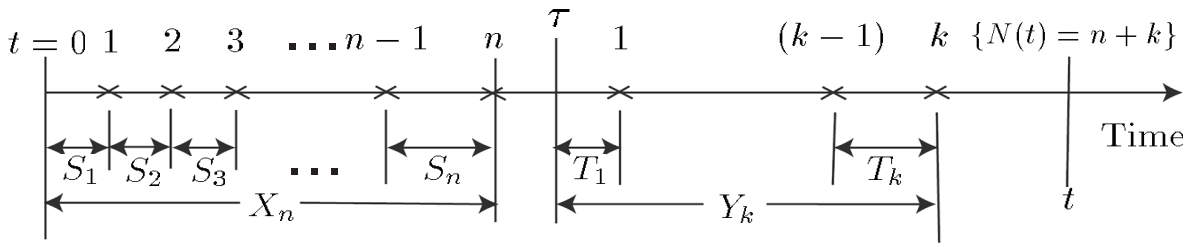

Fig. 2.1 : The stochastic quantities for the software failure-occurrence or fault-detection phenomenon with change-point.

These stochastic quantities for the software failure-occurrence or fault-detection phenomenon with change-point are shown in Fig. 2.1. We can assume that the stochastic quantities before and those after change-point have the following relationships, respectively [7].

$$\begin{cases} Y_i = \alpha(X_i), \\ T_i = \alpha(S_i), \\ K_i(t) = J_i(\alpha^{-1}(t)). \end{cases} \tag{2.4}$$

$\alpha(t)$ is a testing-environmental function representing their relationship between the stochastic quantities of software failure-occurrence time or time-intervals before and those after change-point. $J_i(t)$ and $K_i(t)$ are the probability distribution functions with respect to the random variables S_i and T_i, respectively. Also, we assume that the testing-environmental function is given as

$$\alpha(t) = \alpha t \qquad (\alpha > 0), \tag{2.5}$$

where α is the proportional constant representing the relative magnitude of effect of change-point on the software reliability growth process. Eq. (2.5) is an example for the testing-environmental function. We can get to know the effect of change-point on the software reliability growth process simply by assuming Eq. (2.5) as the testing-environmental function in our modeling framework.

Suppose that n faults have been detected up to change-point, and the fault-detection time from the test-beginning ($t = 0$) have been observed as $0 < x_1 < x_2 < \cdots < x_n \leq \tau$, where τ represents change-point. Then, the probability distribution function of T_1, a random variable representing the time-interval from change-point to the 1-st software failure-occurrence after

change-point, can be derived as

$$\begin{aligned}\bar{K}_1(t) &\equiv \Pr\{T_1 > t\} \\ &= \frac{\Pr\{S_{n+1} > \tau - x_n + t/\alpha\}}{\Pr\{S_{n+1} > \tau - x_n\}} \\ &= \frac{\exp[-\{M_B(\tau + t/\alpha) - M_B(x_n)\}]}{\exp[-\{M_B(\tau) - M_B(x_n)\}]},\end{aligned} \tag{2.6}$$

where $\bar{K}_1(t)$ indicates the cofunction of the probability distribution function, $K_1(t) \equiv \Pr\{T_1 \leq t\}$, i.e., $\bar{K}_1(t) \equiv 1 - K_1(t)$ and $M_B(t)(\equiv \omega K_1(t))$, represents the expected number of faults detected up to change-point, i.e., a mean value function for the NHPP before change-point. From Eq. (2.6), the expected number of faults detected up to $t \in (\tau, \infty]$ after change-point, $M_A(t)$, can be formulated as

$$\begin{aligned}M_A(t) &= -\log \Pr\{T_1 > t - \tau\} \\ &= -\log \bar{K}_1(t - \tau) \\ &= M_B\left(\tau + \frac{t-\tau}{\alpha}\right) - M_B(\tau).\end{aligned} \tag{2.7}$$

Then, the expected number of faults detected up to testing-time t ($t \in [0, \infty), 0 < \tau < t$) can be derived as follows:

$$\Lambda(t) = \begin{cases} \Lambda_B(t) = M_B(t) & (0 \leq t \leq \tau), \\ \Lambda_A(t) = M_B(\tau) + M_A(t) & \\ \qquad = M_B\left(\tau + \frac{t-\tau}{\alpha}\right) & (\tau < t). \end{cases} \tag{2.8}$$

From Eq. (2.8), we can see that an NHPP-based SRGM with change-point can be developed by assuming a suitable probability distribution function for the software failure-occurrence time before change-point. Therefore, when a mean value function before change-point follows an exponential SRGM, we have the following change-point model (EXP-CP model).

$$\Lambda(t) = \begin{cases} \Lambda_1(t) = \omega\{1 - \exp[-bt]\} & (0 \leq t \leq \tau), \\ \Lambda_2(t) = \omega\{1 - \exp[-b(\tau + \frac{t-\tau}{\alpha})]\} & (t > \tau). \end{cases} \tag{2.9}$$

Next, when a mean value function before change-point follows a delayed S-shaped SRGM, we have the following change-point model (DSS-CP model).

$$\Lambda(t) = \begin{cases} \Lambda_1(t) = \omega\{1 - (1 + bt)\exp[-bt]\} & (0 \leq t \leq \tau), \\ \Lambda_2(t) = \omega\{1 - (1 + b(\tau + \frac{t-\tau}{\alpha}))\exp[-b(\tau + \frac{t-\tau}{\alpha})]\} & (t > \tau). \end{cases} \tag{2.10}$$

Finally, when a mean value function before change-point follows an inflection S-shaped SRGM, we have the following change-point model (ISS-CP model).

$$\Lambda(t) = \begin{cases} \Lambda_1(t) = \omega\left(\dfrac{1-\exp\{-bt\}}{1+c\cdot\exp\{-bt\}}\right) & (0 \le t \le \tau), \\ \Lambda_2(t) = \omega\left(\dfrac{1-\exp\{-b(\tau+\frac{t-\tau}{\alpha})\}}{1+c\cdot\exp\{-b(\tau+\frac{t-\tau}{\alpha})\}}\right) & (t > \tau). \end{cases} \tag{2.11}$$

2.4 Change-Point Model with Uncertainty of Testing-Environmental Factor

When we apply a gamma distribution as a probability distribution which can express the difference of the software failure-occurrence phenomenon with uncertainty between before and those after change-point. The gamma distribution [16] is given as

$$f_k(\alpha) = \frac{\theta^k \alpha^{k-1} e^{-\theta\alpha}}{\Gamma(k)} \qquad (k > 0, \alpha > 0, \theta > 0), \tag{2.12}$$

where $\Gamma(k)$ is the gamma function. k and θ are the parameters of the gamma distribution. Consequently, a testing-environmental function is derives as

$$\int_0^\infty \alpha(t) f_k(\alpha) d\alpha = \frac{k}{\theta} t, \tag{2.13}$$

by Eqs. (2.5) and (2.12).

Therefore, when we apply Eq. (2.13) as the testing-environmental factor, it can be formulated as

$$\begin{aligned} M_A(t) &= -\log \Pr\{T_1 > t-\tau\} \\ &= -\log \bar{K}_1(t-\tau) \\ &= M_B\left(\tau + \frac{\theta t}{k}\right) - M_B(\tau). \end{aligned} \tag{2.14}$$

Then, the expected number of faults detected up to testing-time t ($t \in [0,\infty), 0 < \tau < t$) can be derived as follows:

$$\Lambda(t) = \begin{cases} \Lambda_B(t) = M_B(t) & (0 \le t \le \tau), \\ \Lambda_A(t) = M_B(\tau) + M_A(t) & \\ \qquad = M_B\left(\tau + \frac{\theta(t-\tau)}{k}\right) & (\tau < t). \end{cases} \tag{2.15}$$

Therefore, we have a change-point model (EXP2-CP model) in a case that a mean value function before change-point follows an exponential SRGM.

$$\Lambda(t) = \begin{cases} \Lambda_1(t) = \omega\{1 - \exp[-bt]\} & (0 \leq t \leq \tau), \\ \Lambda_2(t) = \omega\{1 - \exp[-b(\tau + \frac{\theta(t-\tau)}{k})]\} & (t > \tau). \end{cases} \tag{2.16}$$

Next, we have a change-point model (DSS2-CP model) in a case that a mean value function before change-point follows a delayed S-shaped SRGM.

$$\Lambda(t) = \begin{cases} \Lambda_1(t) = \omega\{1 - (1 + bt)\exp[-bt]\} & (0 \leq t \leq \tau), \\ \Lambda_2(t) = \omega\{1 - (1 + b(\tau + \frac{\theta(t-\tau)}{k})) \cdot \exp[-b(\tau + \frac{\theta(t-\tau)}{k})]\} & (t > \tau), \end{cases} \tag{2.17}$$

where b is the fault detection rate. l is the inflection coefficient, and c represents $c = (1 - l)/l$ $(0 < l \leq 1)$.

2.5 Parameter Estimation

We estimate the parameters of our models based on a maximum-likelihood method [37, 38]. Suppose that we have observed K data pairs $(t_k, y_k)(k = 0, 1, 2, \cdots, K)$ with respect to the cumulative number of faults, y_k, detected during a constant time-interval $(0, t_k](0 < t_1 < t_2 < \cdots < t_K)$. When the change-point τ and α are given, the log-likelihood function about the probability process $\{N(t), t \geq 0\}$ is the following equation:

$$\begin{aligned} &\ln L(\boldsymbol{\theta}|\tau, \alpha) \\ &\quad = \sum_{k=1}^{K} (y_k - y_{k-1}) \ln[H(t_k; \boldsymbol{\theta}|\tau, \alpha) - H(t_{k-1}, \boldsymbol{\theta}|\tau, \alpha)] \\ &\qquad - H_i(t_K; \boldsymbol{\theta}, \alpha) - \sum_{k=1}^{K} \ln[(y_k - y_{k-1})!], \end{aligned} \tag{2.18}$$

where $L(\boldsymbol{\theta}|\ \tau)$ is the joint probability function or the likelihood function with respect to the probability process $\{N(t), t \geq 0\}$, and $\boldsymbol{\theta}$ is the set of the parameters in the models. We obtain the parameter estimations by solving the log-likelihood equation:

$$\frac{\partial \ln(\boldsymbol{\theta}|\tau, \alpha)}{\partial \boldsymbol{\theta}} = 0, \tag{2.19}$$

with respect to the parameters in the models, respectively.

2.6 Assessment Criteria

We conduct goodness-of-fit comparison in terms of MSE (mean squared errors) [37] and a Kolmogorov-Smirnov (abbreviated as K-S) goodness-of-fit test [38]. The value of MSE is calculated by dividing the sum of the squared vertical distance between the observed and estimated cumulative number of faults, y_k and $\hat{y}(t_k)$, detected during the time-interval $(0, t_k]$, respectively, by the number of observed data pairs. That is, supposing that K data pairs $(t_k, y_k)(k = 1, 2, \cdots, K)$ are observed, we can formulate the MSE as

$$\text{MSE} = \frac{1}{K}\sum_{k=1}^{K}\{y_k - \hat{y}(t_k)\}^2. \tag{2.20}$$

The models having the smallest value of the MSE fit better to the observed data set. Also, the K-S goodness-of-fit test is known as one of the nonparametric goodness-of-fit test techniques. This test is valid for both small and large sample size. The K-S test statistic, D, can be written as

$$\left.\begin{array}{rl} D & = \max\limits_{1\le i\le n-1}\{D_i\} \\ D_i & = \max\left\{\left|\frac{H(t_i)}{H(t_n)} - \frac{y_i}{y_n}\right|, \left|\frac{H(t_i)}{H(t_n)} - \frac{y_{i-1}}{y_n}\right|\right\} \end{array}\right\}. \tag{2.21}$$

The K-S test statistic, D, is needed to be compared with a critical value, $D_{n;\delta}$, where n represents the number of data pairs and δ represents the level of significance. That is, we judge that the applied model fits to the observed data at a level of significance δ if $D < D_{n;\delta}$.

2.7 Reliability Assessment Measures

2.7.1 Expected number of remaining faults

Software reliability assessment measures play an important role in quantitative software reliability assessment based on SRGMs. The expected number of remaining faults by arbitrary testing-time t is formulated as

$$\begin{aligned} M(t) \equiv E[\bar{N}(t)] &= E[N(\infty) - N(t)] \\ &= \Lambda(\infty) - \Lambda(t). \end{aligned} \tag{2.22}$$

2.7.2 Software reliability function

A software reliability function is defined as the probability which a software failure does not occur in the time-interval, $(t, t + x](t \ge 0, x \ge 0)$ given that the testing or the user operation

has been going up to time t. Then, the software reliability function is derived as

$$R(x \mid t) = \exp[-\{\Lambda(t+x) - \Lambda(t)\}], \tag{2.23}$$

if the counting process, $\{N(t),\ t \geq 0\}$, follows an NHPP with mean value function, $\Lambda(t)$. We should note that Eq. (2.23) is derived under the condition which the software system is operated in the same environment as the testing-phase after the change-point.

2.8 Laplace Trend Test

We assume that a change-point is detected by deriving the Laplace factor, $l(k)$, as

$$l(k) = \frac{\sum_{i=0}^{k}(i-1)n(i) - \frac{(k-1)}{2}\sum_{i=0}^{k} n(i)}{\sqrt{\frac{k^2-1}{12}}\sum_{i=0}^{k} n(i)}, \tag{2.24}$$

where $n(i)$ is the number of faults observed during unit time i. The Laplace factor can be interpreted as follows [22, 34]:

1. Negative values indicate a decreasing failure intensity, and thus reliability growth.
2. Positive values indicate an increasing failure intensity, and thus a decrease in the reliability.
3. Values between -2 and +2 indicate stable reliability.

2.9 Numerical Examples

We use the following actual data sets:

DS1: $(t_k, y_k)(k = 1, 2, \cdots, 26\ ;\ t_{26} = 26, y_{26} = 40, \tau = 18)$,
DS2: $(t_k, y_k)(k = 1, 2, \cdots, 29\ ;\ t_{29} = 29, y_{29} = 73, \tau = 24)$,
DS3: $(t_k, y_k)(k = 1, 2, \cdots, 26\ ;\ t_{26} = 26, y_{26} = 34, \tau = 17)$,
DS4: $(t_k, y_k)(k = 1, 2, \cdots, 29\ ;\ t_{29} = 29, y_{29} = 25, \tau = 18)$,
DS5: $(t_k, y_k)(k = 1, 2, \cdots, 28\ ;\ t_{28} = 28, y_{28} = 43, \tau = 18)$,

where t_k is measured on the basis of days, and y_k represents the total number of faults detected

during $[0, t_k]$. These actual data sets were collected from actual testing-phases of the web systems. DS1 and DS2 are the S-shaped software reliability growth curved data. DS3, DS4, and DS5 are the exponential software reliability growth curved data. Also, the change-point is known by changing the tester and increasing the test personnel.

2.9.1 Change-point models

We show numerical examples of the EXP-CP, DSS-CP, and ISS-CP models. Figs. 2.2, 2.3, and 2.4 show the time-dependent behavior of the estimated mean value functions with the effect of the change-point on the software reliability growth process, and its 95% confidence limits. From these figures, the time-dependent behavior of the estimated software reliability growth curves changes at the change-point along with the actual behavior. As an example, we estimated the parameters of the EXP-CP model for DS5 as follows: $\hat{\omega} = 44.8226$, $\hat{b} = 0.0908$, and $\hat{\alpha} = 0.5789$. The expected number of remaining faults and software reliability were estimated as follows: $\hat{M}(28) \approx 1.823 \approx 2$ and $\hat{R}(1.0|28.0) \approx 0.7676$. Next, we estimated the parameters of the DSS-CP model for DS3 as follows: $\hat{\omega} = 35.343$, $\hat{b} = 0.1637$, and $\hat{\alpha} = 0.6425$. The expected number of remaining faults and software reliability were estimated as follows: $\hat{M}(26) \approx 1.343 \approx 2$ and $\hat{R}(1.0|26.0) \approx 0.7724$. Finally, we estimated the parameters of the ISS-CP model for DS1 as follows: $\hat{\omega} = 40.227$, $\hat{b} = 0.2917$, $\hat{l} = 0.0230$, and $\hat{\alpha} = 0.6318$. The expected number of remaining faults and software reliability were estimated as follows: $\hat{M}(26) \approx 0.227 \approx 1$ and $\hat{R}(1.0|26.0) \approx 0.9138$. Table 2.1 shows the results of parameter estimation and Kolmogorov-Smirnov statistics of change-point models, respectively. Furthermore, Table 2.2 shows the results of the goodness-of-fit comparison of change-point models with the corresponding existing models based on MSE. From these tables, we can say that the change-point models have better performance on software reliability assessment than the existing models. That is, we could confirm the effectiveness of consideration of the change-point for the EXP-CP model and the DSS-CP model. On the other hand, we cannot say that the ISS-CP model has better performance because the values of MSE for DS2 and DS3 are big. The estimated inflection coefficient might affect the accuracy for the goodness-of-fit because it was not estimated with the other parameters simultaneously.

2.9.2 Change-point detection method

We show numerical examples of our models (EXP2-CP, DSS2-CP models in Eqs. (2.16) and (2.17)).

Fig. 2.11 shows the behavior of the Laplace factor for DS1 as an example. From Fig. 2.11, we can see that the Laplace factor increases after this testing-time ($\tau = 4, 7, 9, 16$). Next, we estimated $\hat{\omega}$, $\hat{b}$, $\hat{\theta}$, and $\hat{k}$ which were the parameters of our EXP2-CP and DSS2-CP models, based on the maximum likelihood method. Table 2.3 shows the result of parameter estimation for the cases ($\tau = 4, 7, 9, 16$), respectively.

Figs. 2.12-2.15 show the estimated mean value functions with the effect of the change-point in Eqs. (2.16), (2.17), and its 95% confidence limits for DS1 in which ($\tau = 4, 7, 9, 16$). From these figures, we can see that the time-dependent behavior of estimated number of detected faults changes at change-point, and fits well to the actual behavior.

Next, we show numerical examples for software reliability assessment measures based on our EXP2-CP and DSS2-CP models. Table 2.4 shows the result of estimated expected number of remaining faults for DS1 in which ($\tau = 4, 7, 9, 16$). Figs. 2.16-2.19 show the estimated expected number of remaining faults, respectively. Table 2.5 shows the estimated software reliability for DS1 in which ($\tau = 4, 7, 9, 16$). Figs. 2.20-2.23 show the software reliability functions, respectively.

Furthermore, Table 2.6 shows the result of goodness-of-fit comparison of our models with the corresponding existing models based on MSE. From Table 2.6, we can say that our models with the detected change-points have better fitting performance than the existing models. Especially, the delayed S-shaped SRGMs with the detected change-points fit well because the time-dependent behavior for the actual data is S-shaped.

2.10 Conclusion

We discussed the change-point modeling frameworks for improving software reliability assessment. Concretely, we proposed the new NHPP models with change-point by the assumptions for the exponential, the delayed S-shaped, and the inflection S-shaped SRGMs. It has already been discussed that the exponential SRGM and the mean value function for NHPP are equivalent when the software failure-occurrence time distribution, $F(t)$, follows the exponential distribution. However, we could expand the NHPP models by assuming the gamma distribution and the logistic distribution.

Also, we considered the case which the testing-environmental factor follows the gamma distribution. We checked the performance of the existing models and our proposed models, and compared the goodness-of-fit by using MSE. From these results, our approach enables us

to describe the effect of the change-points on the software reliability growth process.

Furthermore, we discussed a change-point detection method based on the Laplace trend test, and checked the effectiveness by using MSE. From numerical examples, we can say that our proposed method enables us to detect the optimal occurrence-time of the change-points because our models with the detected change-points have better performance. Therefore, our proposed method is useful for software development managers to determine a change-point theoretically when they need to determine it by design.

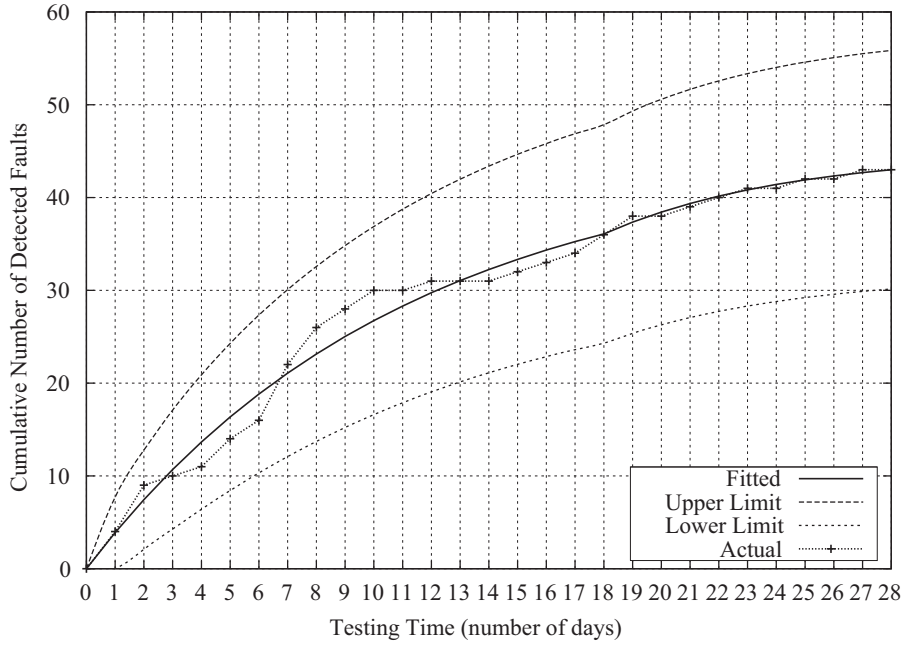

Fig. 2.2 : The estimated mean value function with change-point and its 95% confidence limits. (EXP-CP, DS5)

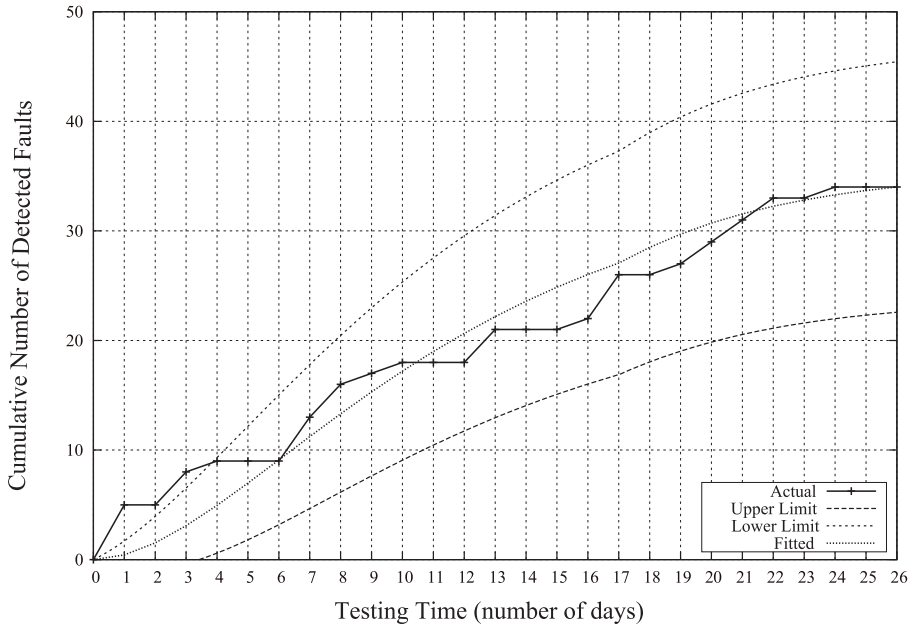

Fig. 2.3 : The estimated mean value function with change-point and its 95 % confidence limits. (DSS-CP, DS3)

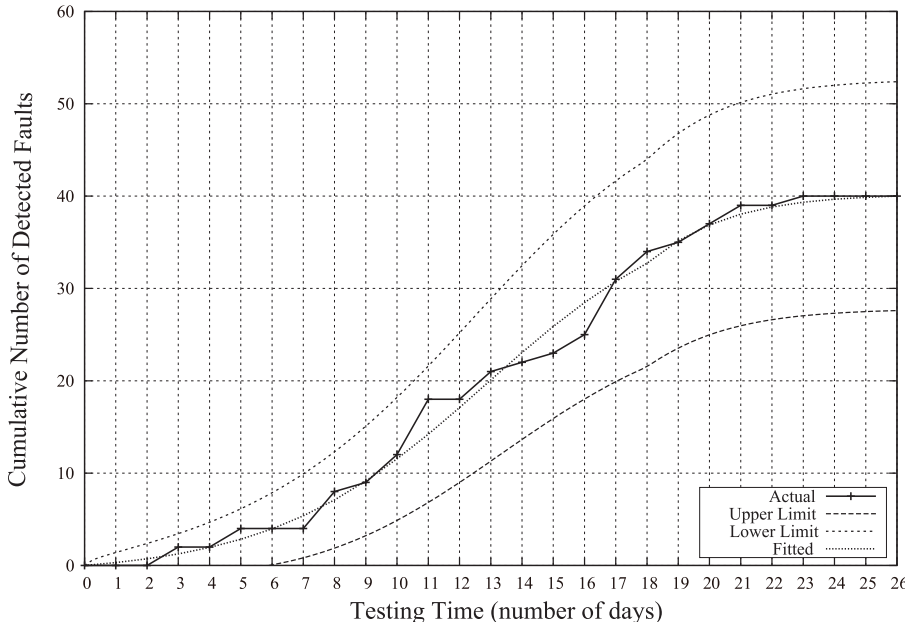

Fig. 2.4 : The estimated mean value function with change-point and its 95% confidence limits. (ISS-CP, DS1)

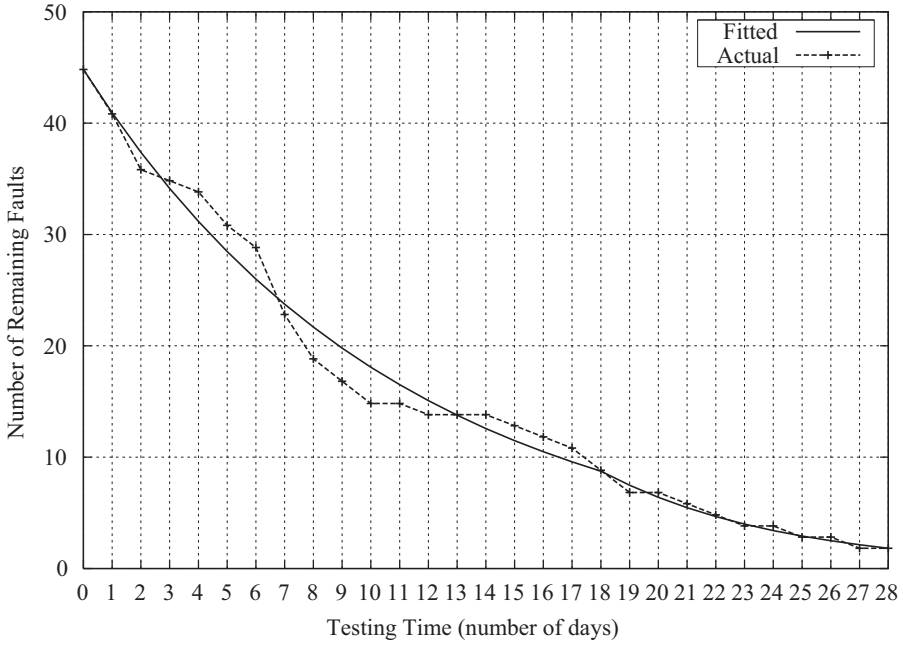

Fig. 2.5 : The expected number of remaining faults, $\widehat{M}(t)$. (EXP-CP, DS5)

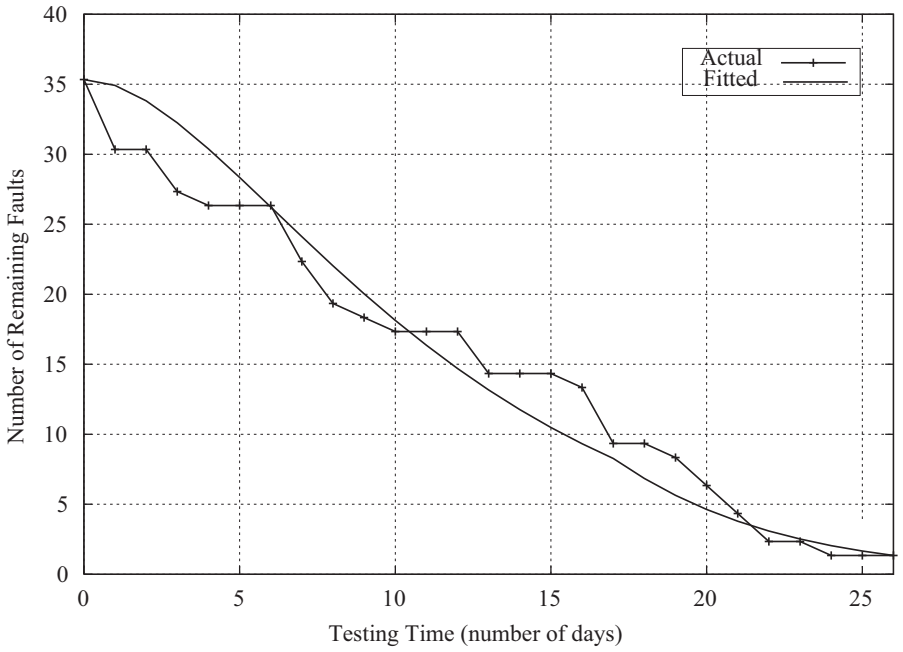

Fig. 2.6 : The expected number of remaining faults, $\widehat{M}(t)$. (DSS-CP, DS3)

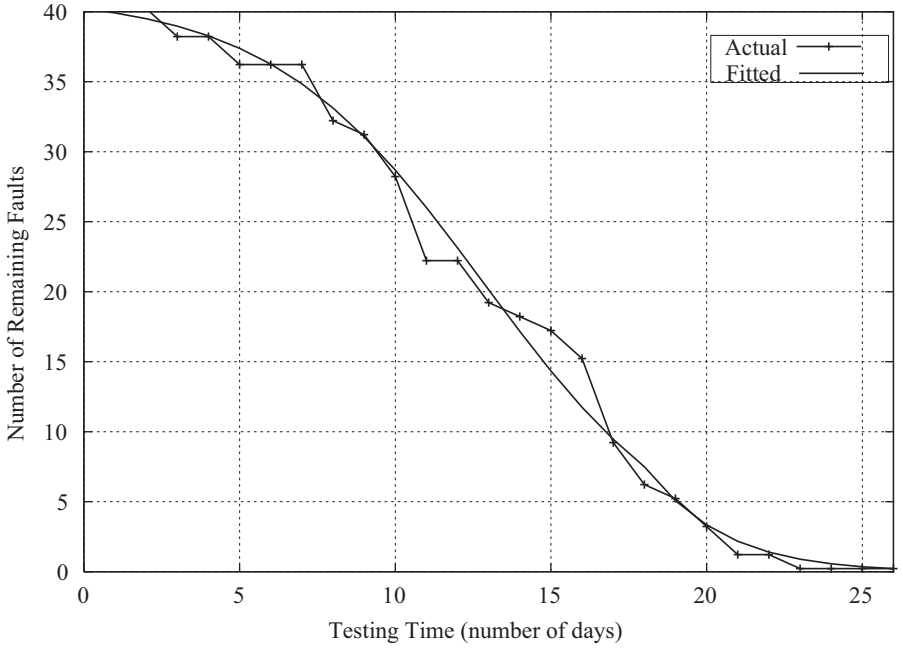

Fig. 2.7 : The expected number of remaining faults, $\widehat{M}(t)$. (ISS-CP, DS1)

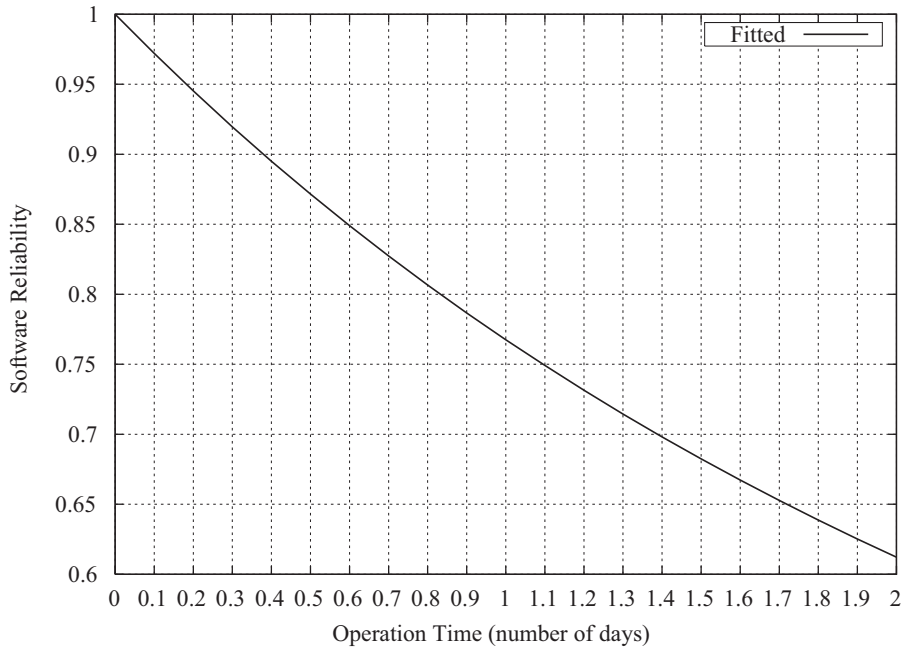

Fig. 2.8 : The software reliability function, $\widehat{R}(x \mid 28)$. (EXP-CP, DS5)

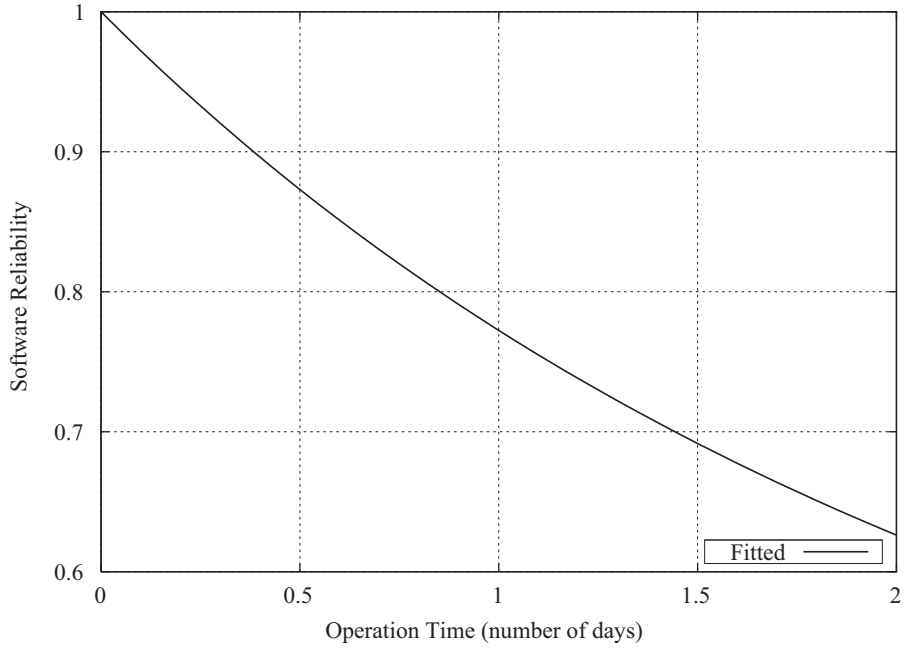

Fig. 2.9 : The software reliability function, $\widehat{R}(x \mid 26)$. (DSS-CP, DS3)

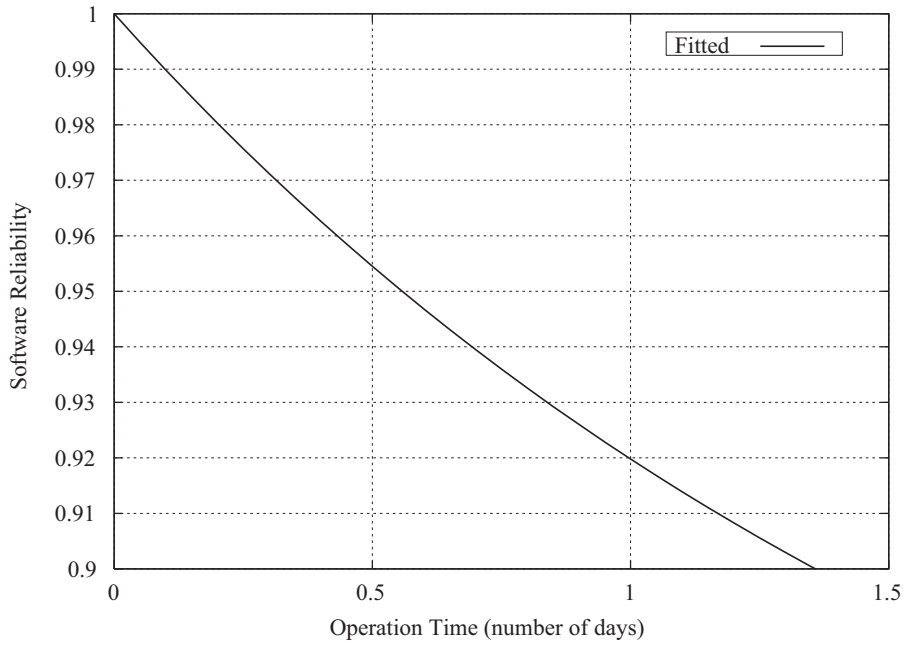

Fig. 2.10 : The software reliability function, $\widehat{R}(x \mid 26)$. (ISS-CP, DS1)

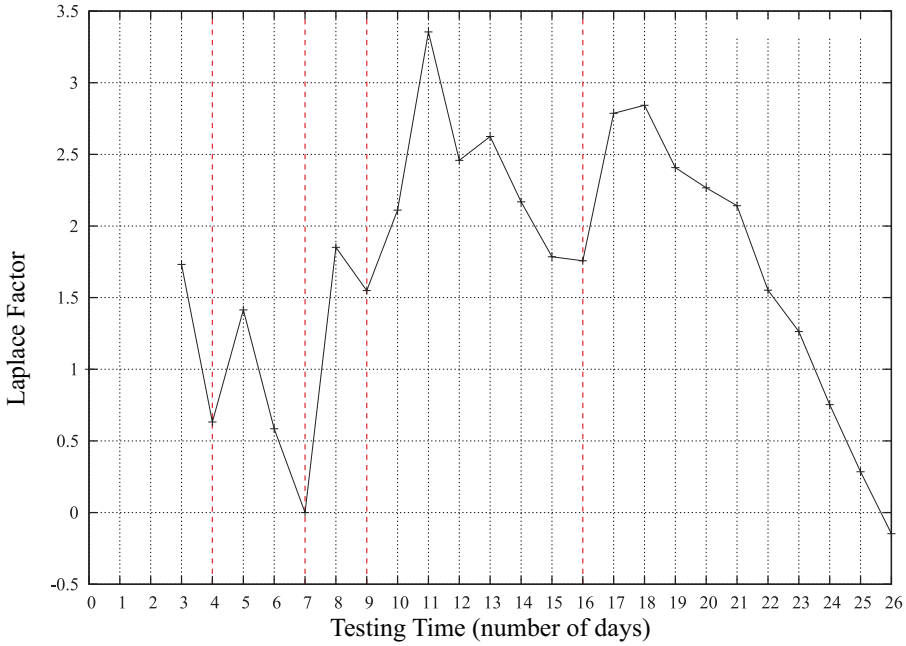

Fig. 2.11 : The temporal behavior of Laplace factor. (DS1)

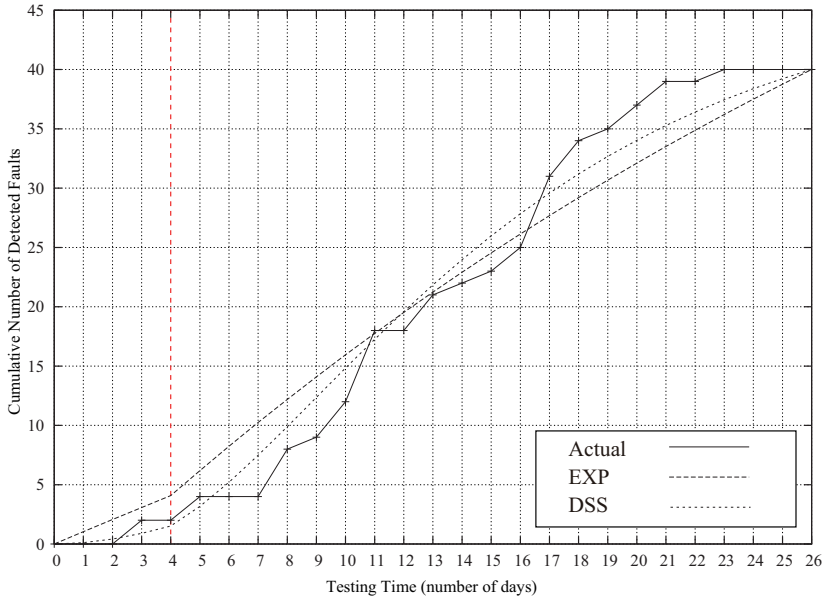

Fig. 2.12 : The estimated mean value function with change-point. ($\tau = 4$, EXP2-CP, DSS-CP, DS1)

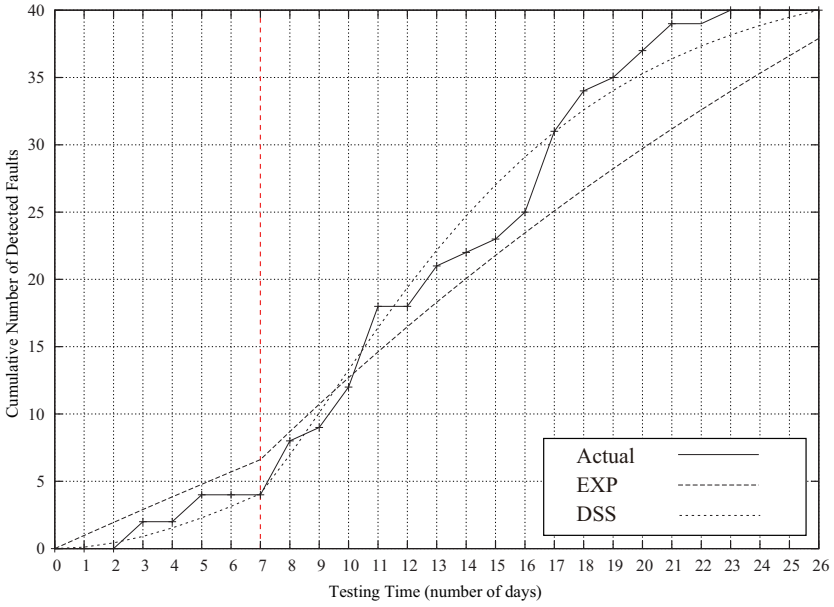

Fig. 2.13 : The estimated mean value function with change-point. ($\tau = 7$, EXP2-CP, DSS-CP, DS1)

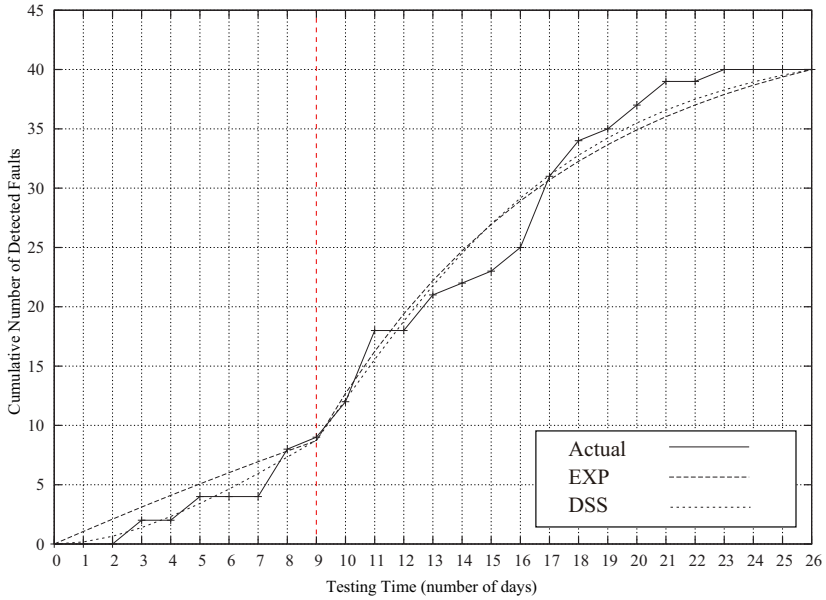

Fig. 2.14 : The estimated mean value function with change-point. ($\tau = 9$, EXP2-CP, DSS-CP, DS1)

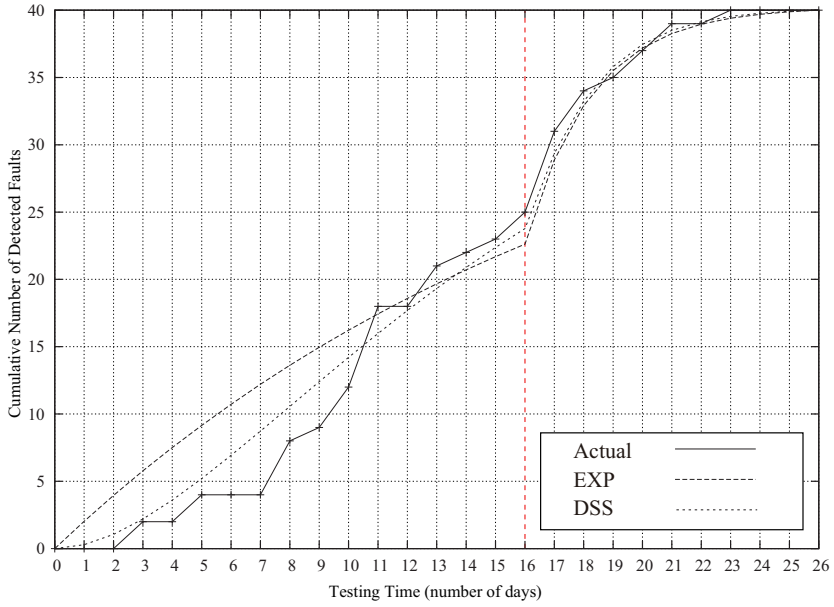

Fig. 2.15 : The estimated mean value function with change-point. ($\tau = 16$, EXP2-CP, DSS-CP, DS1)

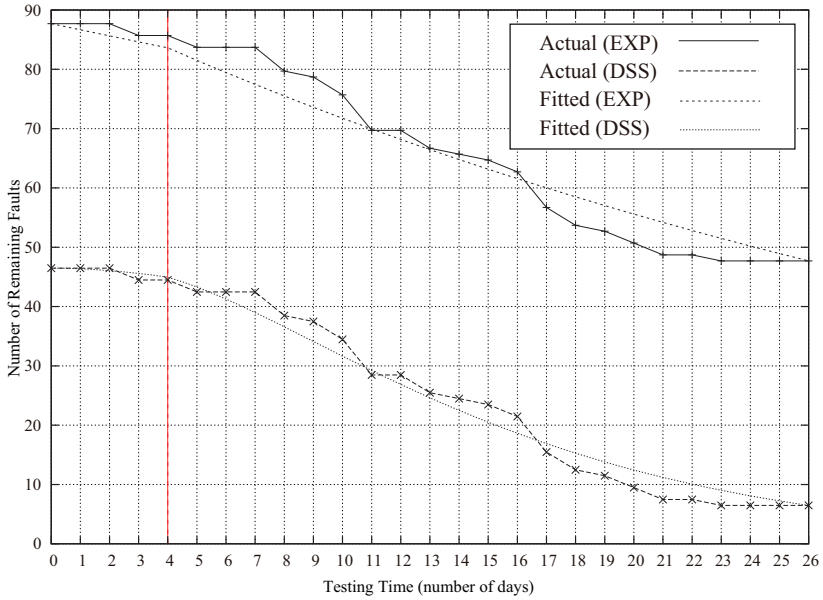

Fig. 2.16 : The expected number of remaining faults. ($\tau = 4$, EXP2-CP, DSS-CP, DS1)

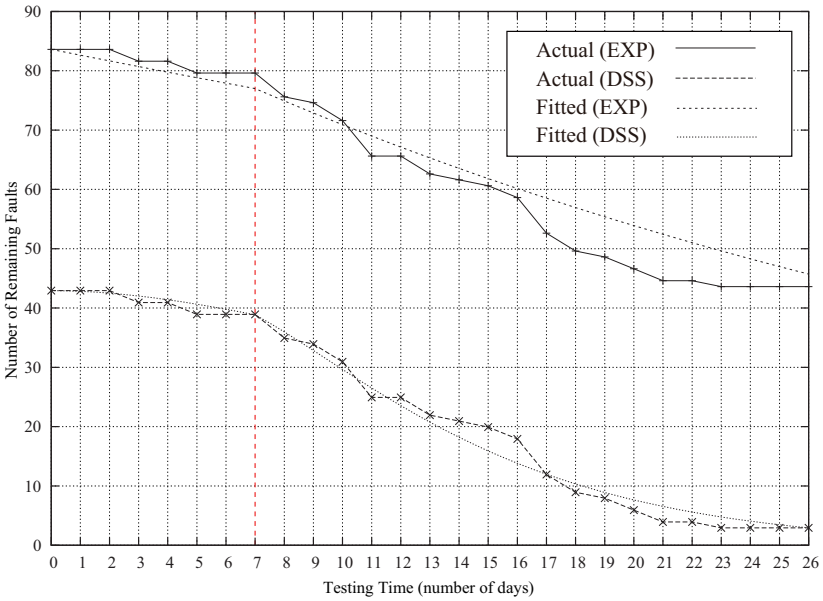

Fig. 2.17 : The expected number of remaining faults. ($\tau = 7$, EXP2-CP, DSS-CP, DS1)

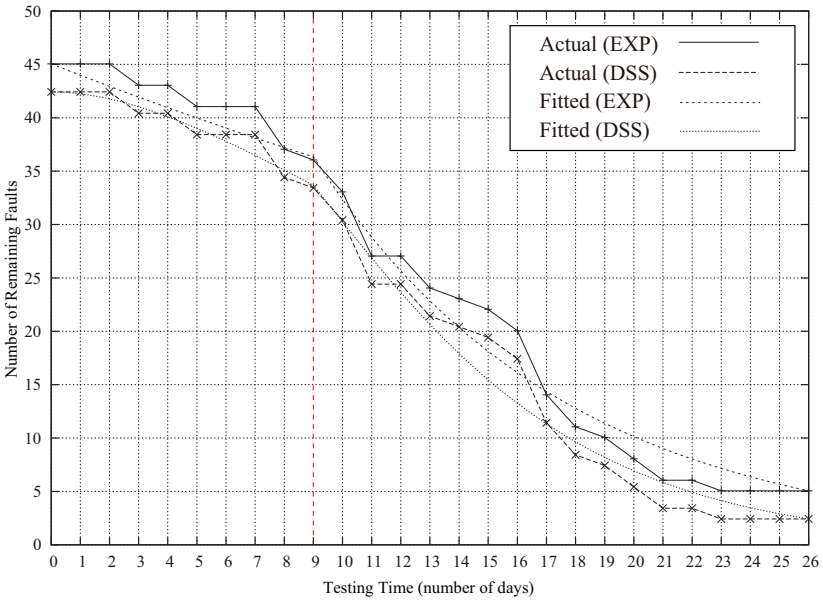

Fig. 2.18 : The expected number of remaining faults. ($\tau = 9$, EXP2-CP, DSS-CP, DS1)

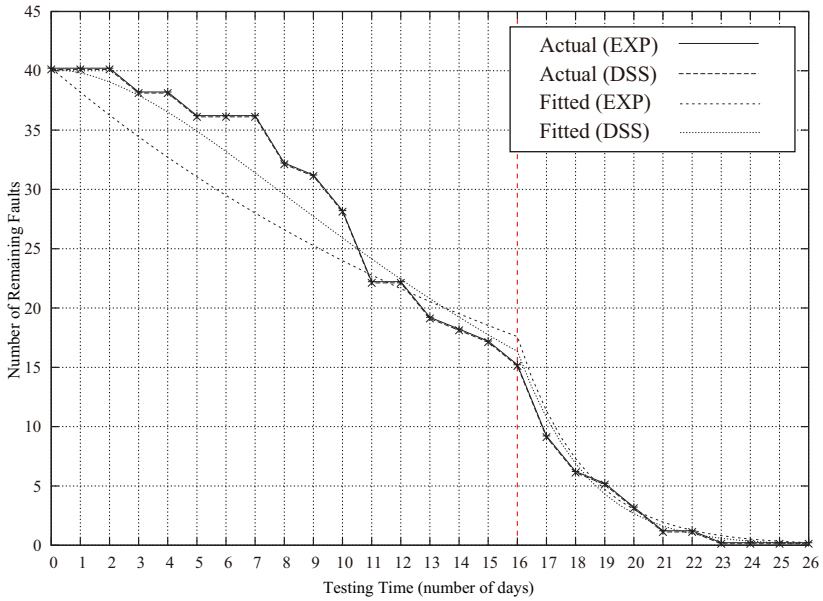

Fig. 2.19 : The expected number of remaining faults. ($\tau = 16$, EXP2-CP, DSS-CP, DS1)

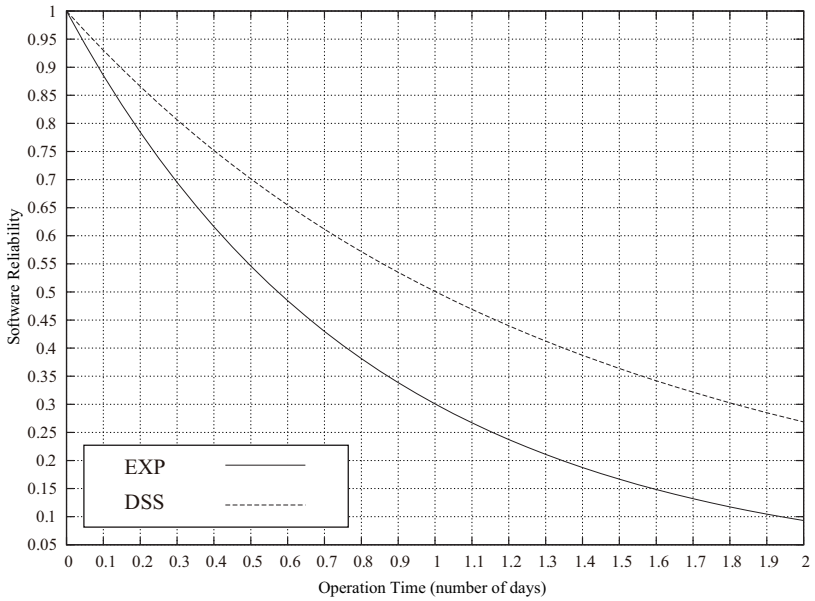

Fig. 2.20 : The software reliability function. ($\tau = 4$, EXP2-CP, DSS-CP, DS1)

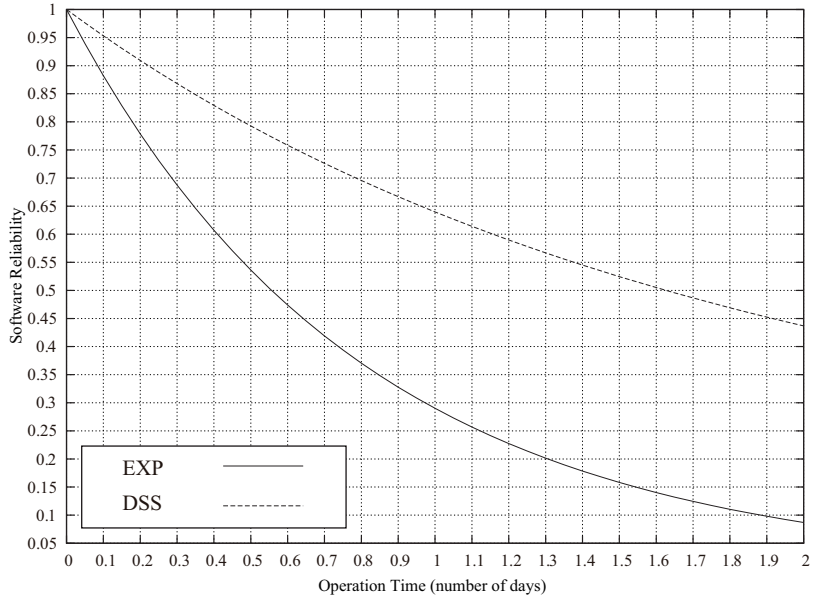

Fig. 2.21 : The software reliability function. ($\tau = 7$, EXP2-CP, DSS-CP, DS1)

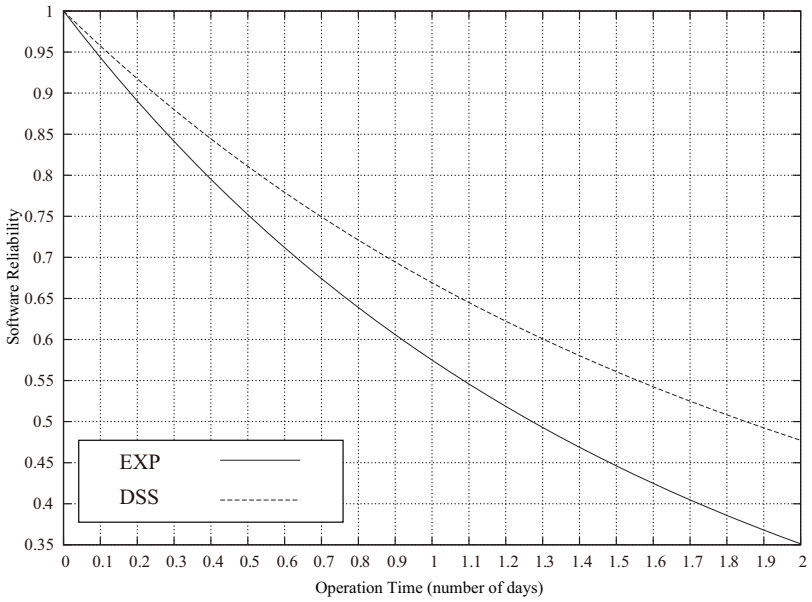

Fig. 2.22 : The software reliability function. ($\tau = 9$, EXP2-CP, DSS-CP, DS1)

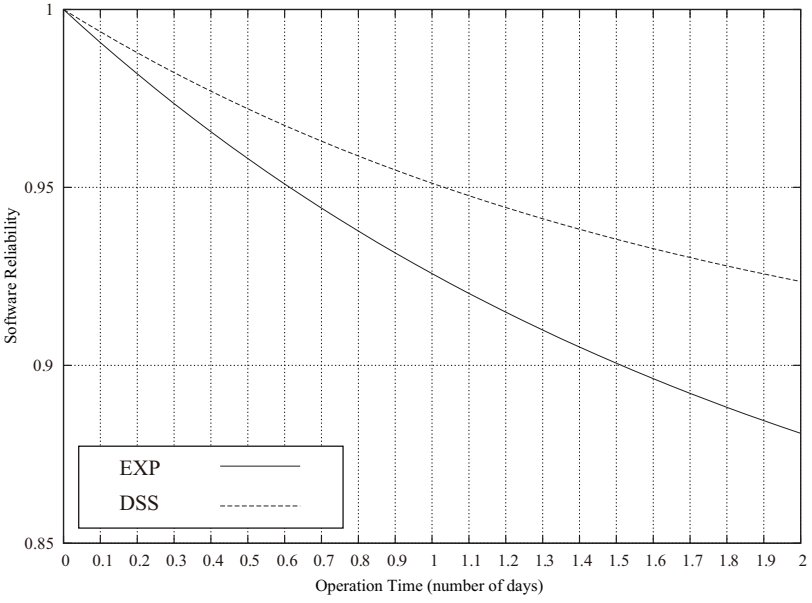

Fig. 2.23 : The software reliability function. ($\tau = 16$, EXP2-CP, DSS-CP, DS1)

Table 2.1 : The results of parameter estimation. (EXP, EXP-CP, DSS, DSS-CP, ISS, ISS-CP)

		τ	$\hat{\alpha}$	$\hat{\omega}$	$\hat{b}$	$\hat{l}$	$D_{\max}$
DS5	Exponential	-	-	47.8524	0.08174	-	0.117522*
	EXP-CP model	18	0.5789	44.8226	0.0908	-	0.124526*
DS3	Delayed S-shaped	-	-	37.0906	0.1584	-	0.171058*
	DSS-CP model	17	0.6425	35.343	0.1637	-	0.14907*
DS1	Inflection S-shaped	-	-	40.9138	0.2905	0.0230	0.334636*
	ISS-CP model	18	0.6318	40.2269	0.2917	0.0230	0.144249*

Table 2.2 : The comparison of goodness-of-fit based on MSE. (EXP, EXP-CP, DSS, DSS-CP, ISS, ISS-CP)

	EXP	EXP-CP	DSS	DSS-CP	ISS	ISS-CP
DS1	**16.0848**	20.5671	**7.2195**	7.77583	2.0476	**1.77933**
DS2	32.0834	**24.6323**	**19.3189**	22.1304	**24.3531**	31.7051
DS3	2.40089	**2.19263**	6.87789	**6.05874**	**2.39061**	25.8046
DS4	0.348901	**0.331126**	1.70385	**1.60806**	0.365351	**0.37564**
DS5	2.47239	**2.2943**	7.63319	**7.49945**	2.44597	**2.32179**

EXP : Exponential SRGM
EXP-CP : Exponential SRGM with change-point（our model）
DSS : Delayed S-shaped SRGM
DSS-CP : Delayed S-shaped SRGM with change-point（our model）
ISS : Inflection S-shaped SRGM
ISS-CP : Inflection S-shaped SRGM with change-point（our model）

Table 2.3 : The result of parameter estimation. (EXP2-CP, DSS2-CP, DS1)

	Parameters	$\tau = 4$	$\tau = 7$	$\tau = 9$	$\tau = 16$
EXP2-CP Model	$\hat{\omega}$	87.7045	59.9291	45.0518	40.2169
	$\hat{b}$	0.0119068	0.0214477	0.023876	0.0516676
	$\hat{\theta}$	704301	3.0807×10^6	323431	1.60138×10^6
	$\hat{k}$	328674	1.3203×10^6	1.6881×10^6	188209
DSS2-CP Model	$\hat{\omega}$	46.4727	42.9295	42.4234	40.1206
	$\hat{b}$	0.0700283	0.0736624	0.0935004	0.124673
	$\hat{\theta}$	-2.03186×10^6	718800	1.7653×10^6	5.92894×10^6
	$\hat{k}$	-981860	261380	750143	1.22985×10^6

Table 2.4 : The expected number of remaining faults. (EXP2-CP, DSS2-CP, DS1)

	$\tau = 4$	$\tau = 7$	$\tau = 9$	$\tau = 16$
EXP2-CP Model	47.7045	19.9291	5.05178	0.21685
DSS2-CP Model	6.47266	2.92955	2.42344	0.120586

Table 2.5 : The software reliability. (EXP2-CP, DSS2-CP, DS1)

	$\tau = 4$	$\tau = 7$	$\tau = 9$	$\tau = 16$
EXP2-CP Model	0.300664	0.378023	0.574871	0.925763
DSS2-CP Model	0.500777	0.639633	0.668987	0.95117

Table 2.6 : The comparison of goodness-of-fit based on MSE.

DS1			DS2			DS3		
τ	EXP	DSS2	τ	EXP	DSS	τ	EXP	DSS
4	10.9472	4.52732	5	18.0426	19.3189	6	2.32528	4.56313
7	7.88326	2.91847	6	22.9249	19.3773	12	2.274	6.97557
9	3.80684	2.6736	9	29.2436	17.1374	15	1.66543	6.5006
16	11.7953	2.84482	18	28.8811	16.4519	19	2.3583	5.16962
τ	Existing Model		τ	Existing Model		τ	Existing Model	
18	14.7191		24	26.4935		17	2.19263	

DS4			DS5		
τ	EXP	DSS	τ	EXP	DSS
4	0.430343	0.616764	4	2.47239	3.6921
16	0.63812	1.72474	6	2.74672	3.58157
18	0.351309	1.63757	12	5.75904	6.28737
20	0.258257	1.31583	17	2.47884	7.70729
τ	Existing Model		τ	Existing Model	
18	0.351309		18	4.80601	

Chapter 3

Optimal Software Release Problem Based on a Change-Point Model

3.1 Introduction

In a testing-phase, software faults which are introduced in the development process, are detected and removed by devoting enormous testing-resources. After that, the software product is released to the user. Normally, the more software is tested, many faults are detected. However, it is difficult to detect all faults because there are some problems in terms of economics. Therefore, what software development managers need to know is whether the software can be shipped without the further testing or not.

If any faults are not detected and removed in the testing-phase, the remaining faults are detected in the operational phase. Then, the maintenance cost, which is the cost for the modification and removal in the operational phase, is higher than the testing cost. It is said that the relative ratio of debugging cost for each development phase of specification, testing, and operation in software life cycle is 1:30:140, as shown in Fig. 3.1 [35]. Therefore, the software development managers or company would reduce the maintenance cost. However, if the testing-time is longer to cut the maintenance cost, the total software cost is expected to be more increasing although the reliability is expected to be more increasing. On the other hand, if the testing-time is too short, the reliability is expected to be more decreasing although the total software cost is expected to be more decreasing. That is, there is a trade-off relationship between the testing cost and maintenance cost [44].

From the background, it is important for software development managers to determine when to stop the testing or release to the user. This problem is called an *"optimal software release problem"*. Especially, we consider a cost-optimal software release problem based on our

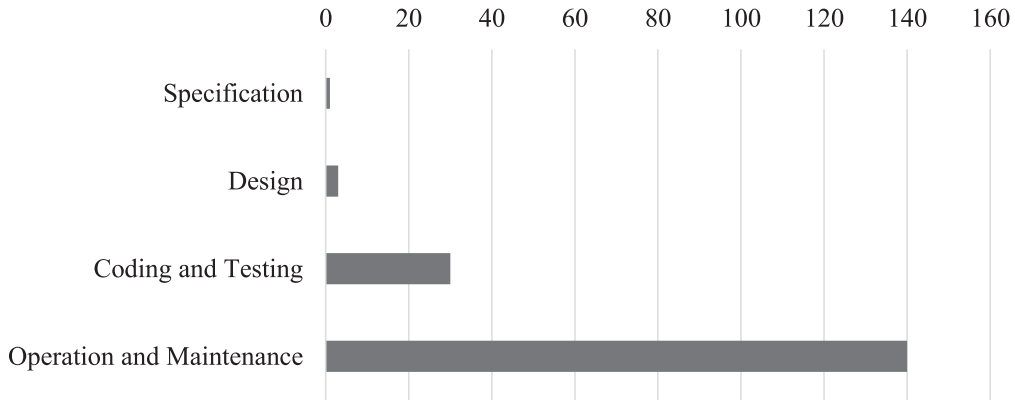

Fig. 3.1 : The relative ratio of debugging cost for each development phase [35].

change-point model by an analytical approach. First, we formulate a cost function based on our proposed EXP-CP model of Eq. (2.9) in Chapter 2. Then, we consider the relationship of the optimal software release time and the optimal occurrence-time of the change-point. Therefore, we can obtain the optimal software release time and the optimal occurrence-time of change-point simultaneously by minimizing the cost function. Finally, we derive the expected total software cost and optimal software release policy. Also, we evaluate the optimal software release time and expected total software cost in terms of the reliability evaluation criterion.

This chapter consists of the following sections. In Section 3.2, we discuss an optimal software release problem based on a change-point model by the analytical approach. In Section 3.3, we show numerical examples. In the final section, we conclude with summary.

3.2 Optimal Software Release Problem by an Analytical Approach

3.2.1 Optimal software release time and change-point

We use the change-point model in which the mean value function before change-point follows the exponential SRGM in Eq. (2.9).

First, we define the following cost parameters:

c_1 : debugging cost for one fault before change-point in the testing phase ($0 < c_1$),
c_2 : debugging cost for one fault after change-point in the testing phase ($0 < c_2$),
c_3 : debugging cost for one fault after change-point in the operational phase ($c_1 < c_3$, $c_2 < c_3$),
c_4 : testing cost at arbitrary testing-time ($0 < c_4$).

Let T be the termination time of testing, and s be the testing-time duration from the change-point to the termination time of testing. Also, T^* is the optimal release time, and s^* is the optimal testing-time duration from the change-point to the termination time of testing. Then, we assume $0 < s^* < T^*$. That is, the optimal occurrence-time of the change-point is derived by $T^* - s^*$.

By using these parameters, we derive the following equation which represents the expected total software cost during the testing and operational phases:

$$C(T,s) = c_1\Lambda_1(T-s) + c_2\{\Lambda_2(T) - \Lambda_1(T-s)\} + c_3\{\omega - \Lambda_2(T)\} + c_4 T, \tag{3.1}$$

where $\Lambda_1(t)$ and $\Lambda_2(t)$ represent the mean value functions before the change-point and those after change-point, respectively.

We can derive the optimal software release time and the optimal occurrence-time of the change-point by solving the following equation:

$$\frac{\partial C(T,s)}{\partial T} = \frac{\partial C(T,s)}{\partial s} = 0. \tag{3.2}$$

From Eq. (3.2), we have the optimal release time, T^*, as

$$T^* \equiv T(s) = \frac{1}{b}\log\left[\frac{\omega b(c_3-c_2)}{\alpha c_4 \exp[b(\frac{s(1-\alpha)}{\alpha})]}\right]. \tag{3.3}$$

Then, the sufficient condition for $T^* > 0$ is given as

$$s < \frac{\log[\frac{\omega b(c_3-c_2)}{\alpha c_4}]}{b(\frac{1}{\alpha}-1)} (\equiv A). \tag{3.4}$$

It is noted that Eq. (3.4) is not a constraint condition for optimization problems. Next, substituting Eq. (3.3) into Eq. (3.2), we can derive the following equation:

$$Z(s) \equiv -\alpha c_4\left[\frac{c_2-c_1}{c_3-c_2}e^{\frac{1}{\alpha}bs} + 1 - \frac{1}{\alpha}\right]. \tag{3.5}$$

Finally, we can derive the optimal occurrence-time of the change-point as

$$s^* = \frac{\alpha}{b}\log\left[\frac{(c_3-c_2)(1-\alpha)}{\alpha(c_2-c_1)}\right]. \tag{3.6}$$

3.2.2 Optimal software release policy

We obtain the following cost-optimal software release policy.

【Optimal Software Release Policy】

< I >

When $c_1 < c_2$ and $\alpha < 1$, $Z(s)$ is a monotonically decreasing function with respect to s.

1. If $Z(0) > 0$ and $Z(A) < 0$, unique solutions exist as T^* and s^*. They are given by Eq. (3.3) and Eq. (3.6), respectively.

2. If $Z(0) \leq 0$ and $Z(A) < 0$, we continue the testing in the environment before change-point. Also, T^* is given by Eq. (3.3).

< II >

When $c_2 < c_1$ and $\alpha > 1$, $Z(s)$ is a monotonically increasing function with respect to s.

1. If $Z(0) < 0$ and $Z(A) > 0$, unique solutions exist as T^* and s^*. They are given by Eq. (3.3) and Eq. (3.6), respectively.

2. If $Z(0) \geq 0$ and $Z(A) > 0$, we continue the testing in the environment before change-point. And, T^* is given by Eq. (3.3).

Fig. 3.2 shows the behavior of $Z(s)$ in Eq. (3.5). When $c_1 < c_2$ and $\alpha < 1$, we can see that $Z(s)$ is a monotonically decreasing function with respect to s. When $c_2 < c_1$ and $\alpha > 1$, we can see that $Z(s)$ is a monotonically increasing function with respect to s. Then, whether the optimal occurrence-time of the change-point exists or not is determined by the relationship of c_1 and c_2.

3.3 Numerical Examples

We show the application of the optimal software release policy by using the actual data sets in Chapter 2. The cost parameters are assumed as follows: $c_1 = 1.0$, $c_2 = 2.0$, $c_3 = 150$, and $c_4 = 5.0$.

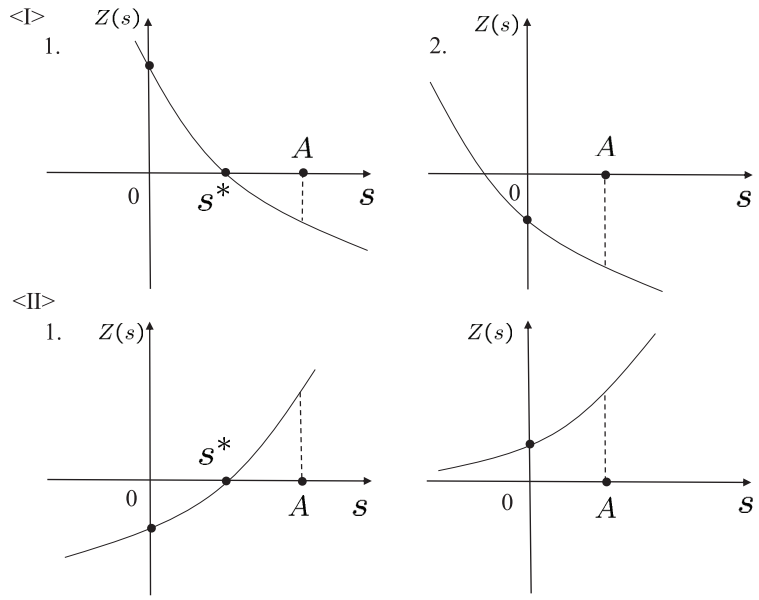

Fig. 3.2 : The behavior of $Z(s)$ based on the optimal software release policy.

< DS1 >

From $\hat{\omega} = 4688.87$, $\hat{b} = 0.00039$, $\hat{\alpha} = 1.81(> 1)$, and $c_1 < c_2$, the optimal software release policy is not concerned.

< DS2 >

From $\hat{\omega} = 349.141$, $\hat{b} = 0.00962$, $\hat{\alpha} = 13.29(> 1)$, and $c_1 < c_2$, the optimal software release policy is not concerned.

< DS3 >

From $\hat{\omega} = 44.482$, $\hat{b} = 0.0507$, $\hat{\alpha} = 0.781(< 1)$, and $c_1 < c_2$, we obtain

$$\begin{cases} s^* = 57.39208, \\ Z(A) = -17478907, \\ Z(0) = 1.068615. \end{cases} \tag{3.7}$$

When we apply the optimal software policy < I >, we have

$$T^* = 71.64259, \tag{3.8}$$

$$C(T^*, s^*) = 501.3143. \tag{3.9}$$

We show the behavior of the estimated expected total software cost in Fig. 3.3.

< DS4 >

From $\hat{\omega} = 81.0232$, $\hat{b} = 0.0114$, $\hat{\alpha} = 0.766(< 1)$, and $c_1 < c_2$, we obtain

$$\begin{cases} s^* = 256.0961, \\ Z(A) = -111654.9, \\ Z(0) = 1.144122. \end{cases} \tag{3.10}$$

When we apply the optimal software policy < I >, we have

$$T^* = 235.3583. \tag{3.11}$$

It is noted that we can not say that the optimal solutions are derived properly because s^* is bigger than T^*. We show the behavior of the estimated expected total software cost in Fig. 3.4.

< DS5 >

From $\hat{\omega} = 44.8226$, $\hat{b} = 0.0908$, $\hat{\alpha} = 0.5789(< 1)$, and $c_1 < c_2$, we obtain

$$\begin{cases} s^* = 29.83497, \\ Z(A) = -6253.591, \\ Z(0) = 2.086009. \end{cases} \tag{3.12}$$

When we apply the optimal software policy< I >, we have

$$T^* = 37.09248, \tag{3.13}$$

$$C(T^*, s^*) = 285.3592. \tag{3.14}$$

We show the behavior of the estimated expected total software cost in Fig. 3.5.

Additionally, we consider the reliability evaluation criterion. Concretely, we assume that the reliability objective is 0.8, and we set $x = 1.0$ in Eq. (2.23). From Figs. 3.6-3.8, the estimated cost-optimal software release time achieves the reliability objective 0.8. We can see that the optimal software release time, the optimal occurrence-time of the change-point, and the expected total software cost are satisfied the cost and reliability evaluation criteria.

3.4 Conclusion

We discussed an optimal software release problem based on a change-point model, and provided the optimal software release policy. The optimal software release time, optimal occurrence-time of the change-point, and expected total software cost were derived analytically. After that, we evaluated the optimal software release time with considering the reliability evaluation criterion. We could expand the existing optimal software release problem by our approach, and propose the new method for determining the optimal occurrence-time of the change-point.

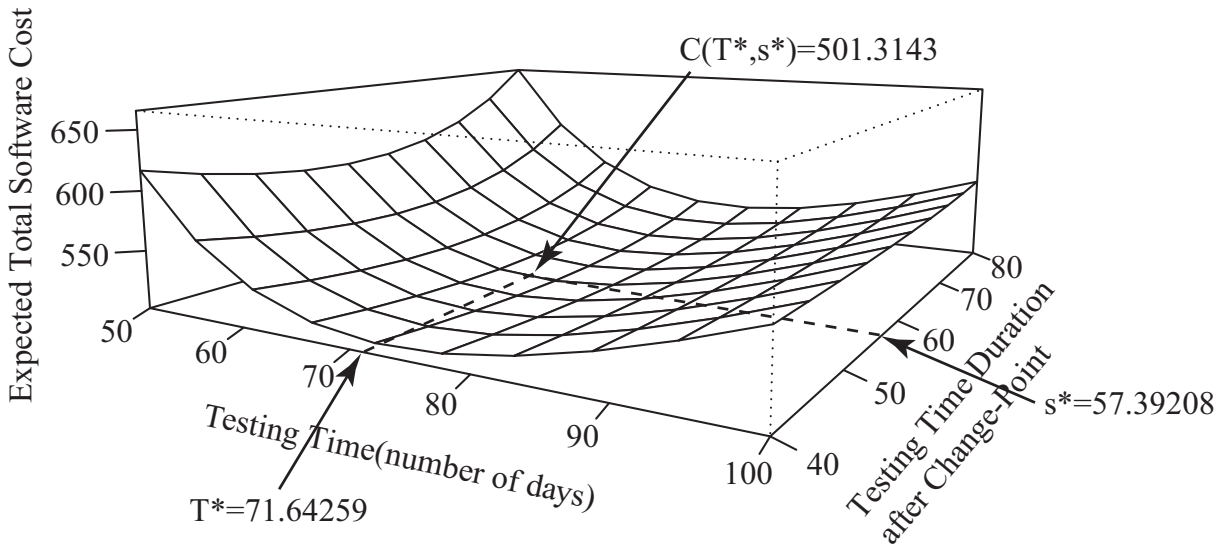

Fig. 3.3 : The expected total software cost for DS3.

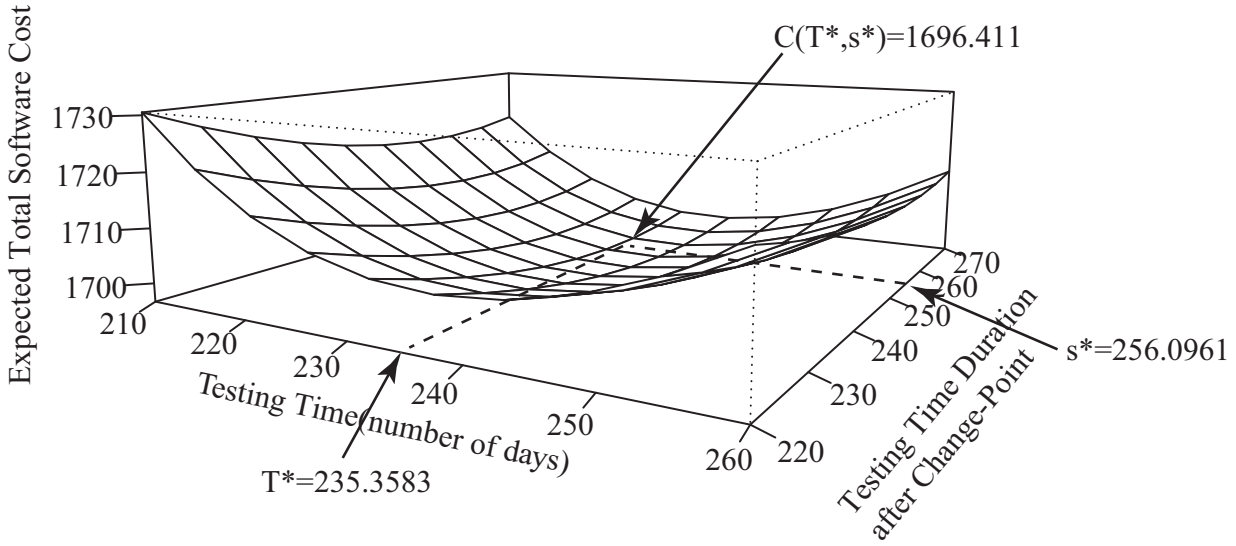

Fig. 3.4 : The expected total software cost for DS4.

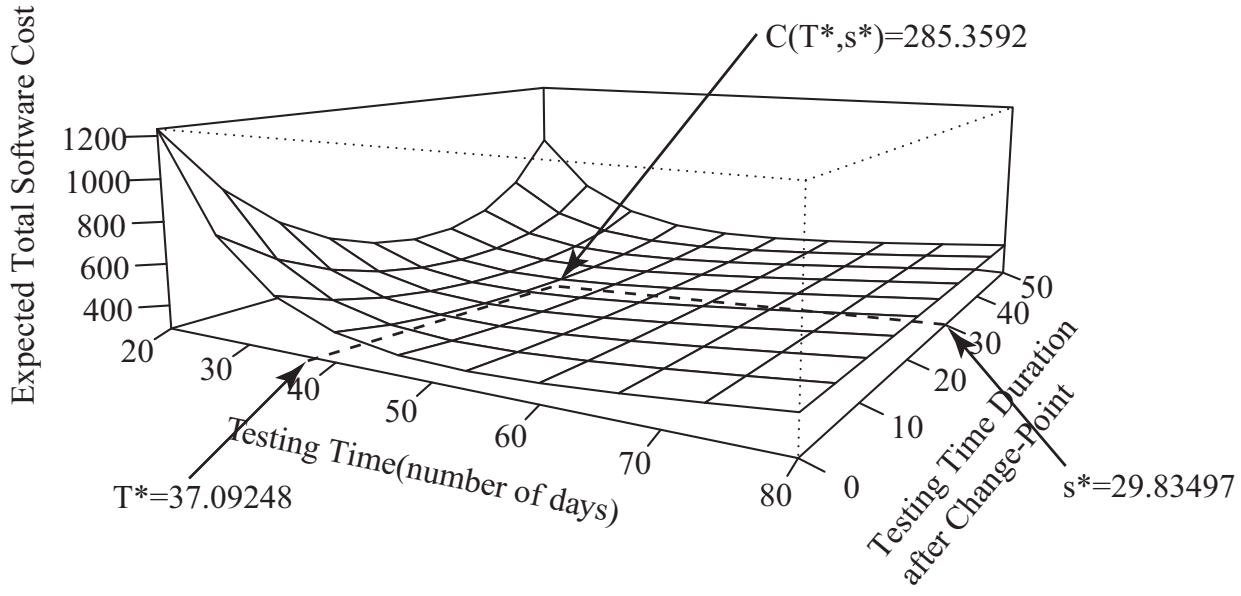

Fig. 3.5 : The expected total software cost for DS5.

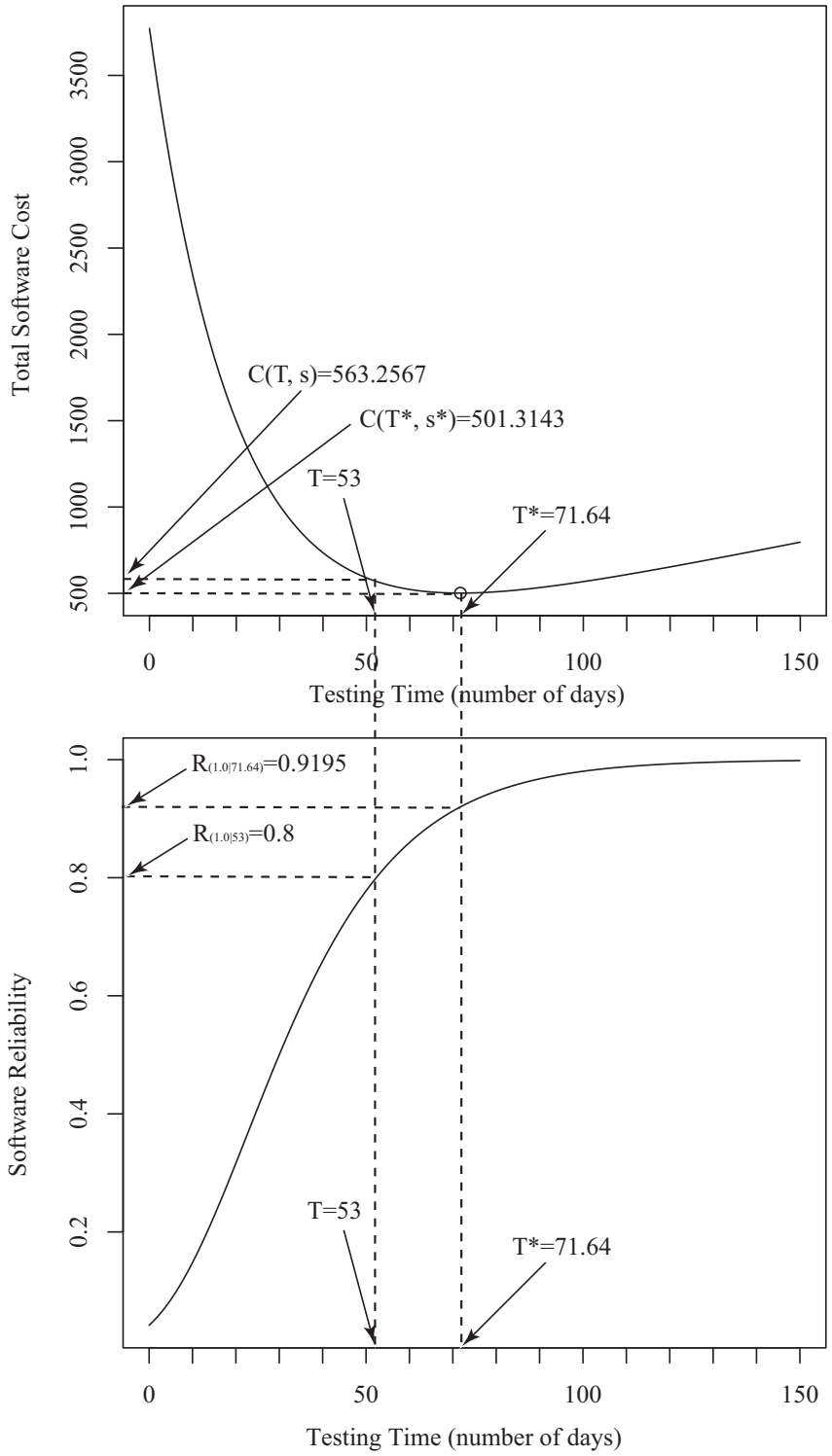

Fig. 3.6 : The optimal total testing period for DS3.

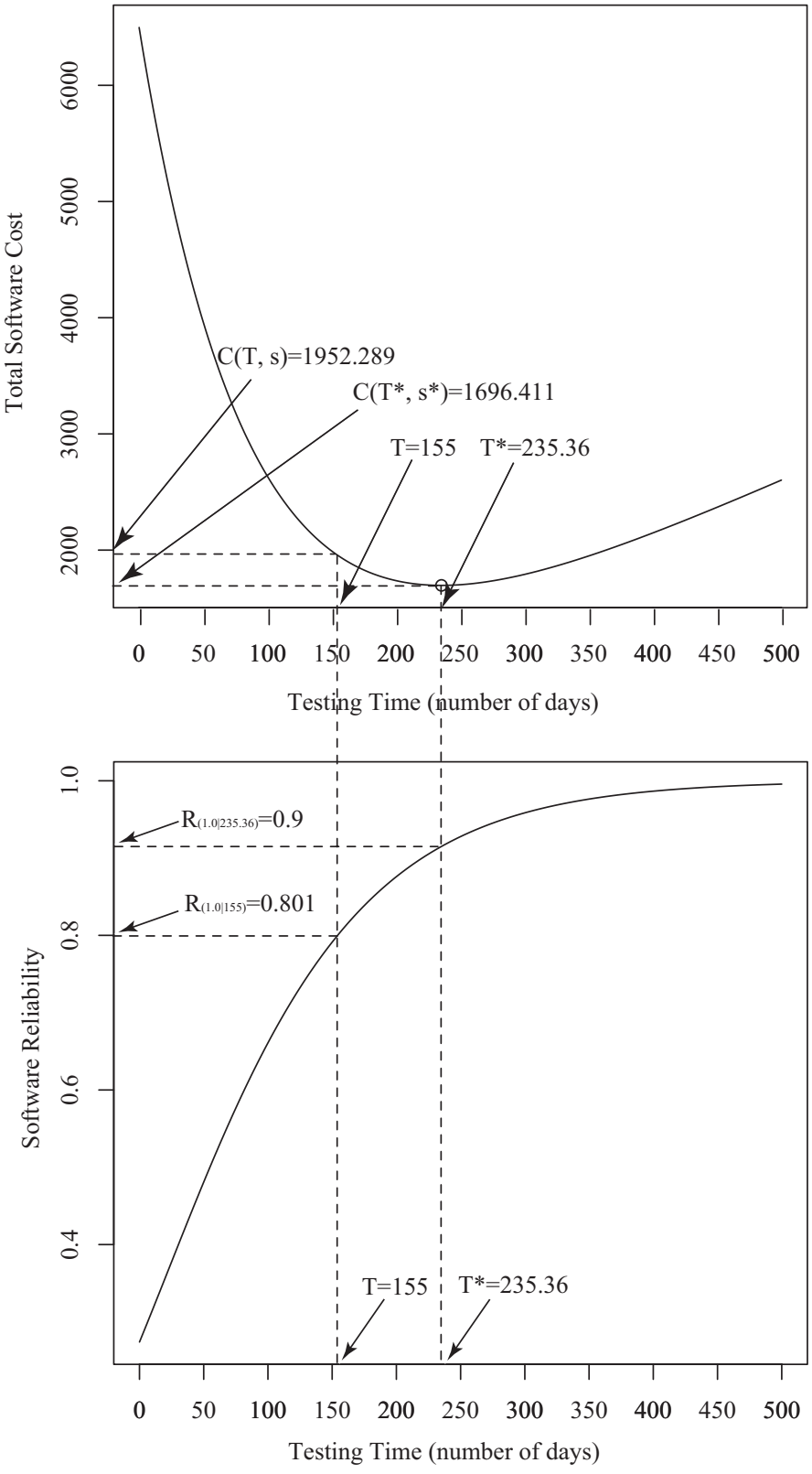

Fig. 3.7 : The optimal total testing period for DS4.

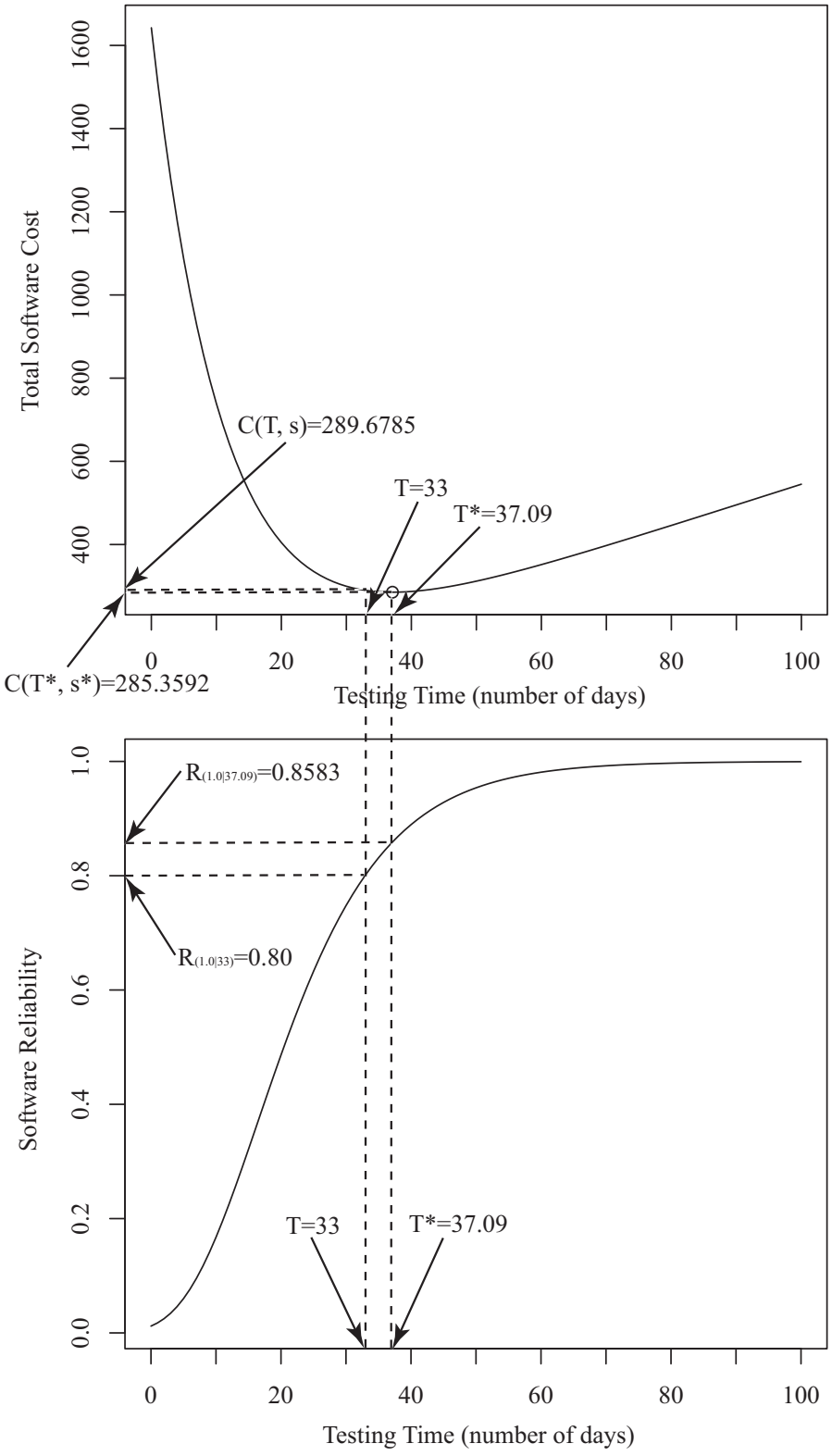

Fig. 3.8 : The optimal total testing period for DS5.

Chapter 4

Optimal Software Release Problem Based on MAUT

4.1 Introduction

In this chapter, we discuss an optimal software release problem based on the multi-attribute utility theory (MAUT). The optimal software release time is determined in consideration of the cost, reliability, and delivery. For example, we have determined the optimal solutions as the time minimizing the expected total software cost. However, the minimum cost is not important necessarily in actuality. From the perspective of management aspects, it is important that considering the various attributes comprehensively. Furthermore, the software development managers want to know the evaluation for their decision-making, quantitatively. Therefore, we apply the multi-attribute utility theory which is a method for the decision-making considering with the utility. It enables us to derive and evaluate the optimal solutions in terms of a utility theory.

The steps of our approach are as follows: First, we define a cost function by using the proposed EXP-CP and DSS-CP models of Eqs. (2.9) and (2.10) in Chapter 2. Next, we set cost and reliability attributes. After that, we develop single-attribute utility functions for each attribute. Finally, we develop a weighted multi-attribute utility function based on the single-attribute utility functions, and maximize it. Then, we can derive the optimal solutions, i.e., the optimal software release time and testing-time duration from the change-point to the termination time of testing, simultaneously. From these optimal solutions, the optimal occurrence-time of the change-point can be obtained. Also, we evaluate the optimal solutions, and check the behavior of the multi-attribute utility function and the expected total software cost.

This chapter consists of the following sections. In Section 4.2, we discuss an optimal software release problem for the case of two attributes. In Section 4.3, we show numerical examples. In the final section, we conclude with summary.

4.2 Optimal Software Release Problem with Two Evaluation Criteria

4.2.1 Selection of attributes

Since the software development managers want to spend less than its budget, the cost attribute is considered as

$$\text{min: } C = \frac{C(T,s)}{C_B}, \tag{4.1}$$

where C_B is the budget, and $C(T, s)$ is given by Eq. (3.1). Next, the reliability attribute is considered as

$$\text{max: } R = \frac{\Lambda(T)}{initial\ fault\ content}, \tag{4.2}$$

where $\Lambda(T)$ is the EXP-CP model and the DSS-CP model in Chapter 2. That is, the parameters, such as T and s, are assumed as with Chapter 2. It is noted that the reliability attribute is needed to be maximized because the software development managers want high reliability.

Furthermore, we can say that these attributes have a competitiveness because they are defined by using the same change-point models. In other words, these attributes depend on each other because the total software cost changes by the number of detected faults.

4.2.2 Development of single-attribute utility function

The single-attribute utility functions for each attribute are based on the following management strategy.

1. For the cost attribute, at least 50% of the budget must be consumed.

2. For the reliability attribute, at least 50% of software faults should be detected and the more better.

3. The management team takes the risk neutral position for each attribute.

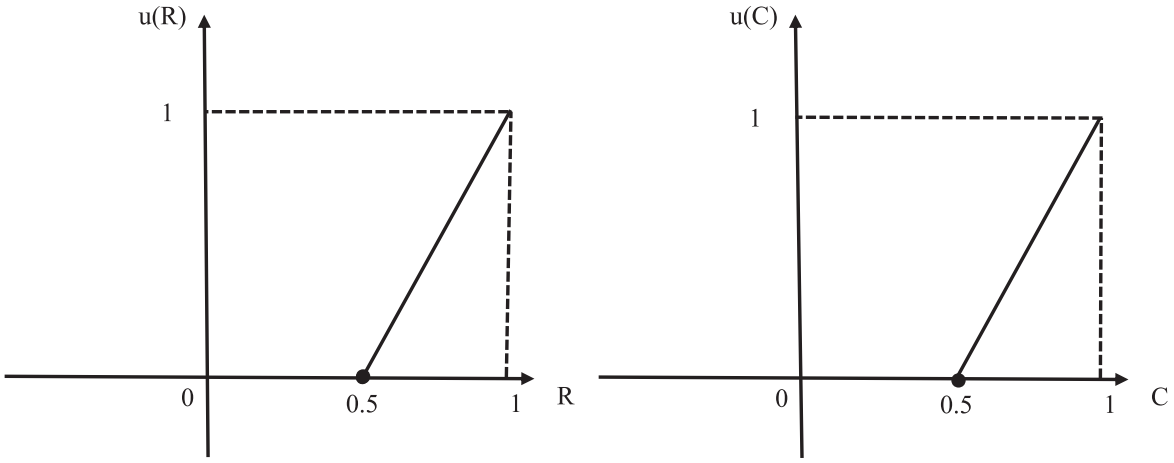

Fig. 4.1 : The single-attribute utility functions.

Then, the lowest and highest requirements for the reliability attribute are $R^L = 0.5$ and $R^H = 1.0$. The lowest and highest consumptions for the cost attribute are $C^L = 0.5$ and $C^H = 1.0$.

We use the additive linear form because the third management strategy is assumed. Therefore, we have the following equations.

$$u(R) = 2R - 1. \tag{4.3}$$

$$u(C) = 2C - 1. \tag{4.4}$$

4.2.3 Development of multi-attribute utility function

We apply the additive form of the multi-attribute utility function. The multi-attribute utility function with some constraints as an optimization problem is given as

$$\begin{aligned} \max : u(C,R) &= w_R \times u(R) - w_C \times u(C) \\ &= w_R \times (2R-1) - w_C \times (2C-1) \\ \text{subject to} \quad & w_R + w_C = 1, \end{aligned} \tag{4.5}$$

where w_R and w_C are the weight parameters for the attributes, R and C, respectively. $u(R)$ and $u(C)$ are the single-attribute utility functions for each attribute. We can obtain the optimal testing-time, T^*, and optimal testing-time duration from the change-point to the termination

time of testing, s^*, by solving the optimization problem of Eq. (4.5). Also, the optimal occurrence-time of the change-point is given as $\tau^* = T^* - s^*$. We calculated it by R which is a tool for the statistical analysis.

4.3 Numerical Examples

We use DS3 and DS5 of the actual data sets in Chapter 2. We use the actual data sets, DS3 and DS5 in Chapter 2. Also, we assume that the cost parameters in the cost function and the budget are $c_1 = 1.0, c_2 = 2.0, c_3 = 150.0, c_4 = 5.0$, $C_B = 1000$ for DS3, and $C_B = 450$ for DS5. The behavior of single-attribute utility functions is shown in Fig. 4.1.

Figs. 4.2, 4.4, and 4.6 show the relationship of expected total software cost and optimal software release time for each change-point model. Figs. 4.3, 4.5, and 4.7 show the relationship of utility and optimal software release time. From these figures, we can see that we could determine the optimal software release time maximizing the utility.

Tables 4.1, 4.2, and 4.3 show the results of sensitive analysis. Additionally, we show the results of optimal occurrence-time of the change-point and expected total software cost. In Tables 4.1 and 4.2, we can see that when the optimal software release time and optimal testing-time duration from the change-point to the termination time of testing increase, the expected total software cost and utility increase. In Table 4.3, we can see that the utility changes by the weight parameters mainly although the sensitivity is small. It is noted that these values are evaluated by comparative assessment.

4.4 Conclusion

We discussed an optimal software release problem based on the multi-attribute utility theory. Especially, we defined that the cost and reliability attributes based on the change-point models. Also, the weighted multi-attribute utility function with the constraints was formulated as an optimization problem by using those attributes. By solving the optimization problem, we estimated the optimal software release time, the testing-time duration from the change-point to the termination time of testing, optimal occurrence-time of the change-point, expected total software cost, and utility. Furthermore, we showed the results of sensitive analysis, and confirmed the behavior of the total software cost and utility. From the results of sensitive analysis, we could evaluate the balance of the attributes in terms of utility theory.

Table 4.1 : The result of sensitive analysis. (EXP, DS3, 2 attributes)

w_R	w_C	T^*	s^*	τ^*	Total Cost	Utility
0.1	0.9	71.89944	57.65013	14.24931	501.325	0.095314
0.3	0.7	72.61439	58.36695	14.24744	501.4646	0.291359
0.5	0.5	73.81941	59.57549	14.24392	502.0493	0.487794
0.7	0.3	76.31965	62.07379	14.24586	504.5321	0.685191
0.9	0.1	84.85926	70.6586	14.20066	523.0659	0.886464

Table 4.2 : The result of sensitive analysis. (EXP, DS5, 2 attributes)

w_R	w_C	T^*	s^*	τ^*	Total Cost	Utility
0.1	0.9	37.14034	29.87466	7.26568	285.3361	-0.142298
0.3	0.7	37.27178	30.01017	7.26161	285.348	0.1094483
0.5	0.5	37.50686	30.25121	7.25565	285.4025	0.3612723
0.7	0.3	38.02501	30.76971	7.2553	285.6632	0.6133076
0.9	0.1	40.1202	32.8696	7.2506	288.4347	0.8664326

Table 4.3 : The result of sensitive analysis. (DSS, DS5, 2 attributes)

w_R	w_C	T^*	s^*	τ^*	Total Cost	Utility
0.1	0.9	32.72308	17.33735	15.38573	235.2164	-0.898841
0.3	0.7	32.72308	17.33735	15.38573	235.2164	-0.60571
0.5	0.5	32.72308	17.33735	15.38573	235.2164	-0.312579
0.7	0.3	32.72308	17.33735	15.38573	235.2164	-0.019448
0.9	0.1	32.72308	17.33735	15.38573	235.2164	0.273683

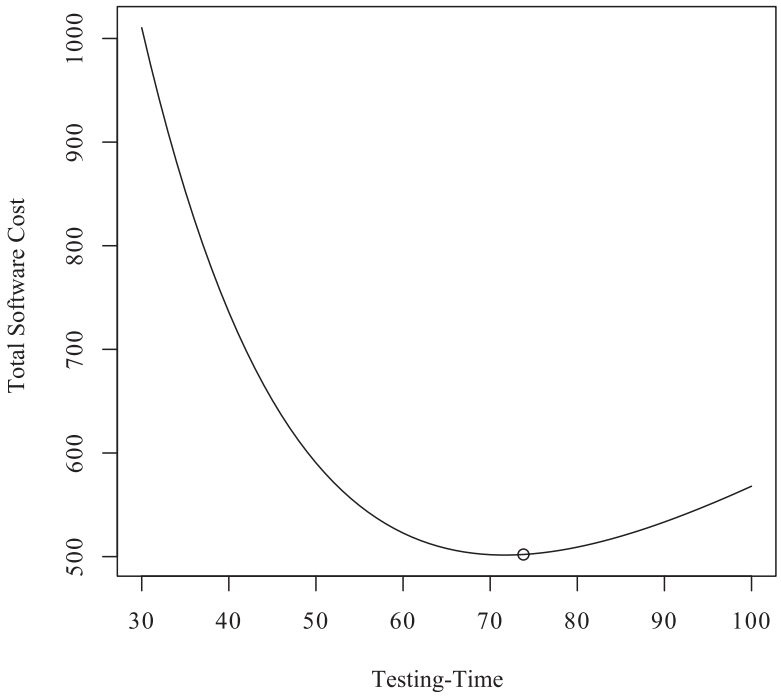

Fig. 4.2 : The expected total software cost. (EXP, DS3)

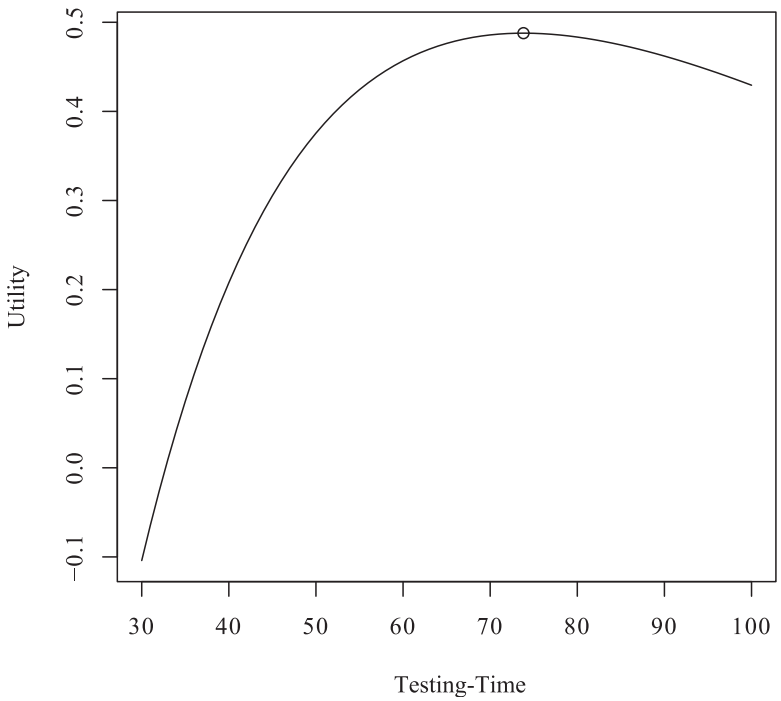

Fig. 4.3 : The multi-attribute utility function. (EXP, DS3)

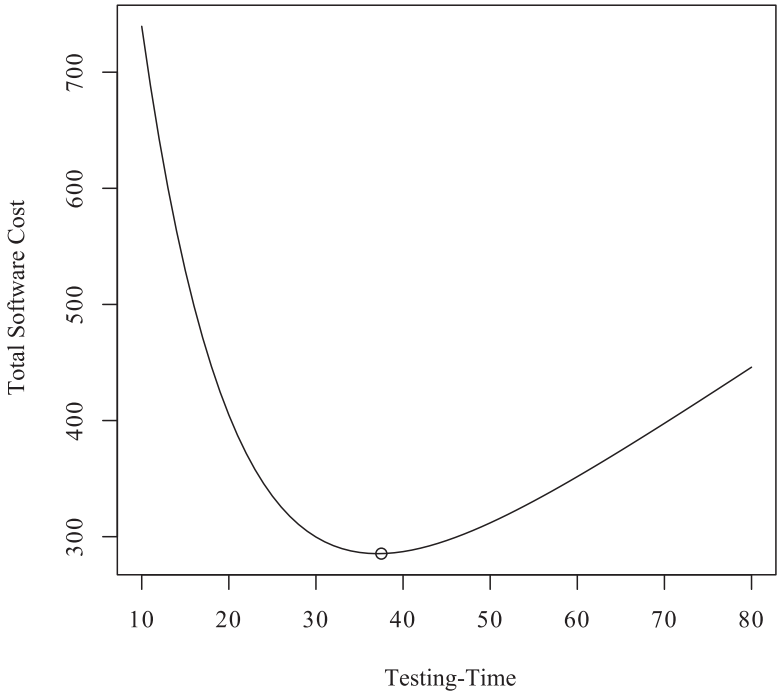

Fig. 4.4 : The expected total software cost. (EXP, DS5)

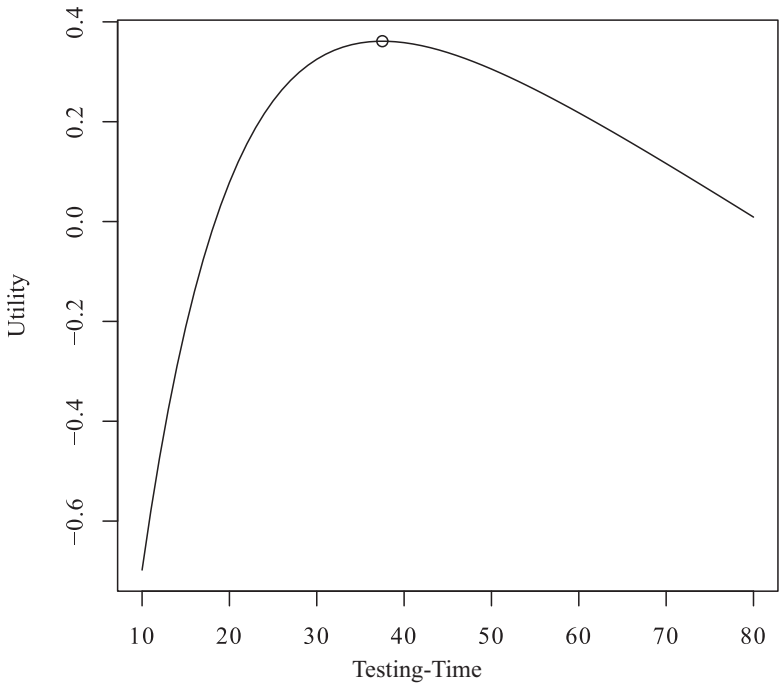

Fig. 4.5 : The multi-attribute utility function. (EXP, DS5)

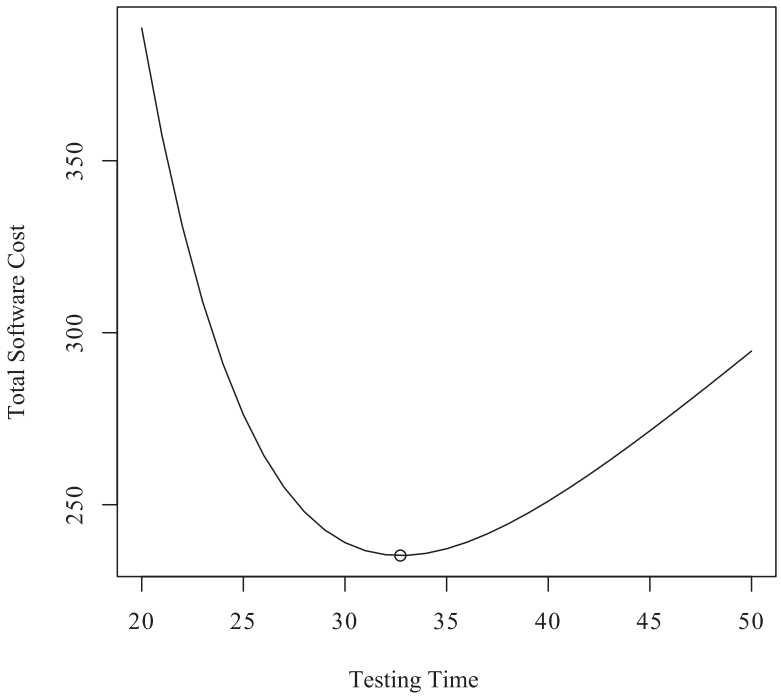

Fig. 4.6 : The expected total software cost. (DSS, DS5)

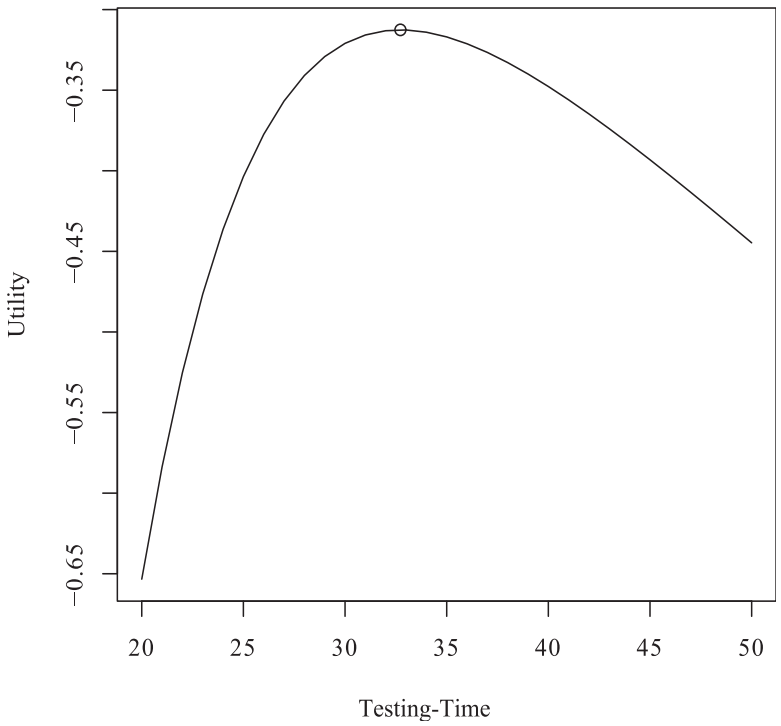

Fig. 4.7 : The multi-attribute utility function. (DSS, DS5)

Chapter 5

Optimal Testing-Resource Allocation Problem Based on MAUT

5.1 Introduction

Normally, software in a testing-phase is tested in the following stages: module, integration, and system testing. Especially, individual software modules are tested independently, and evaluated in terms of the reliability in the module testing. Also, enormous testing-resources are devoted for ensuring enough reliability in the testing-process. The testing-resources mean CPU time, man-power, executed test cases, and so forth. However, they are constrained and the company would like to avoid wasting them by the reason of management aspects. Furthermore, the module testing is said to consume 25% of the total development resources [10]. Therefore, it is very important for the company or software development managers to allocate the testing-resource property in the module testing. This problem is called an *"optimal testing-resource allocation problem"*.

Most of the existing optimal testing-resource allocation problems have been discussed well so far [17, 37, 38, 42, 43]. However, the management aspects of companies for the optimal allocation have not been considered. Also, the existing method can not allocate the testing-resource with considering multiple evaluation criteria. Therefore, we need to discuss an optimal testing-resource allocation problem based on the multi-attribute utility theory. We set reliability, testing-resource, and cost attributes based on the testing-effort dependent SRGM as the evaluation criteria. By applying the multi-attribute utility theory, we can expand the method for the optimal testing-resource allocation as follows: First, We can evaluate an optimal allocation based on the estimated utility by the sensitive analysis. Next, we can consider multiple constraints as the evaluation criteria, such as the cost and reliability, simultaneously. Finally,

the management strategy for the company or software development managers can be reflected to the allocation. From these backgrounds, we estimate the optimal testing-resource expenditures for individual software modules, utility, total amount of optimal testing-resource expenditures, and expected number of remaining faults by optimizing a multi-attribute utility function with the multiple constraints.

This chapter consists of the following sections. In Section 5.2, we introduce an existing optimal testing-resource allocation problem. In Section 5.3, we discuss an optimal testing-resource allocation problem for the case of two attributes. In Section 5.4, we discuss the optimal testing-resource allocation problem for the case of three attributes. In Section 5.5, we show numerical examples for them, respectively. In the final section, we conclude with summary.

5.2 Existing Optimal Testing-Resource Allocation Problem

First, we apply the testing-effort dependent SRGM as a mean value function, $m(t)$. The testing-effort dependent SRGM is defined as

$$m(t) = a(1 - \exp[-rW(t)]), \tag{5.1}$$

where a represents the initial fault content, and r represents the fault detection rate per the testing-resource expenditures, $(0 < r < 1)$. Then, $W(t)$ is given as

$$W(t) = \int_0^t w(x)dx, \tag{5.2}$$

where $w(x)$ represents the testing-resource expenditures at the testing-time t, and $W(t)$ is the cumulative testing-effort function. Also, the expected number of remaining faults is given as

$$\begin{aligned} z(t) &= a - m(t) \\ &= a \cdot \exp[-rW(t)]. \end{aligned} \tag{5.3}$$

We consider the following existing optimal testing-resource allocation problem.

(1) A software system is composed of M independent software modules.

(2) The number of remaining faults in individual software modules can be estimated by the testing-effort dependent SRGM.

(3) The software development managers have to allocate the testing-resource expenditures to individual software modules so that the total number of remaining faults in the software may be minimized.

We assume that the amounts of testing-resource expenditures, q_i, are spent to the test of software module, i ($i = 1, 2, \cdots, M$), in the module testing-process. From Eq. (5.3), the expected number of remaining faults in the software module i, z_i, is given as

$$z_i = a_i \cdot \exp[-r_i q_i], \tag{5.4}$$

where a_i represents the initial fault content for software module, i, and r_i represents the fault detection rate per unit testing-resource, $0 < r_i < 1$. Therefore, from Eq. (5.4), the expected number of remaining faults in the software is given as

$$Z = \sum_{i=1}^{M} z_i. \tag{5.5}$$

Therefore, the software testing-resource allocation problem is formulated as

$$\begin{aligned} &\min : \sum_{i=1}^{M} w_i a_i \cdot \exp[-r_i q_i] \\ &\text{subject to} \quad \sum_{i=1}^{M} q_i \leq Q_S, q_i \geq 0, \end{aligned} \tag{5.6}$$

where w_i represents the weights for individual software modules, Q_S represents the prepared total amount of testing-resource expenditures. To solve the above problem, we consider the following Lagrangian:

$$L = \sum_{i=1}^{M} w_i a_i \cdot \exp[-r_i q_i] + \lambda(\sum_{i=1}^{M} q_i - Q_S). \tag{5.7}$$

The necessary and sufficient conditions for the minimization are

$$\begin{aligned} &\frac{\partial L}{\partial q_i} = -w_i a_i r_i \cdot \exp[-r_i q_i] + \lambda = 0, \\ &\frac{\partial L}{\partial \lambda} = \sum_{i=1}^{M} q_i - Q_S \geq 0, \quad \lambda \geq 0, \\ &\lambda(\sum_{i=1}^{M} q_i - Q_S) = 0. \end{aligned} \tag{5.8}$$

Also, we can assume that the following condition is satisfied for the software modules.

$$A_1 \geq A_2 \geq \cdots A_{k-1} \geq \lambda \geq A_k \geq \cdots \geq A_M, \tag{5.9}$$

where $A_i = w_i a_i r_i$ (i=1,2, $\cdots$, M). Therefore, the optimal testing-resource expenditures, q_i^*, are derived as

$$q_i^* = -\frac{1}{r_i}(\ln A_i - \ln\lambda) \qquad (i = 1, 2, \cdots, k-1), \tag{5.10}$$

$$q_i^* = 0 \qquad (i = k, k+1, \cdots, M). \tag{5.11}$$

Then, ln λ is given by

$$\ln\lambda = \frac{\displaystyle\sum_{i=1}^{M}\frac{1}{r_i}\ln A_i - Q_S}{\displaystyle\sum_{i=1}^{M}\frac{1}{r_i}}. \tag{5.12}$$

That is, the optimal testing-resource expenditures, q_i^*, are given as

$$q_i^* = \max\{0, -\frac{1}{r_i}(\ln A_i - \ln\lambda)\} \qquad (i = 1, 2, \cdots, M). \tag{5.13}$$

5.3 Optimal Testing-Resource Allocation Problem with Two Evaluation Criteria

We consider the following optimal testing-resource allocation problem. Especially, we change the (3) of these assumptions.

(1) A software system is composed of M independent software modules.

(2) The number of remaining faults in individual software modules can be estimated by the testing-effort dependent SRGM.

(3) The software development managers have to allocate the testing-resource expenditures to individual software modules so that the utility may be maximized.

5.3.1 Selection of attributes

We define the digestive rate of the total amount of testing-resource expenditures as the testing-resource attribute. Since the software development managers want to spend less than the prepared total amount of testing-resources, the testing-resource attribute is

$$\min : E = \frac{\sum_{i=1}^{M} q_i}{Q_P} = \frac{Q}{Q_P}, \tag{5.14}$$

where Q (> 0) represents the total amount of testing-resource expenditures. Q_P is the prepared total amount of testing-resources.

Next, we define the rate of the expected number of detected faults in the software as the reliability attribute. From Eq. (5.4), the reliability attribute is given as

$$\max : R = 1 - \frac{\sum_{i=1}^{M} a_i \cdot \exp[-r_i q_i]}{\sum_{i=1}^{M} a_i} = 1 - \frac{Z}{\sum_{i=1}^{M} a_i}. \tag{5.15}$$

As with Chapter 4, we can also say that these attributes have a competitiveness because they are defined by using the same testing-effort dependent SRGM.

5.3.2 Development of single-attribute utility function

The single-attribute utility functions for each attribute are developed based on the following management strategy.

1. For the testing-resource attribute, at least 60% of the prepared total amount of testing-resources must be consumed.

2. For the reliability attribute, at least 80% of software faults should be detected and the more are better.

3. The management team takes the risk neutral position for each attribute.

Then, the lowest and highest consumptions for the testing-resource attribute are $E^L = 0.6$ and $E^H = 1.0$. The lowest and highest requirements for the reliability attribute are $R^L = 0.8$ and $R^H = 1.0$.

We also use the additive linear form. Therefore, we have the following single-attribute utility functions for each attribute:

$$u(E) = 2.5E - 1.5. \tag{5.16}$$

$$u(R) = 5R - 4. \tag{5.17}$$

5.3.3 Development of multi-attribute utility function

From the previous steps, the additive multi-attribute utility function with some constraints as an optimization problem is given as

$$\begin{aligned} \max : u(E,R) &= w_R \times u(R) - w_E \times u(E) \\ &= w_R \times (5R-4) - w_E \times (2.5E - 1.5) \\ \text{subject to} \quad & w_R + w_E = 1,\ q_i \geq 0. \end{aligned} \tag{5.18}$$

where w_R and w_E are weight parameters for the attributes R and E, respectively. Finally, we can obtain the optimal testing-resource expenditures, q_i^* $(i = 1, 2, \cdots, M)$, by maximizing the multi-attribute utility function.

From Eq. (5.18), the optimal testing-resource expenditures, q_i^*, are given by

$$q_i^* = -\frac{1}{r_i}\ln\frac{2.5w_E\sum_{i=1}^{M} a_i}{5w_R a_i r_i Q_P} \qquad (i = 1, 2, \cdots, k-1), \tag{5.19}$$

$$q_i^* = 0 \qquad (i = k, k+1, \cdots, M). \tag{5.20}$$

That is, we can rewrite the optimal testing-resource expenditures, q_i^*, as

$$q_i^* = \max\{0, -\frac{1}{r_i}\ln\frac{2.5w_E\sum_{i=1}^{M} a_i}{5w_R a_i r_i Q_P}\} \qquad (i = 1, 2, \cdots, M).$$

Also, we can assume that the following condition is satisfied for the software modules

$$B_1 \geq B_2 \geq \cdots B_{k-1} \geq \frac{2.5w_E\sum_{i=1}^{M} a_i}{5w_R Q_P} \geq B_k \geq \cdots \geq B_M, \tag{5.21}$$

where $B_i = a_i r_i$ (i=1,2, $\cdots$, M).

5.4 Optimal Testing-Resource Allocation Problem with Three Evaluation Criteria

5.4.1 Selection of attributes

We assume the cost attribute in addition to the testing-resource attribute and reliability attribute. First, we assume the following cost parameters.

c_1 : the debugging cost for one fault in the module testing ($c_1 > 0$),
c_2 : the debugging cost for one undetected fault in the module testing ($c_2 > c_1 > 0$),
c_3 : the cost per unit of testing-resource for the module testing ($c_3 > 0$).

The cost function based on the testing-effort dependent SRGM is formulated as

$$V = c_1 \sum_{i=1}^{M} a_i(1 - \exp[-r_i q_i]) + c_2 \sum_{i=1}^{M} a_i \cdot \exp[-r_i q_i] + c_3 \sum_{i=1}^{M} q_i. \tag{5.22}$$

From Eq. (5.22), the cost attribute is given as

$$\min : C = \frac{V}{C_P}. \tag{5.23}$$

Then, C_P represents the budget.

5.4.2 Development of single-attribute utility function

The single-attribute utility functions for each attribute are developed based on the following management strategy.

1. For the testing-resource attribute, at least 60% of the prepared total amount of testing-resources must be consumed.

2. For the reliability attribute, at least 80% of software faults should be detected and the more better.

3. For the cost attribute, at least 50 % of the budget must be consumed.

4. The management team takes the risk neutral position for each attribute.

From the management strategy, we can rewrite that the lowest and highest consumptions for the cost attribute are $C^L = 0.5$ and $C^H = 1.0$.

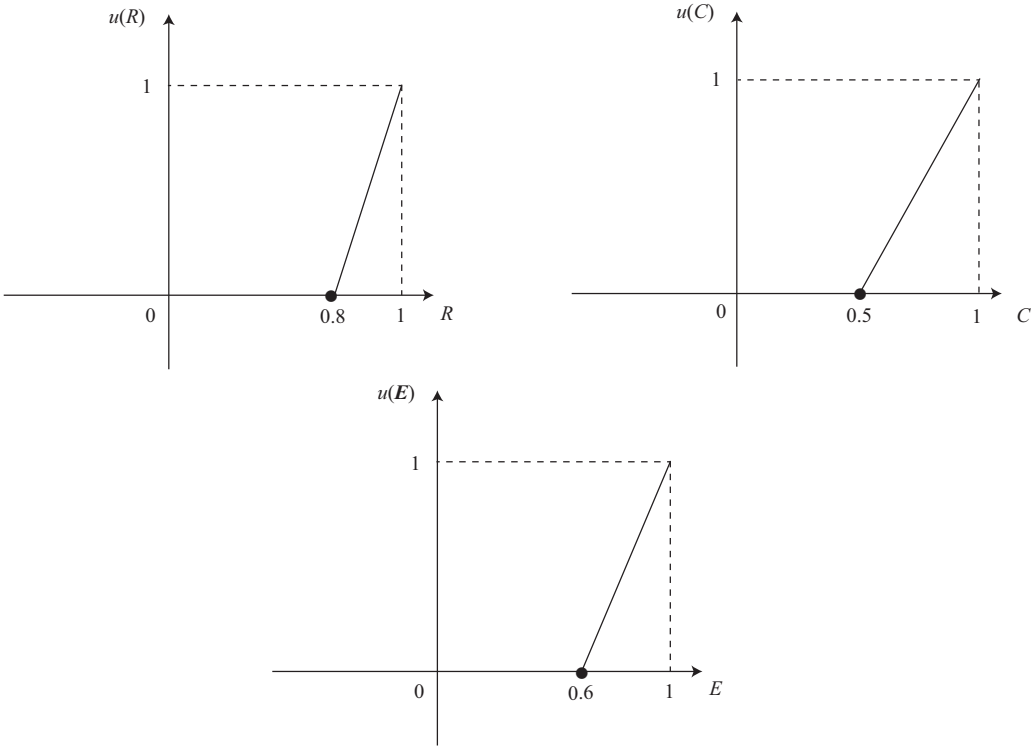

Fig. 5.1 : The single-attribute utility functions.

From the management strategy, we have the following single-attribute utility functions for each attribute:

$$u(E) = 2.5E - 1.5. \tag{5.24}$$

$$u(R) = 5R - 4. \tag{5.25}$$

$$u(C) = 2C - 1. \tag{5.26}$$

5.4.3 Development of multi-attribute utility function

The additive multi-attribute utility function with some constraints as an optimization problem is given by

$$\begin{aligned} \max : u(E,R,C) &= w_R \times u(R) - w_E \times u(E) - w_C \times u(C) \\ &= w_R \times (5R-4) - w_E \times (2.5E - 1.5) - w_C \times (2C-1) \\ \text{subject to} \quad & w_R + w_E + w_C = 1,\ q_i \geq 0. \end{aligned} \tag{5.27}$$

where w_R, w_E, and w_C are the weight parameters for the attributes, R, E, and C, respectively. From Eq. (5.27), the optimal testing-resource expenditures, q_i^*, are given by

$$\begin{aligned} q_i^* &= -\frac{1}{r_i}\ln\frac{\frac{2.5w_E}{Q_P}+\frac{2w_Cc_3}{C_P}}{a_ir_i\left\{\frac{5w_R}{\sum_{i=1}^M a_i}+\frac{(c_2-c_1)\cdot 2w_C}{C_P}\right\}} && (i=1,2,\cdots,k-1), \\ q_i^* &= 0 && (i=k,k+1,\cdots,M). \end{aligned} \quad (5.28)$$

That is, we can rewrite the optimal testing-resource expenditures, q_i^*, as

$$q_i^* = \max\left\{0, -\frac{1}{r_i}\ln\frac{\frac{2.5w_E}{Q_P}+\frac{2w_Cc_3}{C_P}}{a_ir_i\left\{\frac{5w_R}{\sum_{i=1}^M a_i}+\frac{(c_2-c_1)\cdot 2w_C}{C_P}\right\}}\right\} \qquad (i=1,2,\cdots,M). \quad (5.29)$$

Also, when we assume $C_i = a_ir_i (i=1,2,\cdots,M)$, we can assume that the following condition is satisfied for the software modules.

$$C_1 \geq C_2 \geq \cdots \geq C_{k-1} \geq \frac{\frac{2.5w_E}{Q_P}+\frac{2w_Cc_3}{C_P}}{\frac{5w_R}{\sum_{i=1}^M a_i}+\frac{(c_2-c_1)\cdot 2w_C}{C_P}} \geq C_k \geq \cdots \geq C_M. \quad (5.30)$$

5.5 Numerical Examples

we use module-testing data which consists of 10 modules including the initial fault contents for individual software modules, a_i, and fault detection rate per the testing-resource expenditures for individual software modules, r_i, in Table 5.2 [38]. These parameters are estimated by the testing-effort dependent SRGM.

We set the cost parameters as $c_1 = 1.0$, $c_2 = 2.0$, and $c_3 = 5.0$. Also, we set the prepared total amount of testing-resource expenditures, and the budget for the module testing as $Q_P = 1.0\times10^6$ and $C_P = 1.0\times10^6$. Furthermore, the behavior of single-attribute utility functions is shown in Fig. 5.1.

Tables 5.1 and 5.3 show the comparisons of the allocated testing-resource expenditures for individual software modules, total amount of optimal testing-resource expenditures, and utility. "M" means the software module. In Table 5.1, when the weight parameter, w_R, increases, the weight parameter, w_C, decreases. Then, the total amount of optimal testing-resource expenditures, Q^*, increases. Also, we can arrange from P1 to P5 in the order of high utility as P1 > P5 > P2 > P4 > P3. Therefore, we can see that when we weight either the testing-resource attribute or reliability attribute intensively, the utility becomes high. In Table 5.3, when the weight parameters, w_R, increases, w_E and w_C decrease, the total amount of optimal testing-resource expenditures, Q^*, increases. Similarly, we can arrange from P1 to P6 in the

order of high utility as P6 > P1 > P5 > P2 > P3 > P4. Therefore, we can see that when the difference between the maximized weight and minimized weight are large, the utility becomes high.

Tables 5.2 and 5.4 show the expected number of remaining faults based on the optimal testing-resource expenditures, q_i^*. When we focus on P3 in Table 5.2, the expected number of remaining faults is expected to decrease from 251 to 14 by devoting the total amount of optimal testing-resource expenditures. That is, 94.4% of the remaining faults is reduced. Similarly, when we focus on P1 in Table 5.4, the expected number of remaining faults is expected to decrease from 251 to 9 by devoting the total amount of optimal testing-resource expenditures. That is, 96.4% of the remaining faults is reduced.

The optimal testing-resource expenditures for some software modules are $q^* = 0$ because the negative values are derived as the optimal solutions. It does not mean that the module testing is not needed. That is, we can know the software module which is needed the test preferentially.

Finally, Figs. 5.2 and 5.3 show the behavior of the optimal testing-resource expenditures for the modules 1 and 10 in Table 5.2. We can see that if we weight either testing-resource attribute or reliability attribute preferentially, the behavior of the optimal testing-resource expenditures changes the increments or decrements from the midpoint.

5.6 Conclusion

We discussed an optimal testing-resource allocation problem based on the multi-attribute utility theory. Concretely, we developed the cost, reliability, and testing-resource attributes based on the testing-effort dependent SRGM as three evaluation criteria. Then, we optimized the weighted multi-attribute utility function, and derived the optimal testing-resource expenditures, utility, total amount of testing-resource expenditures, and expected number of remaining faults. Furthermore, we checked the sensitivity with changing the combination of the weight parameters, and showed the behavior of the optimal testing-resource expenditures.

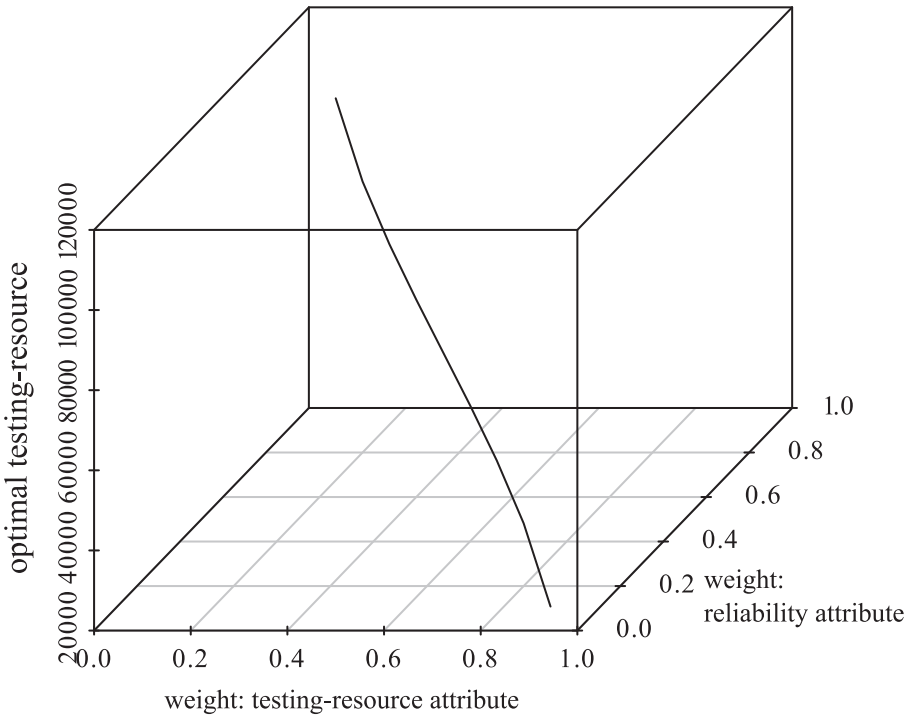

Fig. 5.2 : The behavior of the optimal testing-resource expenditures. (module 1, 2 attributes)

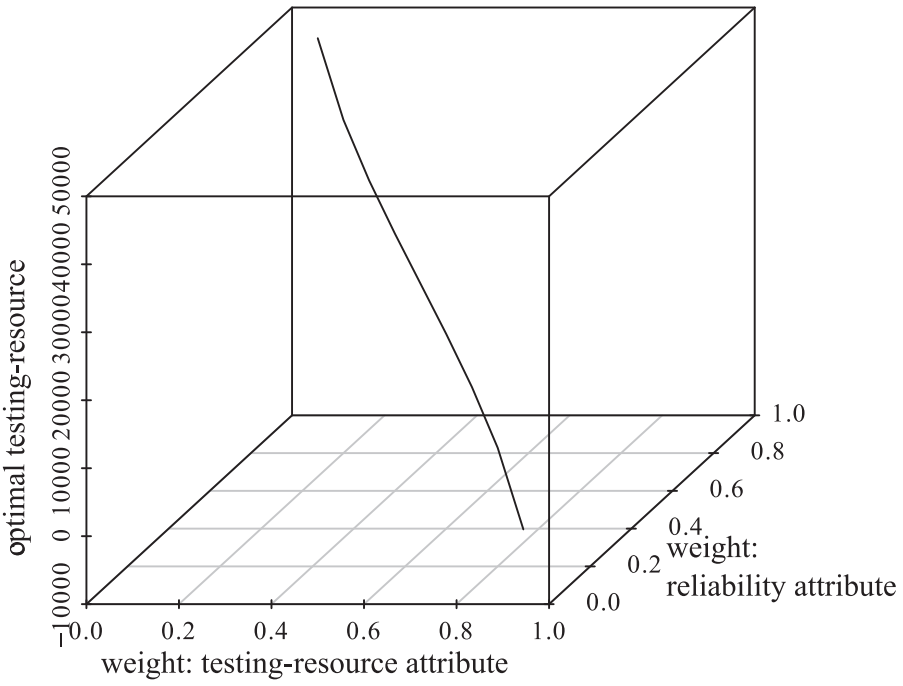

Fig. 5.3 : The behavior of the optimal testing-resource expenditures. (module 10, 2 attributes)

Table 5.1 : The comparison of allocated testing-resource expenditures. (2 attributes)

	Weight		q_i^*					
	w_R	w_E	M1	M2	M3	M4	M5	M6
P1	0.1	0.9	20441	4225.26	1953.46	16397	4794.54	11901.3
P2	0.3	0.7	45758.5	9575.76	4518.88	42512.8	12702.7	35489
P3	0.5	0.5	61649.3	12934.1	6129.11	58904.7	17666.4	50294.1
P4	0.7	0.3	77540.1	16292.4	7739.33	75296.6	22630.1	65099.3
P5	0.9	0.1	102858	21642.8	10304.8	101412	30538.2	88687

	Weight		q_i^*				Q^*	Utility
	w_R	w_E	M7	M8	M9	M10		
P1	0.1	0.9	6177.73	1884.52	579.838	0	68354.67	1.0428
P2	0.3	0.7	19761.2	9629.37	27274.1	13591.2	220813.5	0.763626
P3	0.5	0.5	28287.1	14490.5	44029	23239.3	317623.6	0.70149
P4	0.7	0.3	36812.9	19351.7	60784	32887.5	414433.9	0.753482
P5	0.9	0.1	50396.4	27096.5	87478.2	48259	568672.9	0.879267

Table 5.2 : The comparison of expected number of remaining faults. (2 attributes)

			z_i				
M	a_i	r_i	P1	P2	P3	P4	P5
1	63	5.332×10^{-5}	21.18346	5.491995	2.353713	1.008735	0.261518
2	13	2.523×10^{-4}	4.476812	1.160656	0.497418	0.213179	0.05527
3	6	5.262×10^{-4}	2.146522	0.556507	0.238502	0.102215	0.0265
4	51	5.169×10^{-5}	21.85139	5.665182	2.427936	1.040545	0.269777
5	15	1.707×10^{-4}	6.616869	1.715491	0.735206	0.315087	0.08169
6	39	5.723×10^{-5}	19.73612	5.116784	2.192912	0.939816	0.243657
7	21	9.938×10^{-5}	11.36547	2.946607	1.262824	0.541212	0.140314
8	9	1.743×10^{-4}	6.480209	1.680053	0.720025	0.308579	0.080003
9	23	5.057×10^{-5}	22.33538	5.790641	2.48171	1.063587	0.275745
10	11	8.782×10^{-5}	11.0	3.334478	1.429064	0.612452	0.158784
Z^*			127.1922	33.45839	14.33931	6.145406	1.593256

Table 5.3 : The comparison of allocated testing-resource expenditures. (3 attributes)

	Weight			q_i^*					
	w_R	w_E	w_C	M1	M2	M3	M4	M5	M6
P1	0.8	0.1	0.1	70464.3	14797	7022.33	67997.7	20419.8	58506.9
P2	0.6	0.2	0.2	52069.5	10909.5	5158.39	49022.9	14674	41368.9
P3	0.5	0.25	0.25	44465.5	9302.5	4387.86	41179	12298.8	34284.4
P4	0.4	0.3	0.3	36861.6	7695.53	3617.36	33335.3	9923.68	27200
P5	0.2	0.4	0.4	18468.8	3808.47	1753.61	14362.5	4178.49	10063.8
P6	0.1	0.45	0.45	3264.76	595.318	212.983	0	0	0

	Weight			q_i^*				Q^*	Utility
	w_R	w_E	w_C	M7	M8	M9	M10		
P1	0.8	0.1	0.1	33016.5	17187.1	53323.4	28591.4	371326.43	0.44297
P2	0.6	0.2	0.2	23147.3	11560	33928.3	17423	259261.79	0.166101
P3	0.5	0.25	0.25	19067.5	9233.83	25910.7	12806.2	212936.29	0.102394
P4	0.4	0.3	0.3	14987.8	6907.72	17893.3	8189.46	166611.75	0.0965898
P5	0.2	0.4	0.4	5119.57	1281.2	0	0	59036.44	0.353652
P6	0.1	0.45	0.45	0	0	0	0	4073.061	0.726578

Table 5.4 : The comparison of expected number of remaining faults. (3 attributes)

	z_i					
M	P1	P2	P3	P4	P5	P6
1	1.47105	3.92273	5.88399	8.82579	23.5324	52.9346
2	0.310885	0.829015	1.2435	1.8652	4.97324	11.187
3	0.149062	0.39749	0.596228	0.894317	2.38455	5.36388
4	1.51744	4.04642	6.06954	9.10409	24.2745	51
5	0.459503	1.22532	1.83794	2.75682	7.3506	15
6	1.37055	3.65472	5.48197	8.22276	21.9246	39
7	0.789262	2.10464	3.15691	4.73525	12.6258	21
8	0.450009	1.19999	1.79997	2.69988	7.19878	9
9	1.55105	4.13605	6.20397	9.30573	23	23
10	0.893149	2.38168	3.57246	5.35857	11	11
Z^*	8.96196	23.89806	35.84648	53.76838	138.2645	238.48548

Chapter 6

Conclusion

This thesis discussed several problems, such as he change-point, the optimal software release and optimal testing-resource allocation problems. The obtained main contributions and future studies for the respective chapters are summarized as follows:

- **Chapter 2:**
  Chapter 2 provided the change-point models based on the exponential, the delayed S-shaped, and the inflection S-shaped SRGMs. Also, the change-point models with considering the uncertainty of the testing-environment are provided. They have the better performance than the existing models in terms of MSE. Therefore, we can say that our modeling approach could describe the actual testing-environment. However, the estimation of the inflection coefficient for the ISS-CP SRGM could not have the consistency completely. Therefore, we need to figure out the problem of the estimation, and solve it. Additionally, our proposed models need to be compared the goodness-of-fit by using many actual data sets. As future study, we have to develop more testing-environmental functions, and check the performance of our models which are applied them. Also, we proposed a change-point detection method based on the Laplace trend test. Concretely, we estimated the change-points by analyzing the behavior of the Laplace factor, and checked the performance of our proposed change-point models with the detected change-points. We confirmed that those change-point models have the better performance from the numerical examples. However, the detection method does not have the consistency completely because the relationship between the change-point and Laplace factor is not described theoretically. Therefore, we should make a correlation of them mathematically. As future study, we need to improve the accuracy of our detection method. Furthermore, we have to try the differentiation of change-points as an evaluation criterion for

determining a change-point.

- **Chapter 3:**
  Chapter 3 discussed an optimal software release problem based on the change-point model analytically. We derived the optimal software release time and optimal testing-time duration from the change-point to the termination time of testing when the total software cost is minimized in terms of the cost-evaluation criterion. After that, we considered the reliability-evaluation criterion additionally, and determined the conclusive optimal software release time. However, we could not estimate the optimal software release time for the partial data-set properly. Therefore, we need to figure out the problem, and improve the theoretical consistency.

- **Chapter 4:**
  Chapter 4 discussed an optimal software release problem based on the multi-attribute utility theory. The cost and reliability attributes based on the change-point models were considered as the evaluation criteria. The optimal software release time and optimal testing-time duration from the change-point to the termination time of testing were estimated simultaneously. Also, we derived the optimal occurrence-time of the change-point, expected total software cost, and utility. As future study, we need to consider the evaluation method of utility because our approach does not have the concrete evaluation criterion. In other words, it is difficult to evaluate the results by the absolute evaluation because the utility does not have units. Therefore, as another perspective, we try to develop new evaluation functions with unit.

- **Chapter 5:**
  Chapter 5 discussed an optimal testing-resource allocation problem based on the multi-attribute utility theory. The cost, testing-resource, and reliability attributes based on the testing-effort dependent SRGM were considered as the evaluation criteria. Also, we estimated the optimal testing-resource expenditures for the module testing in terms of utility. After that, the expected number of remaining faults was estimated by using them. In the numerical examples, we showed the behavior of the optimal testing-resource expenditures, and conducted the sensitive analysis. As future study, as with Chapter 4, we need to discuss concrete evaluation method by using derived utility.

We discussed these several problems for software development management in this thesis.

Our change-point models and its application are expected to be a countermeasures for change-points because it is easy for software development managers to apply in the actual testing-environment. Also, our approach by using the multi-attribute utility theory is expected to be a new methodology in terms of the economics.

As future study, we need to discuss the above problems, and improve the accuracy of our proposed software development management methods. We could not derive the optimal solutions properly in the partial problems although we used the optimization methodologies for solving the optimization problems in this thesis. Therefore, we try to apply various optimization methodologies in the mathematical programming to our approach.

References

[1] C.T. Lin and C.Y. Huang, "Enhancing and measuring the predictive capabilities of testing-effort dependent software reliability models," *Journal of Systems and Software*, Vol. 81, No. 6, pp. 1025–1038, 2008.

[2] C.Y. Huang, "Performance analysis of software reliability growth models with testing-effort and change-point," *Journal of Systems and Software*, Vol. 76, No. 2, pp. 181–194, 2005.

[3] C.Y. Huang, "Cost-reliability-optimal release policy for software reliability models incorporating improvements in test efficiency," *Journal of Systems and Software*, Vol. 77, No. 2, pp. 139–155, 2005.

[4] C.Y. Huang and T.Y. Hung, "Software reliability analysis and assessment using queueing models with multiple change-points," *Computers & Mathematics with Applications*, Vol. 60, No. 7, pp. 2015–2030, 2010.

[5] C.Y. Huang and C.T. Lin, "Analysis of software reliability modeling considering testing compression factor and failure-to-fault relationship," *IEEE Transactions on Computers*, Vol. 59, No. 2, pp. 283–288, 2010.

[6] H.J. Shyur, "A stochastic software reliability model with imperfect-debugging and change-point," *Journal of Systems and Software*, Vol. 66, No. 2, pp. 135–141, 2003.

[7] H. Okamura, T. Dohi, and S. Osaki, "A reliability assessment method for software products in operational phase—proposal of an accelerated life testing model," (in Japanese), *The Transactions of IEICE*, Vol. J83-A, No. 3, pp. 294–301, 2000.

[8] H. Pham, *Software Reliability*, Springer-Verlag, Singapore, 2000.

[9] H. Pham, *Handbook of Reliability Engineering*, pp. 285–302, Springer-Verlag, London, 2003.

[10] H. Pham, *Handbook of Engineering Statistics*, Springer-Verlag, London, 2006.

[11] H. Pham (Ed.), *Recent Advances in Reliability and Quality in Design*, Springer-Verlag, London, 2008.

[12] H. Tamura, Y. Nakamura, and S. Fujita, *Mathematical Science of Utility Analysis and Its Applications* (in Japanese), Corona publishing, Tokyo, 1997.

[13] H. Wang and H. Pham, *Reliability and Optimal Maintenance*, Springer-Verlag, London, 2006.

[14] J. Zhao and J. Wang, "Testing the existence of change-point in NHPP software reliability models," *Communications in Statistics—Simulation and Computation*, Vol. 36, pp. 607–619, 2007. (DOI: 10.1080/03610910701236099)

[15] K.B. Misra (Ed.), *Handbook of Performability Engineering*, Springer-Verlag, London, 2008.

[16] K. Narita, *A Friendly Guide to Probability Models with Examples and Solutions to Problems* (in Japanese), Kyoritsu-Shuppan, Tokyo, 2010.

[17] M. Nishiwaki, S. Yamada, and T. Ichimori "Testing-resource allocation policies based on an optimal software release problem," *Journal of Japan Industrial Management Association*, Vol. 46, No. 3, pp. 182–186, 1995.

[18] M. Ohba, "Inflection S-shaped software reliability growth model," in *Stochastic Models in Reliability Theory*, S. Osaki, and Y. Hatayama (eds.), pp. 144–165, Springer-Verlag, Berlin, 1984.

[19] M.R. Lyu, *Handbook of software reliability engineering*, M.R. Lyu (ed.), pp. 3–25, IEEE Computer Society Press and McGraw-Hill, New York, 1996.

[20] M. Zhao, "Change-point problems in software and hardware reliability," *Communication in Statistics—Theory and Methods*, Vol. 22, No. 3, pp. 757–768, 1993.

[21] N. Langberg and N.D. Singpurwalla, "A unification of some software reliability models," *SIAM Journal on Scientific Computing*, Vol. 6, No. 3, pp. 781–790, 1985.

[22] O. Gauodin, "Optimal properties of the Laplace trend test for software-reliability model reliability," *IEEE Transactions on Reliability Model*, Vol. 41, No. 5, pp. 376–381, 1979.

[23] O. Singh, P.K. Kapur, and A. Anand, "A multi-attribute approach for release time and reliability trend analysis of a software," *International Journal of System Assurance Engineering and Management*, Springer-Verlag, Vol. 3, No. 3, pp. 246–254, 2012.

[24] O. Singh, P.K. Kapur, and A.K. Shrivastava, "Release time problem with multiple constraints," *International Journal System Assurance Engineering and Management*, Springer-Verlag, Vol. 6, No. 1, pp. 83–91, 2015.

[25] P.K. Kapur, V.B. Singh, O. Singh, and J.N.P. Singh, "Software release time based on different multi-attribute utility functions," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 20, No. 4, 1350012, 2013.

[26] P.K. Kapur, K.K. Sunil, T. Anshul, and S. Omar, "Release time determination depending on number of test runs using multi attribute utility theory," *International Journal of System Assurance Engineering and Management*, Springer-Verlag, Vol. 5, No. 2, pp. 186–194, 2014.

[27] S. Inoue and S. Yamada, "Optimal software release problems based on discrete NHPP models," (in Japanese), *IEICE Technical Report [Reliability]*, Vol. 102, No. 58, pp. 7–12, 2002.

[28] S. Inoue and S. Yamada, "Discretized software reliability growth models and its applications," (in Japanese), *IEICE Technical Report [Reliability]*, Vol. 104, No. 220, pp. 25–30, 2004.

[29] S. Inoue and S. Yamada, "Software reliability measurement with change-point," *Proceedings of the Fifth International Conference on Quality and Reliability*, Chiang-Mai, Thailand, 2007, pp. 170–175.

[30] S. Inoue and S. Yamada, "Optimal software release policy with change-point," *Proceedings of the 2008 IEEE International Conference on Industrial Engineering Management (IEEM2008)*, Singapore, December 8-11, 2008, CD-ROM (IEEE Catalog Number: CFP08IEI-CDR), pp. 531–535.

[31] S. Inoue and S. Yamada, "Environmental-function-based change-point modeling for software reliability measurement," *Proceedings of the Tenth International Conference on Industrial Management*, Beijing, China, September 16–18, 2010, pp. 403–407.

[32] S. Inoue and S. Yamada, "Software reliability growth modeling frameworks with change of testing-environment," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 18, No. 4, pp. 365–376, 2011.

[33] S. Osaki (Ed.), "Stochastic Models in Reliability and Maintenance," Springer-Verlag, Tokyo/Heidelberg, 2002.

[34] S.S. Gokhale and K.S. Trivedi, "Log-logistic software reliability growth model," *Proceedings of the High-Assurance System Engineering Symposium 1998*, Washington D.C., U.S.A., November 13-14, 1998, pp. 34–41.

[35] S. Yamada, *Software Reliability Models: Fundamentals and Applications* (in Japanese), JUSE Press, Tokyo, 1994.

[36] S. Yamada, *Elements of Software Reliability: Modeling Approach* (in Japanese), Kyoritsu-Shuppan, Tokyo, 2011.

[37] S. Yamada, *Software Reliability Modeling —Fundamentals and Applications—*, Springer Japan, Tokyo/Heidelberg, 2014.

[38] S. Yamada and H. Ohtera, *Software Reliability —Theory and Practical Application—* (in Japanese), Soft Research Center, Tokyo, 1990.

[39] S. Yamada and M. Takahashi, *Introduction to Software Management Model —A Method of Evaluation and Visualization of Software Quality* (in Japanese), Kyoritsu-Shuppan, Tokyo, 1993.

[40] S. Yamada and S. Osaki, "Software Reliability Growth Modeling: Models and Applications," *IEEE Transaction on Software Engineering*, Vol. SE-11, No. 12, pp. 1431–1437, 1985.

[41] S. Yamada and T. Fukushima, *Quality-oriented software management* (in Japanese), Morikita-Shuppan, Tokyo, 2007.

[42] S. Yamada, T. Ichimori, and M. Nishiwaki "Optimal allocation policies for testing-resource based on a software reliability growth model," *International Journal of Mathematical and Computing Modeling*, Vol. 22, No. 10–12, pp. 295–301, 1995.

[43] T. Ichimori, M. Tanaka, and S. Yamada, "An optimal effort allocation problem for both module and integration testing in software development," *Journal of Japan Industrial Management Association*, Vol. 53, No. 3, pp. 201–207, 2002.

[44] W.S. Humphrey, *Managing the Software Process* (in Japanese), JUSE Press, Tokyo, 1991.

[45] X. Teng and H. Pham, "A new methodologies for predicting software reliability in the random field environments," *IEEE Transaction on Reliability*, Vol. 55, No. 3, pp. 458–468, 2006.

[46] Y.P. Chang, "Estimation of parameters for nonhomogeneous Poisson process software reliability with change-point model," *Communications in Statistics — Simulation and Computation*, Vol. 30, No. 3, pp. 623–635, 2001.

Publication List of the Author

(Refereed Papers)

1. Y. Minamino, S. Inoue, and S. Yamada, "NHPP models for software reliability measurement with change-point," *Proceedings of the 17th ISSAT International Conference on Reliability and Quality in Design*, Vancouver, Canada, August 4-6, 2011, pp. 132–136.

2. Y. Fukuta (Minamino), S. Inoue, and S. Yamada, "NHPP models with change-point for software reliability assessment and its application to an optimal software release problem," *Proceedings of the 19th ISSAT International Conference on Reliability and Quality in Design*, Honolulu, Hawaii, U.S.A., August 5-7, 2013, pp. 93–97.

3. Y. Minamino, S. Inoue, and S. Yamada, "On application methodologies of software reliability model with change-point," *Asia Pacific Journal of Industrial Management (APJIM)*, Vol. V, Issue 1, pp. 63–70 , 2014.

4. Y. Minamino, S. Inoue, and S. Yamada, "Change-point modeling and detection method for software reliability assessment," *Proceedings of the 12th International Conference on Industrial Management (ICIM)*, Chendu, China, September 3–5, 2014, pp. 266-270.

5. Y. Minamino, S. Inoue, and S. Yamada, "multi-attribute utility theory for estimation of optimal release time," *International Journal of Reliability, Quality and Safety Engineering(IJRQSE)*, Vol. 22, No. 4, 1550019-1—1550019–14, 2015.

6. Y. Minamino, S. Inoue, and S. Yamada, "Estimating optimal software release time based on a change-point model by multi-attribute utility theory," *Proceedings of the 21st ISSAT International Conference on Reliability and Quality in Design*, Philadelphia, Pennsylvania, U.S.A., August 6-8, 2015, pp. 89–93.

7. Y. Minamino, S. Inoue, and S. Yamada, "NHPP-based change-point modeling for software reliability assessment and its application to software development management," *Annals*

of Operations Research, Vol. 238, 2016. (DOI: 10. 1007/s10479-016-2148-x)

8. Y. Minamino, S. Inoue, and S. Yamada, "A testing-resource allocation problem with multiple constraints for software development," *Proceedings of the 7th Asia-Pacific International Symposium on Advanced Reliability and Maintenance Modeling (APARM)*, Seoul, Korea, August 24-26, 2016, pp. 132–136.

9. Y. Minamino, S. Inoue, and S. Yamada, "A software testing-resource allocation problem with multi-attributes for module testing," *Proceedings of the 13th International Conference on Industrial Management (ICIM)*, Hiroshima, Japan, September 21-23, 2016, pp. 340–345.

(Technical Reports)

1. Y. Minamino, S. Inoue, and S. Yamada, "A study on an optimal software release problem based on a change-point model by using multi-attribute utility theory," (in Japanese) *The Institute of Statistical Mathematics Cooperative Research Report 369—Optimization : Modeling and Algorithms 28*, pp. 79–84, March 2015.

2. Y. Minamino, S. Inoue, and S. Yamada, "A study on an estimation of optimal software release time and change-point based on multi-attribute utility theory," (in Japanese) The Research Institute for Mathematical Sciences of Kyoto University Research Report 1990 —Mathematical Programming Concerning Decision Makings and Uncertainties—, pp. 191–197, April 2016.

(National Conference and Symposium Presentations)

1. Y. Minamino, S. Inoue, and S. Yamada, "Change-point modeling for software reliability assessment based on NHPP," (in Japanese) *Proceedings of the 13th IEEE Hiroshima Section Student Symposium*, Hiroshima, Japan, November 2011, pp. 162–163.

2. Y. Minamino, S. Inoue, and S. Yamada, "On applications of change-point modeling technology into NHPP software reliability growth models," *Proceedings of the 4th Japan Korea Software Management Symposium*, Jeollabuk, Korea, November 18, 2011, pp. 95–104.

3. Y. Fukuta (Minamino), S. Inoue, and S. Yamada, "A study on NHPP modeling with a testing-environmental factor and its application to an optimal software release problem," *Proceedings of the 39th Japan Industrial Management Association Chugoku-Shikoku Branch Student Conference*, Tottori, Japan, March 2, 2013, pp. 3–4.

4. Y. Minamino, S. Inoue, and S. Yamada, "An optimal software release problem with multi-attribute based on a change-point model," *Proceedings of the 8th Japan KoreaSoftware Management Symposium*, Asan, Korea, November 27, 2015, pp. 58–73.

5. Y. Minamino, S. Inoue, and S. Yamada, "A study on an estimation of optimal software release time and change-point with multiple evaluation criteria," (in Japanese) *Proceedings of the 2016 Spring National Conference of ORSJ*, Yokohama, Japan, March 17–18, 2016, pp. 81–82.

Received Awards List of the Author

1. Distinguished Service Award (January 23, 2012)
   The 13th IEEE Hiroshima Section Student Symposium (HISS 2011), Hiroshima, Japan, November 12–13, 2011.

2. Best Research Award (November 13, 2011)
   The 13th IEEE Hiroshima Section Student Symposium (HISS 2011), Hiroshima, Japan, November 12–13, 2011
   –Title–
   Y. Minamino, S. Inoue, and S. Yamada, "Change-point modeling for software reliability assessment based on NHPP," (in Japanese) *Proceedings of the 13th IEEE Hiroshima Section Student Symposium (HISS 2011)*, Hiroshima, Japan, November 12–13, 2011, pp. 162–163.

3. Outstanding Research Presentation Award (March 2, 2013)
   The 39th Japan Industrial Management Association Chugoku-Shikoku Branch Student Conference, Tottori, Japan, March 2, 2013.
   –Title–
   Y. Fukuta (Minamino), S. Inoue, and S. Yamada, "A study on NHPP modeling with a testing-environmental factor and its application to an optimal software release problem," *Proceedings of the 39th Japan Industrial Management Association Chugoku-Shikoku Branch Student Conference*, Tottori, Japan, March 2, 2013, pp. 3–4.

4. Excellent Paper Award (September 6, 2014)
   The 12th International Conference on Industrial Management (ICIM), Chendu, China, September 3–5, 2014.
   —Title—
   Y. Minamino, S. Inoue, and S. Yamada, "Change-point modeling and detection method

for software reliability assessment," *Proceedings of The 12th International Conference on Industrial Management*, Chendu, China, September 3-5, 2014, pp. 266–270.

Received Grants List of the Author

1. The International Conference Travel Grant from the Telecommunications Advancement Foundation (TAF). (2015)
   —Conference Name—
   The 21st ISSAT International Conference on Reliability and Quality in Design, Philadelphia, Pennsylvania, U.S.A., August 6–8, 2015.

2. The Encourage Fund from Tottori University. (February 2015)
   —Subject—
   "A Study on Next-Generation Mathematical Model Which Reflects the Actual Environment for the High Accuracy of Software Reliability Assessment Technology" (in Japanese)

3. The Encourage Fund from Tottori University. (February 2016)
   —Subject—
   "A Study on Software Development Management Method Based on Decision-Making Mechanism" (in Japanese)