A Study on Stochastic Modeling for Accurate Software Reliability Assessment

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

Shinji Inoue

Under the supervision of
Professor Shigeru Yamada

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

January 2006

ABSTRACT

Accurate software reliability assessment is one of the current issues to assess software quality of software systems of which the size, complexity, and diversification have grown drastically. This doctoral dissertation discusses several methods of improved estimation/prediction accuracy for software quality/reliability assessment. Especially, stochastic software reliability modeling which enables us to assess software reliability more accurately is discussed. This dissertation consists of the four main parts.

In the first part discretized software reliability modeling and its application to optimal software release problems are discussed. This part focuses on accuracy improvement of conventional continuous-time software reliability models by discretizing techniques such as integrable difference methods. The discretized software reliability models proposed in this part have better performance than the continuous-time ones in terms of the effort of parameter estimation, goodness-of-fit to actual data, and predictability. The second part discusses software reliability growth modeling with several key factors related to the software reliability growth process, such as testing-coverage and testing-effort expenditures, based on a nonhomogeneous Poisson process and stochastic differential equations, respectively. Taking these factors into consideration in software reliability modeling is effective to develop plausible software reliability growth models. In the third part generalized software reliability modeling techniques based on an infinite server queueing and order-statistics theories are discussed. These generalization approaches are expected to be plausible software reliability modeling since these generalized software reliability models proposed in the third part describe the software failure-occurrence phenomenon or the fault-detection phenomenon comprehensively. The forth part treats software reliability modeling under imperfect debugging environment which is assumed that debugging activities can not always correct and detect faults perfectly in the testing-phase during software development.

Each chapter shows numerical examples of software reliability analysis based on the software reliability models proposed in this dissertation by using actual fault data. The final chapter summarizes the results obtained in this dissertation and refers to the future researches on plausible software reliability modeling and its applications to project management problems.

ACKNOWLEDGEMENTS

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

And I am indebted to Professor Hajime Kawai and Professor Satoru Ikehara of Tottori University, Japan, the members of this dissertation reviewing commitee, for their helpful comments and checking the manuscript.

I wish to thank Associate Professor Koichi Tokuno of Tottori University, Japan, Associate Professor Mitsuhiro Kimura of Hosei University, Japan, and Assistant Professor Yoshinobu Tamura of Tottori University of Environmental Studies, Japan, for their many helpful suggestions and continual warm encouragement. I would like to greatly acknowledge Dr. Takaji Fujiwara of Fujitsu Peripherals Limited, Japan, who gives me many invaluable comments and suggestions for my study in terms of practical software reliability assessment and encourages me warmly. And I also wish to thank Research Professor Hee-Soo Kim of Chosun University, Korea, for her many helpful comments and warm encouragement.

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

Special thanks are due to Mr. Toshiyuki Toyota, Mr. Takafumi Yamamoto, Ms. Yuko Mitsuhata, Mr. Shuichi Iriguchi, Mr. Atsushi Fukuta, Mr. Hajime Morita, and the other past and present members of Information Systems Laboratory (Professor Yamada's Laboratory) of Department of Social Systems Engineering at Tottori University, Japan. This work could not be achieved without their continual warm encouragement.

Finally and most importantly, I wish to thank my parents (Mr. Tsutomu Inoue and Ms. Setsuko Inoue), my grandparent (Mr. Masaharu Inoue), my brother (Mr. Yasuki Inoue), and my brother's wife (Ms. Natsue Inoue), for their understanding and warm support. It is to them that this dissertation is dedicated.

Shinji Inoue
January 2006

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Software Quality/Reliability

Computer systems play an important role in our modern information society. They are used in diverse area, for example, seat reservation systems, online transaction systems, hospital patient monitoring systems, air traffic control systems, and so on. If the computer systems once cause failures, it is possible that inconvenience to our social life is caused, our property is damaged, and people are injured or killed as the most critical case. The computer systems consist of the hardware and software subsystems. It is said that the hardware has attained high productivity, quality, and reliability by progress of production technologies and redundancy techniques. Accordingly, the main concern about productivity, quality, and reliability of computer systems has been changing from the hardware into the software systems. *Software engineering* [3, 20] is an academic field treating these issues about the software systems. The main issue of software engineering is to improve the productivity and quality of software systems by using scientific approaches. The software development technologies discussed in software engineering can be classified into the inherent and management technologies. The inherent technologies, such as specification, design, coding/programming, testing/validation, maintenance techniques, aid the software development directly. And the management technologies, such as quality/reliability, performance/economic assessment, and project management techniques, support the software development process indirectly. This dissertation discusses the management technologies in terms of software quality/reliability and project management.

Software quality is defined as the attribute measuring how well the software product meets stated user functions and requirements [3]. Therefore, it is very important to transform the user's requirements (external quality characteristics) into the quality characteristics of the software system developers (internal quality characteristics). The external quality characteristics

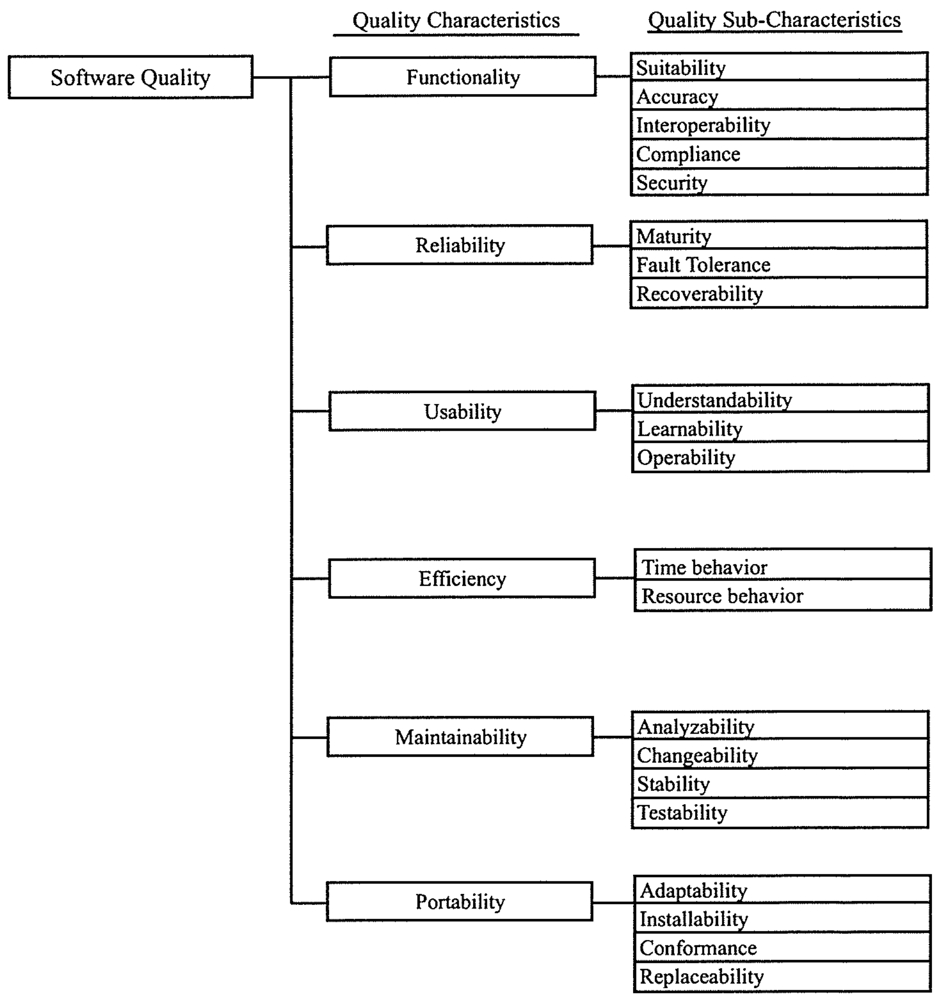


Fig. 1.1 : Software quality characteristics and sub-characteristics (ISO/IEC 9126).

have been standardized as the following six characteristics by ISO/IEC: Functionality, reliability, usability, efficiency, maintainability, and portability [52, 55] (see Fig. 1.1). In the software quality characteristics standardized by ISO/IEC, *software reliability* is defined as the attribute that a software system will perform without causing a software failure over a given time period, under specified operational environment [51, 52, 56]. Accordingly, the software quality characteristic of software reliability has been considered to be a "*must-be quality*" of software products. And software reliability characteristics are different from hardware ones since a software system never deteriorates.

A software system is a product which consists of the programs and documents produced through the software development process, and are produced along with the following software development process: Requirement specification, design, coding, and testing. Fig. 1.2 shows a

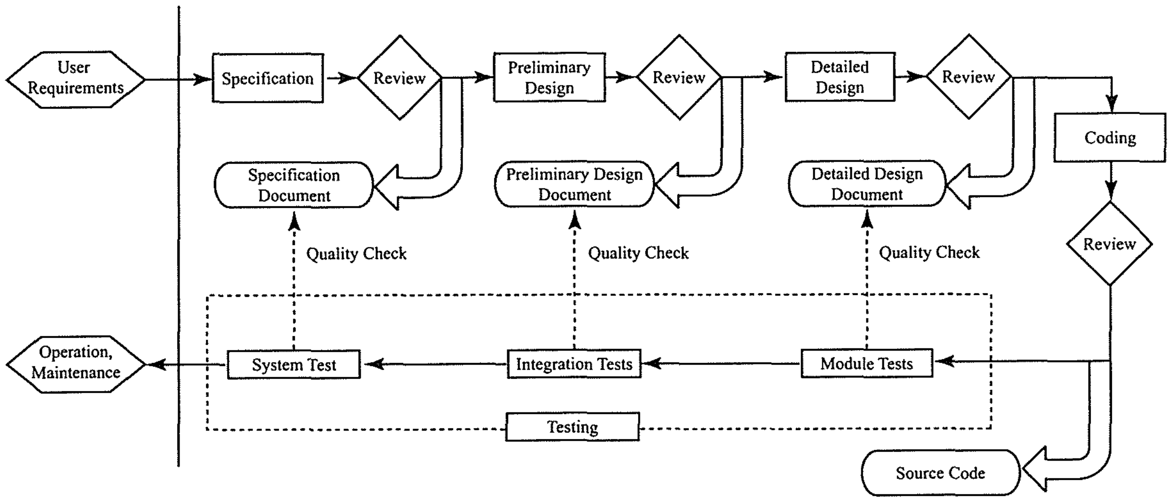


Fig. 1.2 : A general software development process (water-fall paradigm).

general software development process called a water-fall paradigm [57]. Therefore, the software development managers has to control the software development process systematically to satisfy the software quality. Then, *total quality management* (abbreviated as *TQM*) is considered to be one of the key technologies to produce more high quality software products [64]. The concept of the TQM for software development means to assure the quality of software products in each phase to the next phase, such as preventing software faults from causing software failures as much as possible and detecting introduced faults in the software system as early as possible. In practical software development, the software quality is evaluated and assured in the testing-phase which is the final phase of the software development. That is, a lot of software faults introduced in the software system through the first three phases of the software development process by human activities are detected finally. Therefore, we can see that continual software process improvement is very important to eliminate faults introduced during software development process to reduce maintenance cost during operation phase. The present concept of a software quality control is that good software process makes good software products. In recent years, *capability maturity model* (abbreviated as *CMM*) [16] has been proposed as one of the key technologies for software process improvement.

1.2 Software Reliability Engineering

There is now a general agreement on the need to increase software reliability by eliminating faults introduced during software development. It needs measurement/assessment technologies

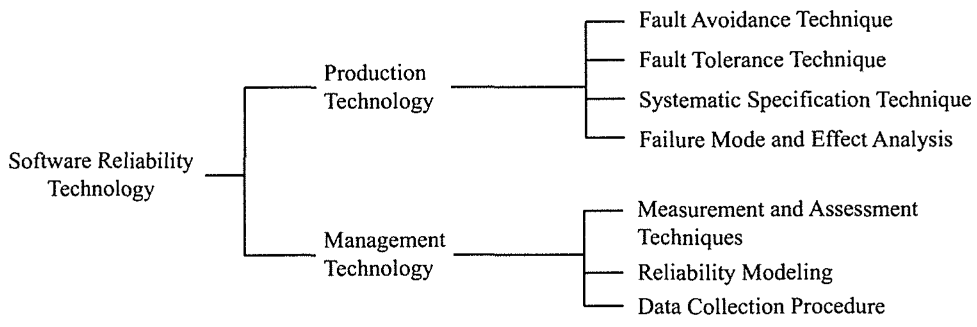

Software Reliability Technology
Production Technology
Management Technology
Fault Avoidance Technique
Fault Tolerance Technique
Systematic Specification Technique
Failure Mode and Effect Analysis
Measurement and Assessment Techniques
Reliability Modeling
Data Collection Procedure

Fig. 1.3 : Software reliability technologies [52].

of software reliability in the software development process to produce software products keeping high degree of reliability. Up to now, a lot of software reliability technologies have been discussed, and are roughly classified into *production technology* and *management technology.* Fig. 1.3 shows such software reliability technologies briefly [52]. The production technology is defined as techniques related to improving and maintaining software reliability directly. The management technology is defined as techniques related to managing software reliability, such as assessing and predicting software reliability.

Software reliability engineering [7,10,38,51,52,55,56] (abbreviated as *SRE*) is a discipline of applying engineering principles and quantitative techniques to assure and enhance software reliability, and known as a subdispline of software engineering. Research on the SRE has been conducting during the past 25 years to grapple with the problems mentioned above. The main purposes on the SRE are measuring, predicting, and controlling reliability of a software system. Fig. 1.4 shows SRE activities along with the software development process [10,52].

Now, we define the following terminologies related to the SRE as follows [38,51,52,54–56]:

- *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 a software system containing a software fault.

Thus, the software fault is considered to be a manifestation of a software error. The severity of

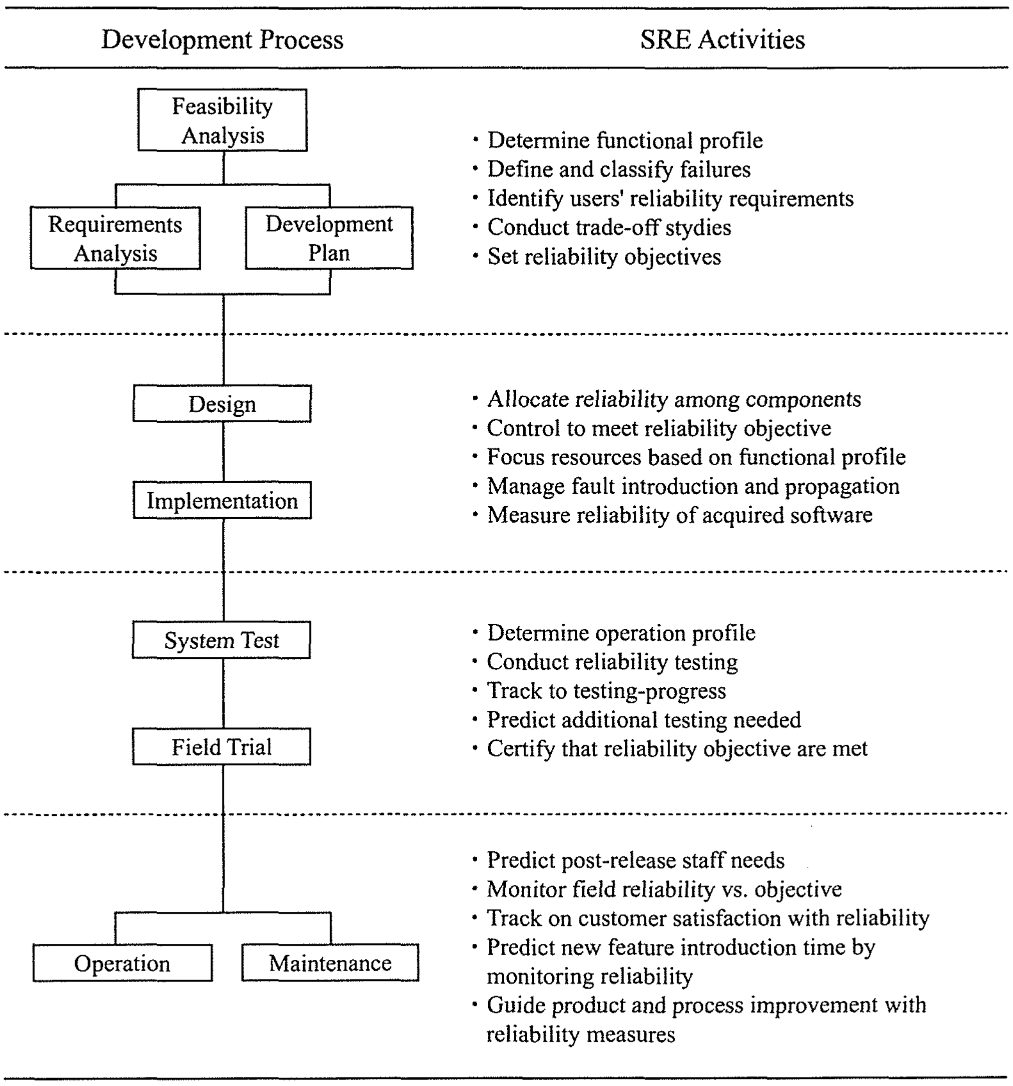

Development Process
SRE Activities
Feasibility Analysis
Requirements Analysis
Development Plan
• Determine functional profile
• Define and classify failures
• Identify users' reliability requirements
• Conduct trade-off stydies
• Set reliability objectives
Design
Implementation
• Allocate reliability among components
• Control to meet reliability objective
• Focus resources based on functional profile
• Manage fault introduction and propagation
• Measure reliability of acquired software
System Test
Field Trial
• Determine operation profile
• Conduct reliability testing
• Track to testing-progress
• Predict additional testing needed
• Certify that reliability objective are met
Operation
Maintenance
• Predict post-release staff needs
• Monitor field reliability vs. objective
• Track on customer satisfaction with reliability
• Predict new feature introduction time by monitoring reliability
• Guide product and process improvement with reliability measures

Fig. 1.4 : SRE activities over the software product life-cycle phases.

the level of software failures may be able to classify into several categories, such as catastrophic, critical, major, and minor, depending on their impact on our social life [38].

1.3 Software Reliability Models

Software reliability models are mathematical tools based on stochastic and statistical principles to assess software reliability quantitatively. Fig. 1.5 shows the hierarchical classification of software reliability models [52, 55]. The software reliability models are roughly classified into *analytical models* and *empirical models*. The role of the analytical model is to derive and analyze several software reliability measures, such as the fault content in the software system,

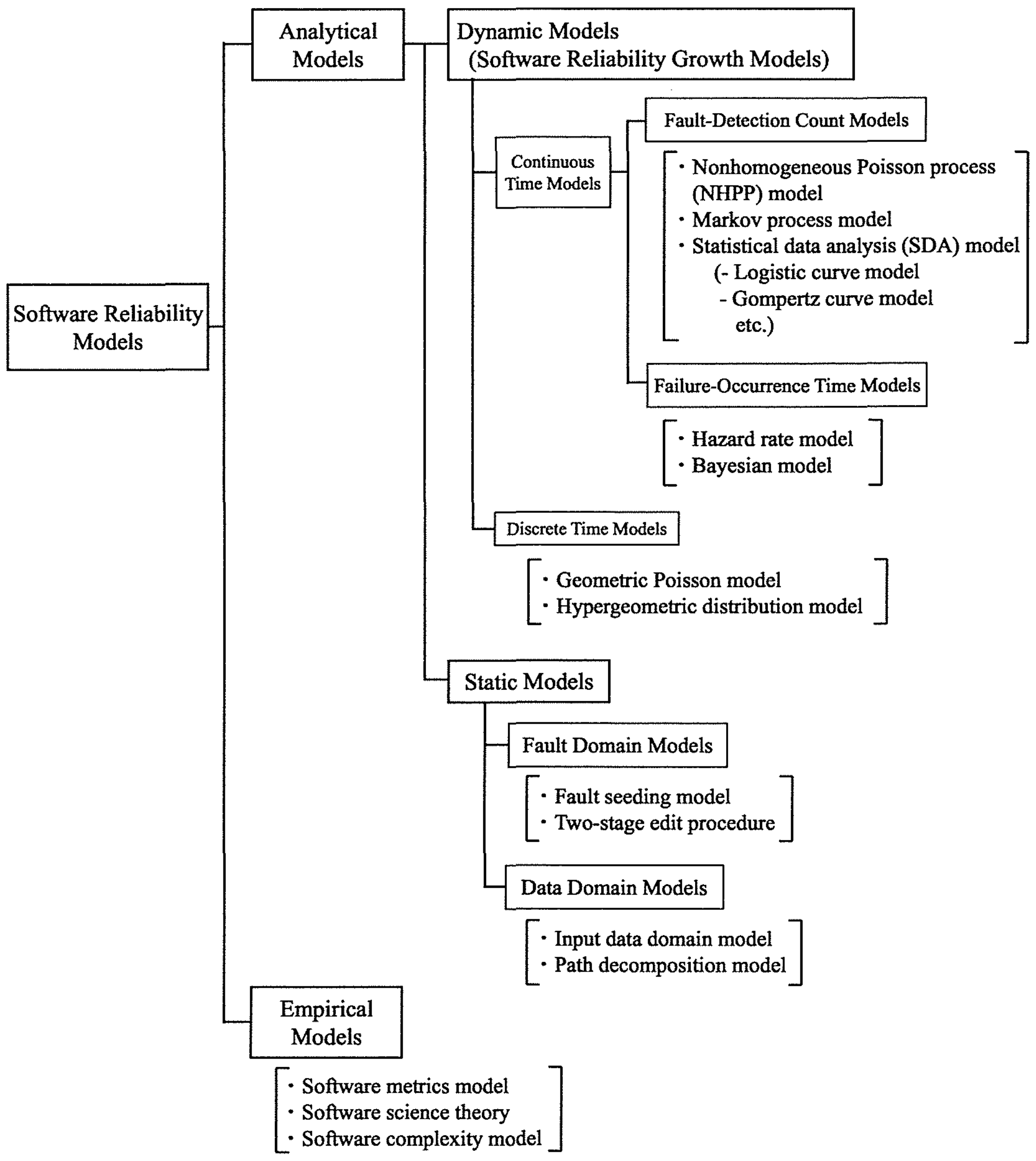

Software Reliability Models
Analytical Models
Dynamic Models (Software Reliability Growth Models)
Continuous Time Models
Fault-Detection Count Models
• Nonhomogeneous Poisson process (NHPP) model
• Markov process model
• Statistical data analysis (SDA) model (- Logistic curve model - Gompertz curve model etc.)
Failure-Occurrence Time Models
• Hazard rate model
• Bayesian model
Discrete Time Models
• Geometric Poisson model
• Hypergeometric distribution model
Static Models
Fault Domain Models
• Fault seeding model
• Two-stage edit procedure
Data Domain Models
• Input data domain model
• Path decomposition model
Empirical Models
• Software metrics model
• Software science theory
• Software complexity model

Fig. 1.5 : A hierarchical classification of software reliability models [55].

software failure rate, software reliability, and mean time between software failures, by developing a model under suitable assumptions about reliability factors in the testing and operational phases. The purpose of the empirical model is to analyze characteristic factors of software development process and software complexity by using empirical data. For examples, a McCabe complexity measure [38, 57] is a well-known model as one of the empirical models, which assesses the complexity of control flow graph of a program by using the cyclomatic number. Additionally, the analytical model is also classified into *dynamic model* and *static model.* The dynamic model is often called a *software reliability growth model* (abbreviated as *SRGM*). This model describes the software fault-detection or software failure-occurrence phenomenon during the testing and operational phases as the software reliability growth process by executing a implemented software [29, 51, 52, 56].

Many software reliability growth models have been developed over the past three decades. The existing models are classified from various view points by several researchers. Specifically, Yamada [52, 56] has classified the software reliability growth models into the following three categories:

- *Software failure-occurrence time model* :
  The stochastic model taking notice of the time-interval between software failure-occurrences.

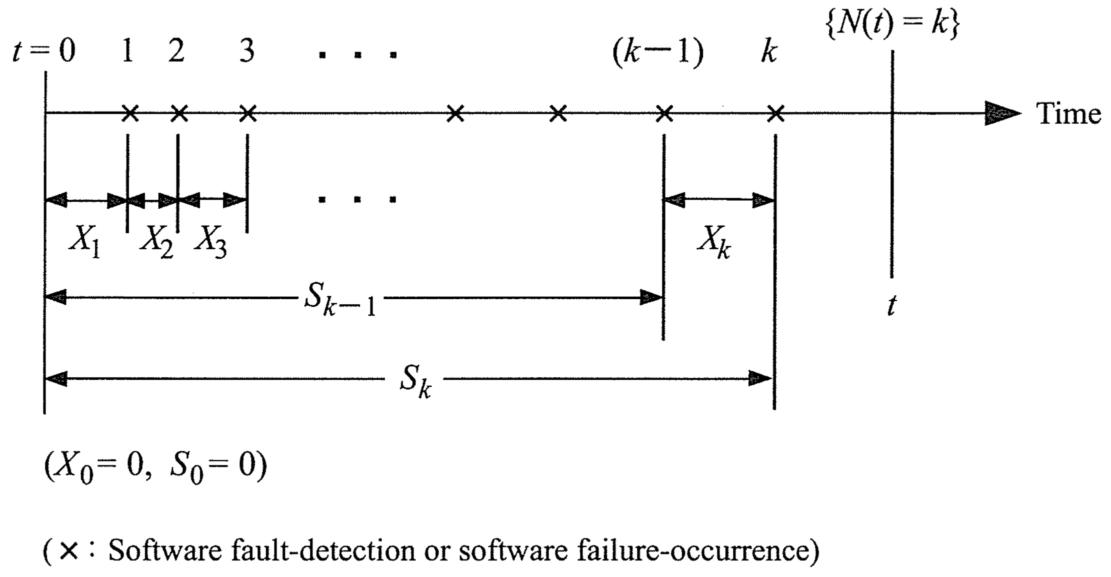
- *Software fault-detection count model* :
  The stochastic model taking notice of the cumulative number of faults detected or software failures occurring over a specified time period.

- *Software availability model* :
  The stochastic model describing the time-dependent behavior of a software system which alternates up and down states.

Especially, the software fault-detection count model based on a *nonhomogeneous Poisson process* (abbreviated as *NHPP*) is one of the well-known stochastic software reliability growth models. Several NHPP models have been practically utilized for assessing software reliability in many computer manufacturers and software houses in terms of the simplicity and applicability of the NHPP models. The main issue in the NHPP model is to estimate the mean value function indicating the expected cumulative number of faults detected up to a certain time point [38]. The NHPP models can be applied to several interesting issues, such as optimal software release [11,22,35,59], optimal testing-resource allocation [63], and software maintenance service contract problems [40].

1.4 Software Reliability Growth Modeling

This dissertation treats software reliability growth modeling which enables us to assess software reliability quantitatively. Then, we here discuss the concepts and some quantities for software reliability growth modeling.



{N(t) = k}
t = 0 1 2 3 . . . (k−1) k
Time
X1 X2 X3
. . .
Xk
S_{k−1}
t
S_k
(X0 = 0, S0 = 0)
(× : Software fault-detection or software failure-occurrence)

Fig. 1.6 : The stochastic quantities related to the software fault-detection and software-failure occurrence phenomena.

Through the testing-phase in the software development process, software faults are detected and removed with a lot of testing-effort expenditures. Then, the number of faults remaining in the software system is decreasing as the testing-time goes on. This means that the probability of software fault-detection or software failure-occurrence is decreasing so that the software reliability is increasing and the time-interval between software failures becomes longer with the testing-time. These phenomena can be modeled by using probability and statistical theories. That is, we can define the following random variables on the number of detected faults and the software failure-occurrence time (see Fig. 1.6):

$N(t)$ = the cumulative number of faults detected up to time t (or the cumulative number of 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 k faults have been detected up to time t. By using these definitions, we have the relationships as

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

Based on these quantities, we can derive several software reliability assessment measures by considering the probability distributions of $N(t)$, X_k, and S_k. The fault count data $(t_i, y_i)(i = 1, 2, \cdots ; 0 < t_1 < t_2 < \cdots)$ which mean that the number of faults, y_i, during the time-interval $(0, t_i]$ is the realization of the stochastic quantity $N(t)$. And the failure-occurrence time data $s_i(i = 1, 2, \cdots ; 0 \leq s_1 \leq s_2 \leq \cdots)$ are the realizations of stochactic quantities S_i.

As to software reliability measurement and assessment, the testing-time and operation time are based on the following units:

(1) the calender time,

(2) the execution time or CPU time,

(3) the testing-effort expenditures,

(4) the number of test runs or executed test cases,

(5) the ratio of testing-progress.

Ordinally, the continuous-time models assess software reliability measured on the basis of the units taking continuous values such as the execution/CPU time. On the other hands. the discrete-time models is based on the basis of the units taking discrete values such as the number of test runs.

1.5 Purposes and Organization of This Study

Measuring and assessing software reliability quantitatively and accurately in the testing-phase or operation phase are one of the important activities of project management to develop highly reliable software products. In the testing-phase software development managemers can get the situation of software reliability growth process by analyzing the observed data, such as the fault count data and software failure-occurrence time data. Applying an SRGM for assessing software reliability can also enable the software development managers to discuss several issues related to project management, such as optimal software release (or shipping) problems, optimal testing-resource allocation problems, and statistical software testing-progress control. Accordingly, SRGM's have been utilized in many actual software project as one of the fundamental technologies for quantitative assessment of software reliability.

During three decades, many SRGM's have been proposed in transactions and conferences on software engineering. However, unfortunately, very few of the SRGM's propopsed mainly in 1980's have been used in practical software reliability assessment. In recent years, as a role of computer systems is expanding rapidly, the size, complexity, and diversification of software systems are growing drastically. Furthermore, software development technologies have been also changing recently. Under the background above, many researchers on software reliability engineering have pointed out that there are limitations on the accuracy of assessment and estimation of software reliability. For the purpose of solving these problems above, this dissertation discusses several methods of improved estimation/prediction accuracy for software reliability assessment. Especially, several kinds of stochastic software reliability modelings, such as discretized software reliability modeling, software reliability modeling with serveral factors related to the software reliability growth process, generalized software reliability modeling, and software reliability modeling under imperfect debugging activities, which enable us to assess software reliability more accurately, are discussed. This dissertation is composed of the following four main parts.

In the first part consisted of Chapters 2 and 3 discretized software reliability modeling and its application to optimal software release problems are discussed. This part focuses on accuracy improvement of conventional continuous-time software reliability growth models by discretizing techniques called integrable difference methods and by transforming discrete deterministic models into stochastic models, respectively. Each content of these chapters in the first part is as follows:

- **Chapter 2**: *Software Reliability Modeling Based on Integrable Difference Equations*

  Chapter 2 discusses software reliability growth modeling based on integrable difference equations. Especially, discrete exponential and inflection S-shaped SRGM's which are derived from the original NHPP models by using Hirota's bilinearization methods are proposed, respectively. After that, goodness-of-fit comparisons of our discrete models with existing deterministic discrete models are performed by using actual data sets. Further, optimal software release problems under simultaneous cost and reliability requirements based on the proposed discrete SRGM's are discussed.

- **Chapter 3**: *Stochastic Software Reliability Modeling Based on Discretized SDA Models*

  Chapter 3 proposes stochastic discrete software reliability growth models by modifying discrete statistical data analysis (abbreviated as SDA) models which are deterministic models, such as discrete logistic, Gompertz, and modified exponential curve models. And we test whether proposed stochastic models have better performance than the ordinary discrete SDA models. Additionally, several software reliability assessment measures are derived based on the stochastic properties.

The second part discusses software reliability growth modeling with several factors related to the software reliability growth process in the testing-phase, such as testing-coverage and testing-effort expenditures. Taking these factors related to the software reliability growth process into consideration in software reliability modeling is very important to aim at accuracy improvement of quantitative software reliability assessment. The second part consists of Chapters 4 and 5 in the following:

- **Chapter 4**: *Software Reliability Modeling with Testing-Coverage*

  Chapter 4 discusses software reliability growth modeling with testing-coverage. The testing-coverage is one of the important metrics related to the software reliability growth process. Our testing-coverage dependent SRGM is developed by formulating the relationship between the testing-coverage maturity process and the software reliability growth process described by an NHPP. Based on our model, several testing-coverage dependent software reliability assessment measures are derived.

- **Chapter 5**: *Lognormal Process Software Reliability Modeling with Testing-Effort*

  Chapter 5 discusses a continuous-state space software reliability growth modeling with a testing-effort factor by applying a mathematical technique of stochastic differential equations of Itô type and its parameters estimation. And we show goodness-of-fit comparisons among our model and existing lognormal processes SRGM's by using actual data sets.

The third part discusses generalized software reliability growth modeling techniques. Generalization or unification frameworks of software reliability modeling are expected to enable software development managers to develop a plausible SRGM. The generalization frameworks in this dissertation mean the modeling frameworks which can develop several types of software reliability models according to the software failure-occurrence patterns in the actual testing-phase. In recent years, many researchers attempt to generalize or unify various SRGM's. In this part generalization approaches for continuous and discrete SRGM's are proposed, respectively. The third part consists of Chapters 6 and 7 which have the following contents:

- **Chapter 6**: *Generalized Discrete Software Reliability Modeling with Program Size*

  Chapter 6 discusses generalized discrete software reliability growth modeling in which the software failure-occurrence times obey a discrete probability distribution. Especially, our generalized discrete software reliability growth models enable us to assess software reliability in consideration of the effect of the program size. And several software reliability assessment measures based on the unified framework for discrete software reliability growth modeling are derived. Additionally, optimal software release problems based on our generalized discrete SRGM are also discusssed.

- **Chapter 7**: *Infinite Server Queueing Modeling for Software Reliability Assessment*

  Chapter 7 proposes an generalization framework for software reliability growth modeling by infinite server queueing theory. Our infinite server queueing model for software reliability assessment is developed by using the basic concept of the well-known delayed S-shaped software reliability growth modeling. Finally, the ralationship between our infinite server queueing model and existing NHPP models is discussed with the physical interpretation for the fault-detection phenomenon.

The fourth part treats imperfect debugging modeling for software reliability assessment. The imperfect debugging model is developed by assuming an imperfect debugging environment where latent faults are not always detected and corrected perfectly and the debugging activities may have a possibility that new faults are introduced. The imperfect debugging environment can be considered as a suitable assumption for an actual testing-phase. The fourth part discussed as Chapter 8 treats the following content:

- **Chapter 8**: *Software Reliability Modeling with Imperfect Debugging Activities*

  Chapter 8 proposes software reliability growth models which incorporate 2-types of imperfect debugging activities, such as the activities introducing new faults and the imperfect fault correction activities, simultaneously. Then, several software reliability assessment measures based on our models are derived. Further, the estimation methods of model parameters are discussed.

Each chapter provides numerical examples of software reliability analysis based on the software reliability models proposed in this dissertation by using actual fault data. The final chapter summarizes the results obtained in this dissertation and refers to the future researches on plausible software reliability modeling and its applications to software project management problems.

Part I

DISCRETIZED MODELING

Chapter 2

Software Reliability Modeling Based on Integrable Difference Equations

2.1 Introduction

An SRGM is one of the fundamental techniques to assess software reliability quantitatively. In particular, the SRGM based on an NHPP is known as a useful tool for assessing and predicting software reliability of the developed software system. At present, the NHPP models contribute to software reliability assessment in many computer manufactures and software houses. However, because the size, complexity, and diversification of software systems have grown drastically in recent years, software development managers require SRGM's which enable us to assess software reliability more accurately than conventional SRGM's proposed so far. As one of the efficient techniques to improve the performance of the SRGM's, discretized software reliability growth modeling techniques have been proposed. The discrete SRGM's can be derived by discretizing the original continuous-time SRGM's via using *Hirota's bilinearization methods* [14, 31]. Satoh [44] and Satoh and Yamada [47] have proposed a discrete Gompertz curve model and a software reliability assessment method for discrete logisitc curve models, respectively. They have performed goodness-of-fit comparisons among these discrete SDA models by using a new goodness-of-fit evaluation criterion [46].

In this chapter, discretized NHPP models, such as a discrete exponential SRGM and a discrete inflection S-shaped SRGM, derived by discretizing the original continuous-time NHPP models are discussed. Our discrete NHPP models discussed in this chapter are derived by special difference methods applying the Hirota's bilinearization methods to the basic assumptions of conventional continuous-time NHPP models. Generally, the ordinary forward or central difference equations do not have exact solutions, i.e., the difference equations do not conserve the properties of the continuous-time NHPP models. However, the discretization methods by using the Hirota's bilinearization methods can overcome the problems above. And, our discretized NHPP models can easily obtain the parameter estimates by using the method of least-squares.

2.2 Continuous-Time NHPP Models

In this section we discuss conventional continuous-time NHPP models, such as an exponential SRGM and an inflection S-shaped SRGM, which are treated in this chapter.

2.2.1 Exponential SRGM

An exponential SRGM [13] is based on the assumption that the number of faults detected at testing-time t is proportional to the current number of faults in a program. Let $H(t)$ denote the expected cumulative number of software faults detected up to arbitrary testing-time t from the test beginning. Then, we can obtain the following differential equation from the assumptions of the exponential SRGM:

$$\frac{dH(t)}{dt} = b\,[a - H(t)] \qquad (a > 0,\ b > 0)\,, \tag{2.1}$$

where a represents the expected total number of potential faults detected in an infinitely long duration or the expected initial fault content, and b the fault-detection rate per fault. Solving the differential equation in Eq. (2.1) with respect to $H(t)$, we can obtain the solution as

$$H(t) = a(1 - e^{-bt}). \tag{2.2}$$

This model is also known as the Goel-Okumoto model.

2.2.2 Inflection S-shaped SRGM

An inflection S-shaped SRGM [33] is based on the dependency of faults in terms of the software architecture and program path. Let $I(t)$ denote the expected cumulative number of software faults detected up to testing-time t from the test beginning. Then, we can obtain the following differential equation based on the assumptions of the inflection S-shaped SRGM:

$$\frac{dI(t)}{dt} = D\,(I(t))\,[a - I(t)], \tag{2.3}$$

where $D(I(t))$ represents the fault-detection rate per fault considering with fault-detection skill of the testing-team, which is formulated as

$$D(I(t)) = b\left[l + (1-l)\frac{I(t)}{a}\right] \qquad (a>0,\ b>0,\ c>0,\ 0 \le l \le 1). \tag{2.4}$$

In Eqs. (2.3) and (2.4), a represents the expected initial fault content, b the fault-detection rate per fault, and l the inflection rate which indicates the ratio of the number of detectable faults to the total number of faults in the program. Substituting Eq. (2.4) into Eq. (2.3), we can obtain the following solution by solving the differential equation with respect to $I(t)$:

$$I(t) = \frac{a\left(1-e^{-bt}\right)}{1+c\cdot e^{-bt}} \qquad (c>0), \tag{2.5}$$

where c represents the inflection parameter defined by $c = (1-l)/l$. The inflection point of the growth curve of the inflection S-shaped SRGM is derived as

$$t^* = \frac{\log c}{b}. \tag{2.6}$$

Then,

$$I(t^*) = \frac{a}{2}\left(1-\frac{1}{c}\right). \tag{2.7}$$

We can understand that the inflection point of the inflection S-shaped SRGM is depend on the inflection parameter c or l from Eq. (2.6). And, we can see that the inflection point does not exist when $l = 1$. That is, this model is a flexible SRGM which means the shape of the growth curve can be changed from an exponential curve into an S-shaped one as the value of the inflection parameter l takes $1 \to 0$.

2.3 Discrete NHPP Modeling

Now we assume that a discrete counting process $\{N_n, n \ge 0\}(n = 0,1,2,\cdots)$ representing the cumulative number of faults detected up to n-th testing-period has the following properties based on a continuous-time NHPP [36]:

$$\Pr\{N_n = x \mid N_0 = 0\} = \frac{\{\Lambda_n\}^x}{x!}\exp[-\Lambda_n] \qquad (n, x = 0,1,2,\cdots), \tag{2.8}$$

where $\Pr\{A\}$ means the probability of event A. Λ_n in Eq. (2.8) is a mean value function of the discrete counting process, which represents the expected cumulative number of faults detected up to n-th testing-period. The discrete counting process $\{N_n, n \ge 0\}(n = 0,1,2,\cdots)$ obeying the above stochastic properties is called a *discrete NHPP* in this chapter. In this section we develop two types of mean value functions of discrete NHPP's, i.e., we develop discrete exponential and inflection S-shaped SRGM's, respectively. And we also discuss parameter estimation methods for these discrete SRGM's, respectively.

2.3.1 Discrete exponential SRGM

We propose a discrete analog of the original exponential SRGM which is the simplest form among the SRGM's. Let H_n denote the expected cumulative number of faults detected up to n-th testing-period from the test beginning. Then, we derive a discrete exponential SRGM as

$$H_{n+1} - H_n = \delta b\,(a - H_n)\,, \tag{2.9}$$

from the assumptions of the continuous-time NHPP model in Eq. (2.1) by discretization methods based on the Hirota's bilinearization methods. Solving the above integrable difference equation, we can obtain an exact solution H_n in Eq. (2.9) as

$$H_n = a\left[1 - (1 - \delta b)^n\right] \qquad (a > 0,\ b > 0)\,, \tag{2.10}$$

where δ represents the constant time-interval, a the expected total number of potential faults to be detected in an infinitely long duration or the expected initial fault content, and b the fault detection rate per fault. As $\delta \to 0$, Eq. (2.10) converges to an exact solution of the original exponential SRGM which is described by the differential equation as Eq. (2.1).

The discrete exponential SRGM in Eq. (2.10) has two parameters, a and b, which have to be estimated the values by using an actual data. The parameter estimates $\widehat{a}$ and $\widehat{b}$ which are the estimated values of a and b can be obtained by the following procedure using the method of least-squares. First, we derive the following regression equation from Eq. (2.9):

$$Y_n = A + BH_n, \tag{2.11}$$

where

$$\begin{cases} Y_n = H_{n+1} - H_n \\ A \ = \delta ab \\ B \ = -\delta b. \end{cases} \tag{2.12}$$

Based on regression analysis, we can estimate $\widehat{A}$ and $\widehat{B}$ which are the estimates of A and B in Eq. (2.11). Then, the parameter estimates $\widehat{a}$ and $\widehat{b}$ can be obtained as

$$\begin{cases} \widehat{a} = -\widehat{A}/\widehat{B} \\ \widehat{b} = -\widehat{B}/\delta. \end{cases} \tag{2.13}$$

Y_n in Eq. (2.11) is independent of δ because δ is not used in calculating Y_n in Eq. (2.12). Hence, we can obtain the same parameter estimates $\widehat{a}$ and $\widehat{b}$, respectively, when we choose any value of δ.

2.3.2 Discrete inflection S-shaped SRGM

We also propose a discrete analog of the original inflection S-shaped SRGM. Let I_n denote the expected cumulative number of faults detected up to n-th testing-period from the test beginning. Then, we can derive a discrete inflection S-shaped SRGM as

$$I_{n+1} - I_n = \delta abl + \frac{\delta b(1-2l)}{2}\left[I_n + I_{n+1}\right] - \frac{\delta b(1-l)}{a} I_n I_{n+1}, \tag{2.14}$$

from the assumptions of the continuous-time NHPP model in Eq. (2.3) by using the Hirota's bilinearization methods. Solving the above integrable difference equation in Eq. (2.14) with respect to I_n, we can obtain an exact solution I_n as

$$I_n = \frac{a\left[1 - \left(\frac{1-\frac{1}{2}\delta b}{1+\frac{1}{2}\delta b}\right)^n\right]}{1 + c\left(\frac{1-\frac{1}{2}\delta b}{1+\frac{1}{2}\delta b}\right)^n} \qquad (a > 0,\ b > 0,\ c > 0,\ 0 \le l \le 1)\,, \tag{2.15}$$

where δ represents the constant time-interval, a the expected total number of potential faults to be detected in an infinitely long duration or the expected initial fault content, b the fault detection rate per fault, and c the inflection parameter. The inflection parameter is defined as $c = (1-l)/l$ where l is the inflection rate which indicates the ratio of the number of detectable faults to the total number of faults in the software system. As $\delta \to 0$, Eq. (2.15) converges to an exact solution of the original inflection S-shaped SRGM which is described by the differential equation as Eq. (2.3).

The inflection point can be derived as the following. Defining the difference operator as

$$\Delta I_n \equiv \frac{I_{n+1} - I_n}{\delta}, \tag{2.16}$$

we show that the inflection point occurs when

$$\bar{n} = \begin{cases} [\, n' \,] & (\text{if } \Delta I_{[\, n' \,]} \ge \Delta I_{[\, n' \,]+1}) \\ [\, n' \,] + 1 & (\text{otherwise}), \end{cases} \tag{2.17}$$

where $[\, x \,]$ represents the Gaussian symbol for any real number x, and

$$n' \;=\; -\frac{\log c}{\log\left(\frac{1-\frac{1}{2}\delta b}{1+\frac{1}{2}\delta b}\right)} - 1. \tag{2.18}$$

Moreover, we define t^{**} as

$$t^{**} = n'\delta. \tag{2.19}$$

When n' is an integer, we can show that t^{**} converges the inflection point of the original inflection S-shaped SRGM as $\delta \to 0$ as follows:

$$t^{**} = -\delta \frac{\log c}{\log\left(\frac{1-\frac{1}{2}\delta b}{1+\frac{1}{2}\delta b}\right)} - \delta \quad \to \quad \frac{\log c}{b} \quad \text{as } \delta \to 0. \tag{2.20}$$

By the way, the inflection S-shaped SRGM is regarded as a Riccati equation. Hirota [14] has proposed a discrete Riccati equation which has an exact solution. A Bass model [2] which forecasts the innovation diffusion of products is also a Riccati equation. Satoh [45] has proposed a discrete Bass model which can overcome the shortcomings of the ordinary least-square procedures in the continuous-time Bass model.

The discrete inflection S-shaped SRGM has the three parameters, a, b, and l. These parameter can be estimated by the following estimation procedure. First, we derive a regression equation to estimate the model parameters from Eq. (2.14). The regression equation is obtained as

$$Y_n = A + BK_n + CL_n, \tag{2.21}$$

where

$$\begin{cases} Y_n &= I_{n+1} - I_n \\ K_n &= I_n + I_{n+1} \\ L_n &= I_n I_{n+1} \\ A &= \delta abl \\ B &= \delta b\left(1-2l\right)/2 \\ C &= -\delta b\left(1-l\right)/a. \end{cases} \tag{2.22}$$

Based on regression analysis, we can obtain $\widehat{A}$, $\widehat{B}$, and $\widehat{C}$ which are the estimates of A, B, and C, respectively. Therefore, we can obtain $\widehat{a}$, $\widehat{b}$, and $\widehat{l}$ which are the parameter estimates of a, b, and l from Eq. (2.22), respectively, as follows:

$$\begin{cases} \widehat{a} = \widehat{A}/\left(\sqrt{\widehat{B}^2 - \widehat{A}\widehat{C}} - \widehat{B}\right) \\ \widehat{b} = 2\sqrt{\widehat{B}^2 - \widehat{A}\widehat{C}}/\delta \\ \widehat{l} = \left(1 - \widehat{B}/\sqrt{\widehat{B}^2 - \widehat{A}\widehat{C}}\right)/2. \end{cases} \tag{2.23}$$

Y_n, K_n, and L_n in Eq. (2.21) are independent of δ because δ is not used in calculating Y_n, K_n, and L_n in Eq. (2.21). Hence, we can obtain the same parameter estimates $\widehat{a}$, $\widehat{b}$, and $\widehat{l}$, respectively, when we choose any value of δ.

2.4 Model Comparisons

We perform goodness-of-fit comparisons of our discrete NHPP models with the discrete SDA models which are logistic and Gompertz curve models [44, 46, 47]. First, we arrange the following four data sets used in the model comparisons:

- DS1 [52] : $(t_k, y_k)(k = 1, 2, \cdots, 25\ ;\ t_{25} = 25, y_{25} = 136)$ where t_k is measured on the basis of CPU hours,
- DS2 [6] : $(t_k, y_k)(k = 1, 2, \cdots, 12\ ;\ t_{12} = 12, y_{12} = 2657)$ where t_k is measured on the basis of months,
- DS3 [44] : $(t_k, y_k)(k = 1, 2, \cdots, 59\ ;\ t_{59} = 59, y_{59} = 5186)$ where t_k is measured on the basis of weeks,
- DS4 [6] : $(t_k, y_k)(k = 1, 2, \cdots, 35\ ;\ t_{35} = 35, y_{35} = 1301)$ where t_k is measured on the basis of month,

where y_k represents the cumulative number of faults detected up to testing-time t_k. The data sets of DS1 and DS2 indicate exponential growth curves, and those of DS3 and DS4 indicate S-shaped growth curves, respectively.

2.4.1 Comparison criteria

We conduct goodness-of-fit comparisons in terms of a *predicted relative error* [52], a *mean square error* (abbreviated as *MSE*) [38, 52, 55], and *Akaike's information criterion* (abbreviated as *AIC*) [1, 43].

The predicted relative error is a useful criterion for indicating the relative errors between the predicted number of faults detected up to the termination time of testing by using the part of observed data from the test beginning and the observed number of faults detected up to the termination time. Let $R_e[t_e]$ denote the predicted relative error at arbitrary testing-time t_e. Then, the predicted relative error is given by

$$R_e[t_e] = \frac{\widehat{y}(t_e; t_q) - q}{q}, \tag{2.24}$$

where $\widehat{y}(t_e; t_q)$ is the estimated value of the mean value function at the termination time t_q by using the observed data by the arbitrary testing-time t_e $(0 \leq t_e \leq t_q)$, and q the observed cumulative number of faults detected by the termination time.

The value of MSE is calculated by dividing the sum of squared vertical distance between the observed and estimated cumulative numbers of faults, y_k and $\widehat{y}\,(t_k)$, 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

$$\mathrm{MSE} = \frac{1}{K}\sum_{k=1}^{K}\left[y_k - \widehat{y}\left(t_k\right)\right]^2, \tag{2.25}$$

where $\widehat{y}\left(t_k\right)$ denotes the estimated value of the expected cumulative number of faults by using exact solutions of each model by arbitrary testing-time $t_k (k = 1, 2, \cdots, K)$. The model having the smallest value of the MSE fits best to the observed data set.

And, the AIC enables us to select the optimal model among ones estimated by the method of maximum-likelohood. The AIC is known as a goodness-of-fit evaluation criterion considering with the number of model parameters. The value of AIC is calculated by

$$\begin{aligned} \mathrm{AIC} = -2 \times (&\text{the logarithmic maximum-likelihood}) \\ &+2 \times (\text{the number of free model parameters}). \end{aligned} \tag{2.26}$$

Note that the value of AIC themselves are not significant. The absolute value of difference among their values are significant. We can judge that the model having the smallest value of AIC fits best to the actual data set when their differences are greater than or equal to 1. When their differences are less than 1, there are no significant.

2.4.2 Results of model comparison

First, Figs. 2.1–2.5 shows the results of model comparisons based on the predicted relative error for DS1, DS2, DS3, and DS4, respectively. From Figs. 2.1 and 2.2, we can see that the discrete exponential SRGM can accurately predict the cumulative number of faults detected up to the termination time throughout all the testing-time. And, from Figs. 2.3 and 2.4, we can see that the discrete inflection S-shaped SRGM does not always predict accurately the cumulative number of faults detected up to the termination time in the early testing-time. However, the discrete inflection S-shaped SRGM can predict more accurately than any other SRGM used in these model comparisons as the testing-time goes on. Considering that practical activities on software reliability assessment is started after 60% of the testing progress [17], we can say that the discrete inflection S-shaped SRGM is superior in the prediction of faults detected up to the termination time of the testing after 60% of the testing-progress.

Table 2.1 shows the result of model comparisons based on the MSE for the each model. From Table 2.1, we can see that the discrete inflection S-shaped SRGM fits better to all data sets except for DS2. In these model comparisons based on the MSE, the results depend on the

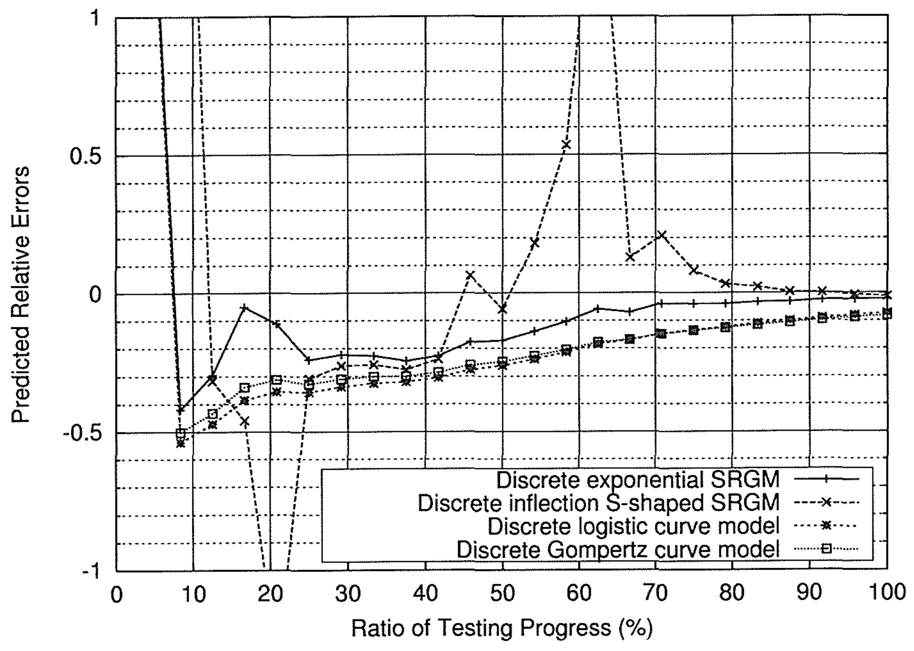


Fig. 2.1 : The predicted relative errors for DS1.

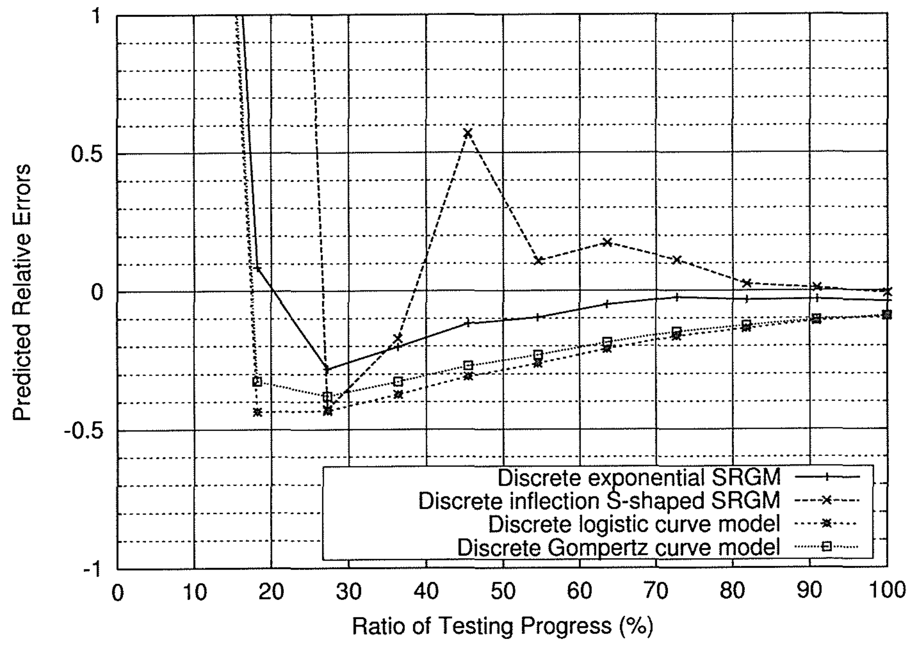


Fig. 2.2 : The predicted relative errors for DS2.

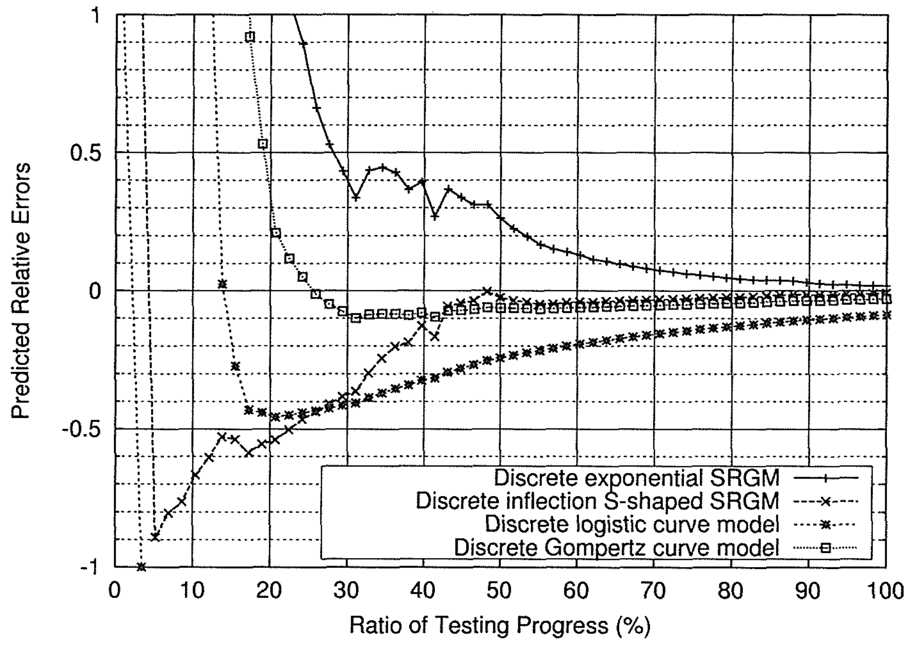


Fig. 2.3 : The predicted relative errors for DS3.

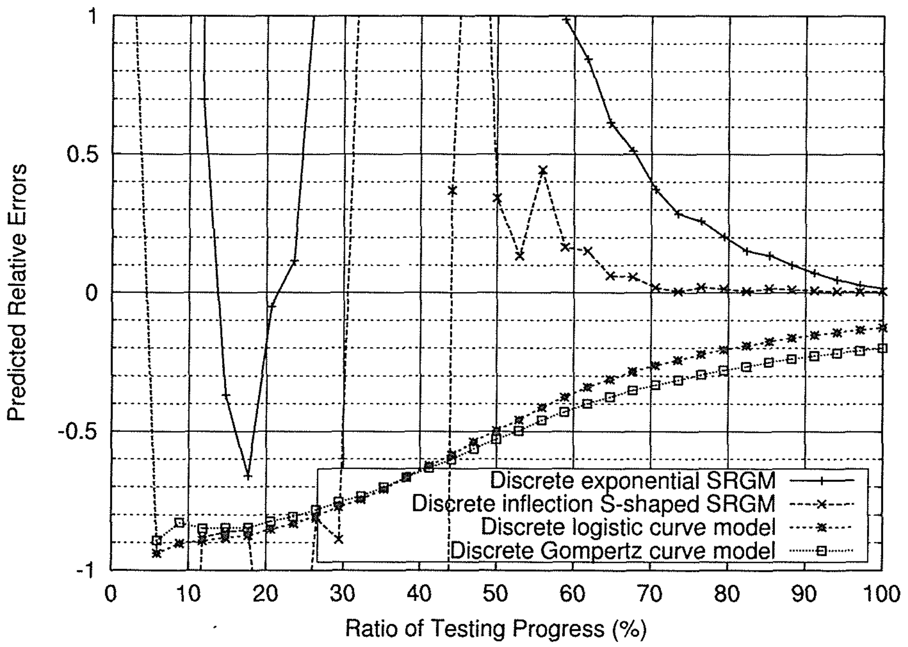


Fig. 2.4 : The predicted relative errors for DS4.

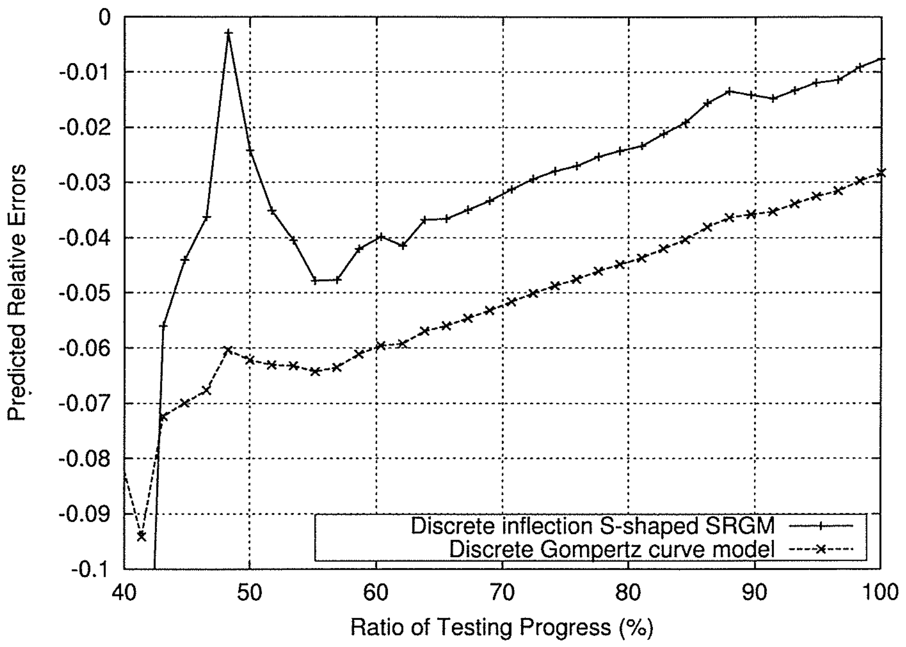


Fig. 2.5 : The model comparison between the discrete Gompertz curve and the discrete inflection S-shaped SRGM based on the predicted relative error for DS3.

Table 2.1 : The result of model comparisons based on the MSE.

Data Set	discrete exponential SRGM	discrete inflection S-shaped SRGM	discrete logistic curve model	discrete Gompertz curve model
DS1	39.643	**12.141**	101.92	72.854
DS2	**1762.5**	2484.0	27961	13899
DS3	25631	**9598.1**	149441	19579
DS4	11722	**438.59**	49741	27312

(SRGM : Software Reliability Growth Model)

Table 2.2 : The result of model comparisons between the discrete exponential and the discrete inflection S-shaped SRGM's based on the AIC.

Data Set	discrete exponential SRGM	discrete inflection S-shaped SRGM	absolute value of difference
DS1	110.031	109.195	0.836
DS2	**115.735**	118.752	3.017
DS3	617.434	**606.132**	11.30
DS4	315.069	**274.818**	40.25

number of model parameters of each model. Accordingly, we provide Table 2.2 which shows the result of model comparisons based on the AIC for the discrete exponential having two

Table 2.3 : The estimated parameters of $\widehat{H_n}$ for DS1 and $\widehat{I_n}$ for DS3.

	$\widehat{a}$	$\widehat{b}\ (\delta = 1)$	$\widehat{c}$	n'	$[n']$	$\bar{n}$
H_n	139.956	0.113	—	—	—	—
I_n	5217.88	0.0906	2.350	8.383	8	9

parameters and inflection S-shaped SRGM's having three parameters which fit better to the actual data sets than the others. From Table 2.2, the model comparison based on the MSE can be validated.

From these three results of goodness-of-fit comparisons, we can conclude that the discrete exponential SRGM is more useful model for software reliability assessment for the observed data which indicates an exponential growth curve, and the discrete inflection S-shaped SRGM is more useful one for the assessment after 60% of the testing-progress ratio for the observed data which indicates an S-shaped growth curve.

2.5 Software Reliability Assessment Measures

In this section we derive several software reliability measures for our proposed models, such as expected number of remaining faults, fault detection rates, software reliability functions, and reliability growth rates, which are useful for assessing software reliability quantitatively. We adopt DS1 and DS3 for the discrete exponential SRGM and the discrete inflection S-shaped SRGM, respectively. First, Table 2.3 shows the obtained parameter estimates and the related quantities of these models. By using these parameter estimates, Figs. 2.6 and 2.7 depict the estimated mean value functions of H_n in Eq. (2.10) and I_n in Eq. (2.15), respectively.

2.5.1 Expected number of remaining faults

The expected number of remaining faults indicate the expected number of undetected faults in the software system by arbitrary testing-period from the test beginning. Then, the expected number of remaining faults at n-th testing-period, M_n, is formulated as

$$\begin{aligned} M_n &\equiv \mathrm{E}\left[\bar{N}_n\right] \\ &= \mathrm{E}\left[N_\infty - N_n\right] \\ &= a - \Lambda_n, \end{aligned} \tag{2.27}$$

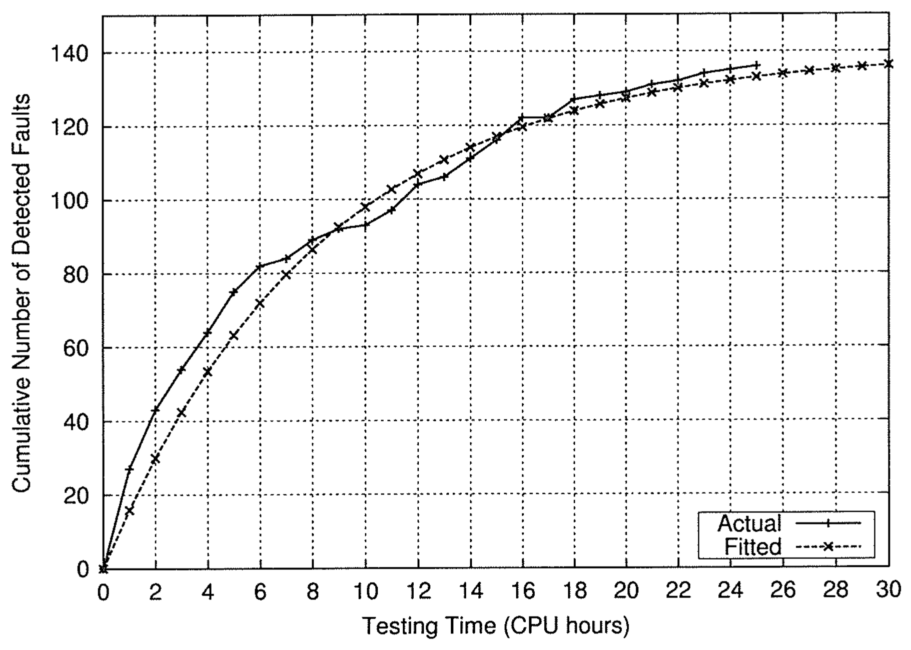


Fig. 2.6 : The estimated discrete mean value function, $\widehat{H}_n$, for DS1.

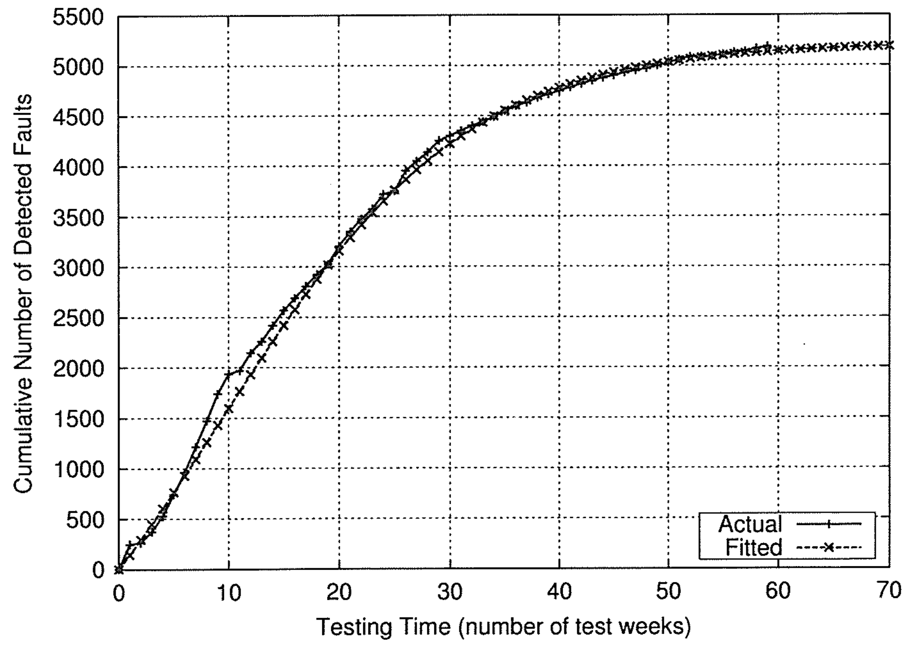


Fig. 2.7 : The estimated discrete mean value function, $\widehat{I}_n$, for DS3.

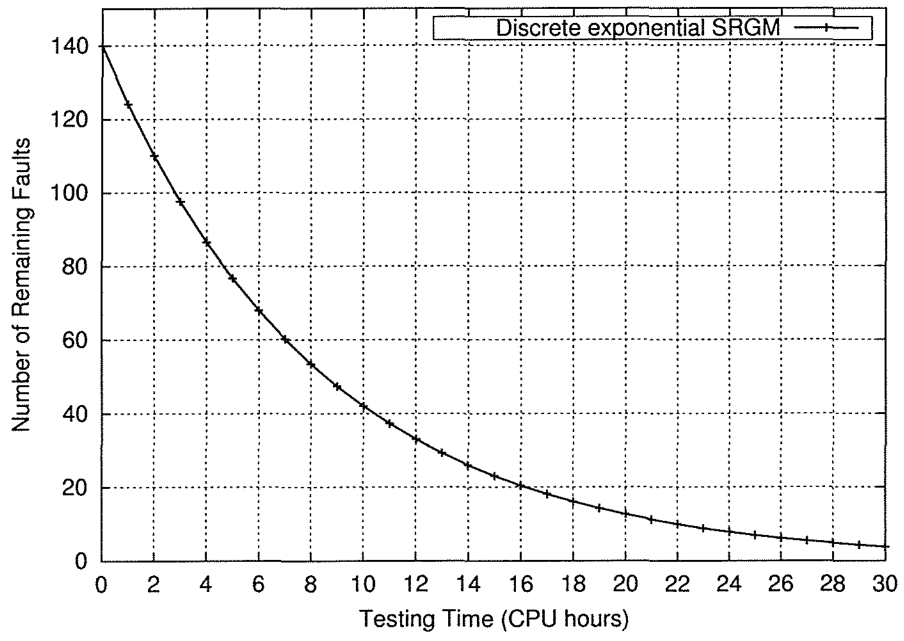


Fig. 2.8 : The estimated expected number of remaining faults, $\widehat{M_n^e}$, for DS1.

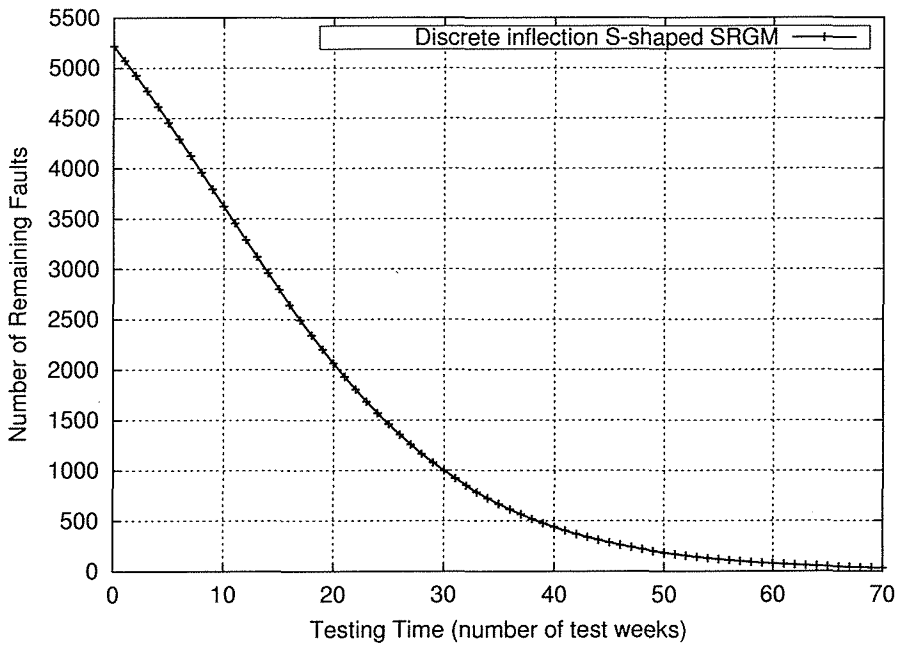


Fig. 2.9 : The estimated expected number of remaining faults, $\widehat{M_n^i}$, for DS3.

where E[·] means the expectation. Figs. 2.8 and 2.9 show the estimated expected number of remaining faults based on the discrete exponential and the discrete inflection S-shaped SRGM's, $\widehat{M_n^e}$ and $\widehat{M_n^i}$, for DS1 and DS3, respectively. From Figs. 2.8 and 2.9, we can estimate the expected number of remaining faults $\widehat{M_{24}^e}$ for DS1 to be about 8 faults, and $\widehat{M_{58}^i}$ for DS3 to be about 90 faults.

2.5.2 Fault detection rate

We discuss *fault detection rates* as new simple measures for quantitative software reliability assessment. The expected number of faults detected during each period is derived by differing the discrete mean value function as $\delta = 1$, which is substitute for the intensity function of the continuous-time NHPP model. However, this is not enough to use as a useful assessment measure because it gives only the number of faults detected during each testing-period to the software development managers. It is important to know how much faults are detected during each testing-period for the initial fault content. Accordingly, we introduce a new reliability assessment measure which represents the ratio of the number of faults detected during each period to the estimated expected initial fault content as follows:

$$D_n \equiv (\Lambda_n - \Lambda_{n-1})/\widehat{a}. \tag{2.28}$$

Figs. 2.10 and 2.11 show the estimated fault detection rate for DS1 and DS3, respectively. From these figures, we can estimate the fault detection rate D_{25} for DS1 to be about $6 \cdot 321 \times 10^{-3}$. And we can also estimate D_{59} for DS3 to be about $1 \cdot 477 \times 10^{-3}$.

2.5.3 Software reliability function

We derive *software reliability functions* which are also ones of the useful software reliability assessment measures. The software reliability function is derived as

$$\begin{aligned} R(n,h) &\equiv \Pr\{N_{n+h} - N_n = 0 | N_n = x\} \\ &= \exp[-\{\Lambda_{n+h} - \Lambda_n\}]. \end{aligned} \tag{2.29}$$

by using the properties of the discrete NHPP in Eq. (2.8). Now we suppose $\delta = 1$. Then, the estimated software reliability functions for H_n after the testing termination time $n = 25$ (CPU), $\widehat{R}_e(25,h)$, and for I_n after the testing termination time $n = 59$ (weeks), $\widehat{R}_i(25,h)$, are shown in Figs. 2.12 and 2.13, respectively. Assuming that the software users operate these software under the same environment as the software testing, we can estimate the software reliability

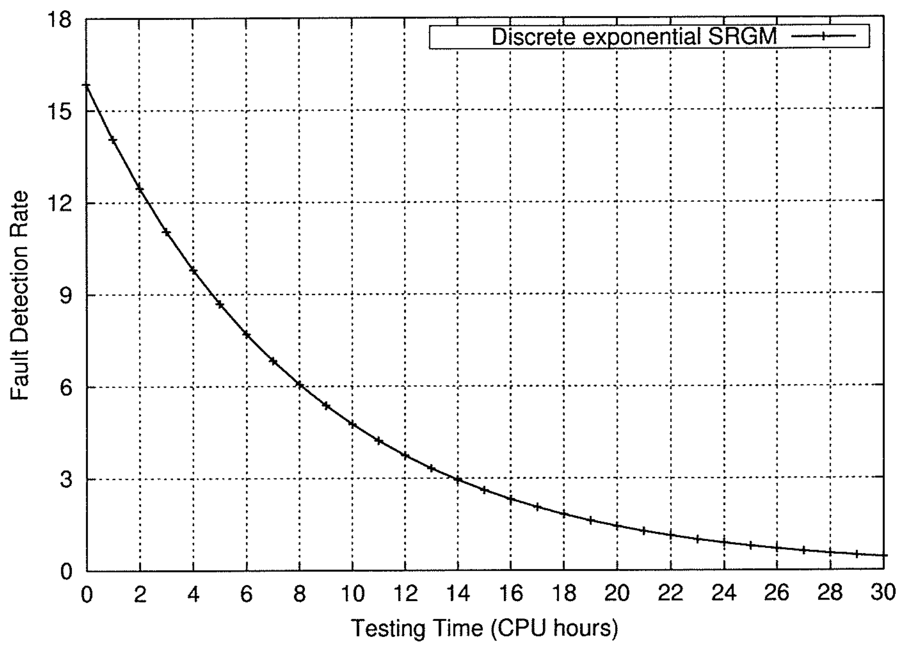


Fig. 2.10 : The estimated fault detection rate, $\widehat{D}_n^e$, for DS1.

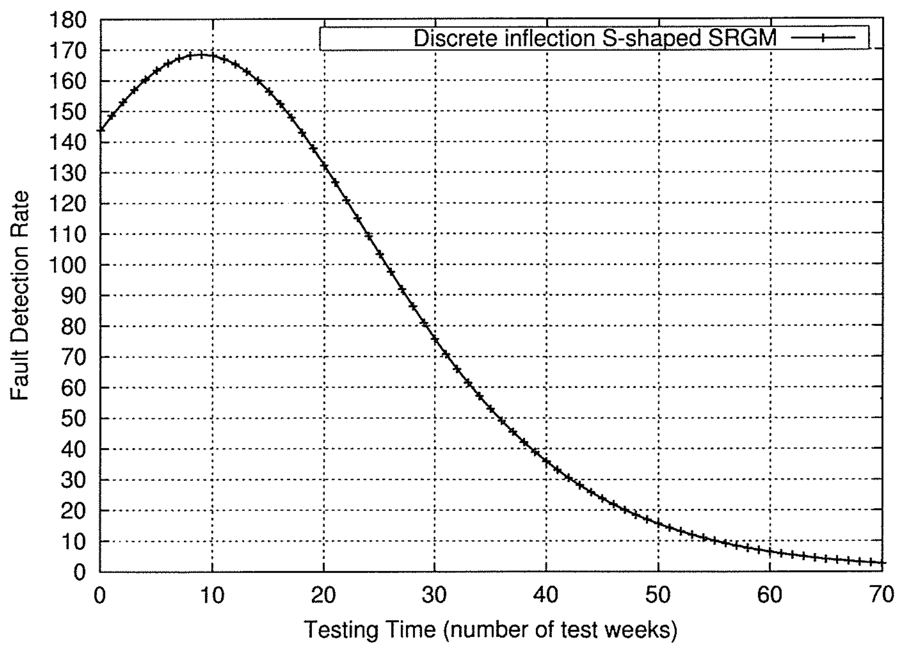


Fig. 2.11 : The estimated fault detection rate, $\widehat{D}_n^i$, for DS1.

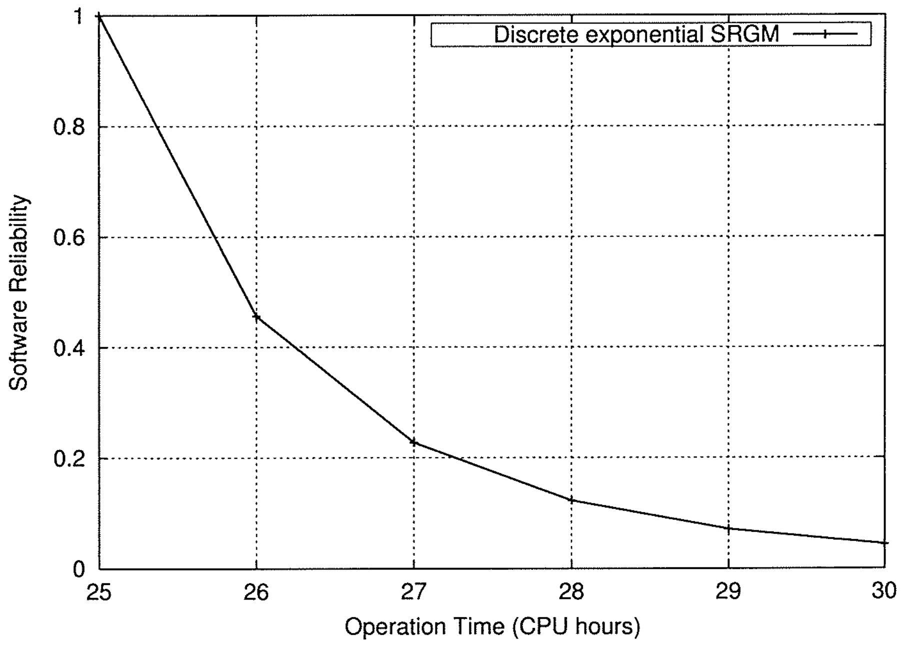


Fig. 2.12 : The estimated software reliability function, $\widehat{R}_e(25, h)$, for DS1.

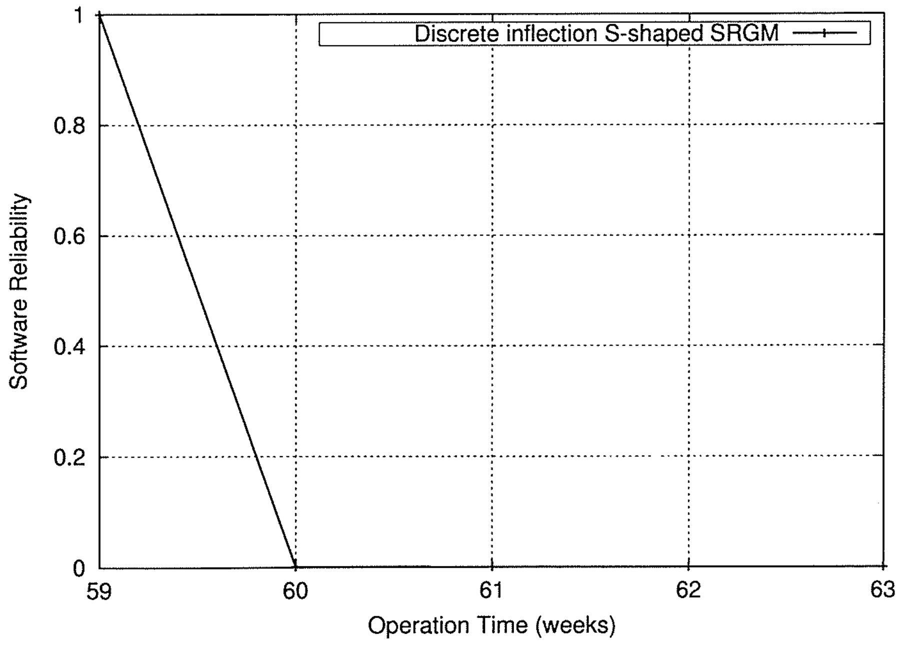


Fig. 2.13 : The estimated software reliability function, $\widehat{R}_i(59, h)$, for DS3.

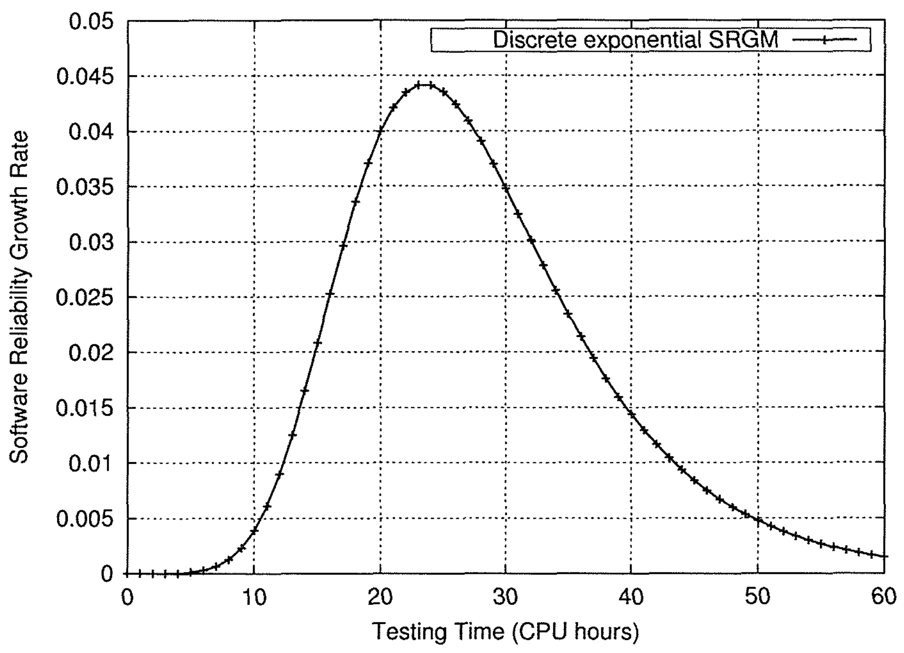


Fig. 2.14 : The estimated software reliability growth rate, $\widehat{r}_e(n, 1)$, for DS1.

$\widehat{R}_e(25, 1.0)$ for the discrete exponential SRGM to be about 0·46. And we can also estimate one $\widehat{R}_i(59, 0.1)$ for the inflection S-shaped SRGM to be about 0·48.

2.5.4 Reliability growth rate

In addition to the software reliability function in Eq. (2.29), we consider *software reliability growth rates* which can reflect the influence of debugging-effort on software reliability during each testing-period. The software reliability growth rate is formulated as

$$r(n, h) \equiv R(n, h) - R(n-1, h). \tag{2.30}$$

by using Eq. (2.29). Suppose that $h = 1$. Then, the estimated software reliability growth rates for H_n (DS1) and for I_n (DS3), $r_e(n, 1)$ and $r_i(n, 1)$, are shown in Figs. 2.14 and 2.15, respectively.

In Fig. 2.14, for given $h = 1$, we can see that the software reliability growth rate per period increases by around $n = 25$. We can estimate the software reliability growth rate at the testing termination time of DS1 to be about $r(25, 1) \approx 4 \cdot 353 \times 10^{-2}$, i.e., it is gained software reliability, $r(25, 1) \approx 4 \cdot 353 \times 10^{-2}$, during the period between 24-th and 25-th testing. In Fig. 2.15, we can also see that the testing by about 40-th period does not contribute to the software reliability growth. Therefore, we can say that it is neccessary to test more after around 40-th

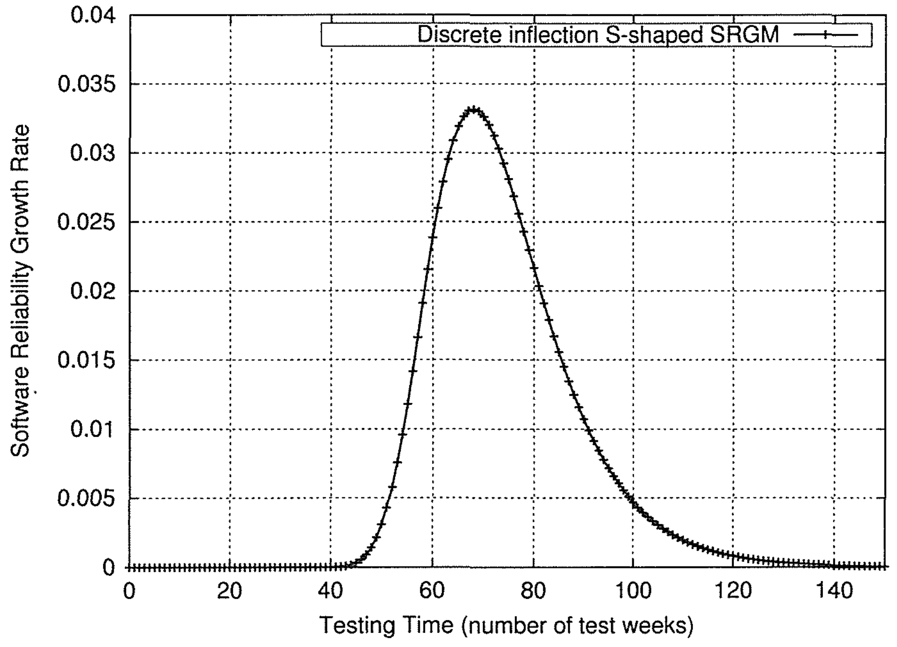


Fig. 2.15 : The estimated software reliability growth rate, $\widehat{r_i}(n, 1)$, for DS3.

testing-period. We can also estimate it at the termination time of the testing of DS3 to be about $r(59, 1) \approx 2 \cdot 155 \times 10^{-2}$.

2.6 Optimal Software Release Problems

The costs of developing a software system entail on great expenses. Especially, it is reported that the cost of the testing-phase is expended more than 40% of software developing costs [5]. Accordingly, software developing managers have a great interest how to develop a reliable software system economically. Generally, the longer testing is expected to attain a more reliable software system. However, the total cost of testing-phase is expected to be more increasing, and the testing is not always finished by the specified day of delivery. On the other hand, if the total testing-time is too short, the testing cost is reduced, but the customers' risk is caused by operating the unreliable software system. That is, there is the trade-off relationship to derive the optimal time to release the software from the testing-phase to the operational phase.

Problems determining when to stop the testing or release to the user in consideration of the software reliability, the related costs, and the delivery have been called *optimal sofware release problems.* Up to now, based on several SRGM's, optimal software release policies based on the total software cost have been discussed by Foreman and Singpurwalla [11], Okumoto and Goel [35], Yamada and Osaki [59], and other many researchers. Additionally, in recent years, Kimura et al. [22] have discussed optimal software release policies which consider both the present value of the total software cost and the warranty period in which the developer has to pay the cost for fixing any detected faults.

In this section we discuss discrete optimal software release policies based on the discrete NHPP models proposed in this chapter. That is, we derive the optimal policies to estimate a termination time of testing which minimizes the total expected software cost. And then, we also discuss discrete optimal software release policies with simultaneous cost and reliability requirements from the software quality control point of view. Finally, applying fault count data observed in practical testing-phases, we show numerical examples for the derived optimal software release policies.

2.6.1 Cost-optimal software release policies

We discuss cost-optimal software release policies based on the discrete NHPP models which have the discrete mean value functions denoted by Λ_n generally. The following notations are defined:

c_1 : debugging cost per one fault in the testing-phase,

c_2 : debugging cost per one fualt in the operational phase, where $c_1 < c_2$,

c_3 : testing cost per constant period.

First, let Z be the software release period. Then, the expected cost for debugging faults detected in the testing-phase is derived as $c_1\Lambda_Z$, the expected cost for debugging faults detected in the operational phase as $c_2(a-\Lambda_Z)$, and the testing cost by Z-th period as c_3Z. Accordingly, the expected total software cost C_Z which indicates the expected total cost during the testing-phase and the operational phase can be formulated as

$$C_Z = c_1\Lambda_Z + c_2(a - \Lambda_Z) + c_3 Z. \tag{2.31}$$

The cost-optimal software release period is the testing-period which minimizes the total expected software cost C_Z in Eq. (2.31). From Eq. (2.31), we can derive the following equation:

$$\frac{C_{Z+1} - C_Z}{\delta} = \frac{c_2 - c_1}{\delta}\left[\frac{c_3}{c_2 - c_1} - W_Z\right], \tag{2.32}$$

where W_Z represents the expected number of faults detected during a Z-th period. Additionally, we define the following notation:

$$< n > \; = \begin{cases} [\, n\,] & (\text{if } C_{[n]} \le C_{[n]+1}) \\ [\, n\,] + 1 & (\text{otherwise}), \end{cases} \tag{2.33}$$

where $[\, n\,]$ represents the Gaussian symbol for any real number n.

When the discrete exponential SRGM in Eq. (2.10) is applied to Eq. (2.31), W_Z^e which represents W_Z in Eq. (2.32) for the discrete exponential SRGM is derived as

$$W_Z^e = \delta ab(1 - \delta b)^Z. \tag{2.34}$$

From Eq. (2.34), we can confirm the following properties:

$$W_{Z+1}^e < W_Z^e, \qquad W_0^e = \delta ab, \qquad W_\infty^e = 0. \tag{2.35}$$

That is, W_Z^e in Eq. (2.34) is a monotonically decreasing function in terms of the testing-period Z. Therefore, we obtain cost-optimal release policies as follows:

Optimal Release Policy 1-1

Suppose that $c_2 > c_1 > 0$ and $c_3 > 0$.

(1) If $W_0^e \le \frac{c_3}{c_2 - c_1}$, then the optimal software release period is $Z^* = 0$.

(2) If $W_0^e > \frac{c_3}{c_2 - c_1}$, then we have the following an only solution $Z = Z_0$ minimizing Eq. (2.31):

$$Z_0 = \frac{\ln\left[\frac{c_3}{(c_2 - c_1)\delta ab}\right]}{\ln(1 - \delta b)}. \tag{2.36}$$

Thus, the optimal software release period is $Z^* =< Z_0 >$.

Next, we discuss optimal software release policies based on the discrete inflection S-shaped SRGM in Eq. (2.15). From Eq. (2.15), the expected number of faults detected during a Z-th testing-period W_Z^I in Eq. (2.32) has a maximum value at the following inflection point $\bar{Z}$ for the discrete inflection S-shaped SRGM:

$$\bar{Z} = \begin{cases} [\, Z'\,] & (\text{if } \Delta I_{[\, Z'\,]} \ge \Delta I_{[\, Z'\,]+1}) \\ [\, Z'\,] + 1 & (\text{otherwise}), \end{cases} \tag{2.37}$$

where

$$Z' \; = \; -\frac{\log c}{\log\left(\frac{1 - \frac{1}{2}\delta b}{1 + \frac{1}{2}\delta b}\right)} - 1. \tag{2.38}$$

ΔW_Z^I representing the forward difference of W_Z^I is confirmed that $\Delta W_0^I > 0$ and $\Delta W_\infty^I \le 0$. Therefore, we obtain the optimal release policies based on the discrete inflection S-shaped SRGM as follows:

Optimal Release Policy 1-2

Suppose that $c_2 > c_1 > 0$ and $c_3 > 0$.

(1) If $W_Z^I \le \frac{c_3}{c_2 - c_1}$, then the optimal software release period Z^* is $Z^* = 0$.

(2) If $W_Z^I > \frac{c_3}{c_2 - c_1} > W_0^I$, then we have an only solution Z_0 satisfying $W_Z^I \ge \frac{c_3}{c_2 - c_1}$ and $W_{Z+1}^I < \frac{c_3}{c_2 - c_1}$. When the C_0 is $C_0 < C_{<Z_0>}$, the optimal software release period Z^* is $Z^* = 0$. When the C_0 is $C_0 \ge C_{<Z_0>}$, Z^* is $Z^* =< Z_0 >$.

(3) If $W_0^I > \frac{c_3}{c_2 - c_1}$, then we have an only solution Z_0 satisfying $W_Z^I \ge \frac{c_3}{c_2 - c_1}$ and $W_{Z+1}^I < \frac{c_3}{c_2 - c_1}$. Thus, the optimal software release period Z^* is $Z^* =< Z_0 >$.

2.6.2 Cost-reliability-optimal software release policies

Additionally, we discuss optimal software release problems which take both total software cost and reliability criteria into consideration simultaneously. In an actual software development, the software developing manager has to spend and control the testing resources minimizing the total software cost and satisfying the software reliability requirement rather than only minimizing the cost.

Now, let R_0 $(0 < R_0 \le 1)$ be a software reliability objective. Using the software reliability function in Eq. (2.29), we can derive optimal software release policies which minimize the expected total software cost in Eq. (2.31) with satisfying the software reliability objective R_0. Thus, the optimal software release problem can be formulated as follows:

$$\left.\begin{array}{l} \text{Minimize } C(Z) \\ \text{subject to } R(Z,h) \ge R_0,\ Z \ge 0 \end{array}\right\}. \tag{2.39}$$

Supposing h is a constant value, we can obtain the following properties as to the discrete software reliability function in Eq. (2.29):

$$R(Z+1,h) > R(Z,h), \qquad R(0,h) = \exp[-\Lambda_h], \qquad R(\infty,h) = 1. \tag{2.40}$$

Therefore, we can see that the discrete software reliability function, $R(Z,h)$, is a monotonically increasing function in terms of the testing-period Z when we suppose that h is a constant value. Accordingly, if $R(0,h) < R_0$, then we have only finite solution Z_1 satisfying $R(Z,h) \le R_0$ and $R(Z+1,h) > R_0$. Furthermore, if $R(0,h) \ge R_0$, then $R(Z,h) \ge R_0$ for any nonnegative integer.

Therefore, in this case, we discuss an optimal software release period based on only the cost criterion.

From the discussion above, cost-reliability-optimal software release policies based on the discrete exponential SRGM in Eq. (2.10) can be obtained as follows:

Optimal Release Policy 2-1

Suppose that $c_2 > c_1 > 0$, $c_3 > 0$, $0 < R_0 \leq 1$, and $h \geq 0$.

(1) If $W_0^e \leq \frac{c_3}{c_2-c_1}$ and $R(0,h) < R_0$, then the optimal software release period is $Z^* = Z_1$.

(2) If $W_0^e \leq \frac{c_3}{c_2-c_1}$ and $R(0,h) \geq R_0$, then the optimal software release period is $Z^* = 0$.

(3) If $W_0^e > \frac{c_3}{c_2-c_1}$ and $R(0,h) < R_0$, then the optimal software release period is $Z^* = \max\{< Z_0 >,\ Z_1\}$.

(4) If $W_0^e > \frac{c_3}{c_2-c_1}$ and $R(0,h) \geq R_0$, then the optimal software release period is $Z^* = < Z_0 >$.

And also, based on the discrete inflection S-shaped SRGM in Eq. (2.15), optimal software release policies can be obtained as follows:

Optimal Release Policy 2-2

Suppose that $c_2 > c_1 > 0$, $c_3 > 0$, $0 < R_0 \leq 1$, and $h \geq 0$.

(1) If $W_{\bar{Z}}^I \leq \frac{c_3}{c_2-c_1}$ and $R(0,h) < R_0$, then the optimal software release period Z^* is $Z^* = Z_1$.

(2) If $W_{\bar{Z}}^I \leq \frac{c_3}{c_2-c_1}$ and $R(0,h) \geq R_0$, then the optimal software release period Z^* is $Z^* = 0$.

(3) If $W_{\bar{Z}}^I > \frac{c_3}{c_2-c_1} > W_0^I$ and $R(0,h) < R_0$, then the optimal software release period Z^* is $Z^* = Z_1$ when $C_0 < C_{<Z_0>}$, and $Z^* = \max\{< Z_0 >,\ Z_1\}$ when $C_0 \geq C_{<Z_0>}$.

(4) If $W_{\bar{Z}}^I > \frac{c_3}{c_2-c_1} > W_0^I$ and $R(0,h) \geq R_0$, then the optimal software release period Z^* is $Z^* = 0$ when $C_0 < C_{<Z_0>}$, and $Z^* =< Z_0 >$ when $C_0 \geq C_{<Z_0>}$.

(5) If $W_0^I > \frac{c_3}{c_2-c_1}$ and $R(0,h) < R_0$, then the optimal software release period Z^* is $Z^* = \max\{< Z_0 >,\ Z_1\}$.

(6) If $W_0^I > \frac{c_3}{c_2-c_1}$ and $R(0,h) \geq R_0$, then the optimal software release Z^* is $Z^* =< Z_0 >$.

2.6.3 Numerical examples

We show numerical examples for the derived optimal software release policies discussed in this section by using fault count data observed in actual testing-phases. We use DS1 and DS3

Table 2.4 : Numerical examples of cost-optimal software release policies for the discrete exponential SRGM.

c_1	c_2	c_3	Z^*	W_{Z^*}	$c_3/(c_2-c_1)$	$C_{Z^*}(\times 10^2)$
1	2	30	0	1.586×10	30	2.799
1	2	20	0	1.586×10	20	2.799
1	2	10	4	9.803	10	3.579
1	4	10	13	3.321	3.333	3.579
1	8	10	20	1.431	1.429	4.284
1	16	10	26	6.955×10^{-1}	6.667×10^{-1}	4.920
1	32	10	32	3.380×10^{-1}	3.226×10^{-1}	5.524

Table 2.5 : Numerical examples of cost-optimal software release policies for the discrete inflection S-shaped SRGM.

c_1	c_2	c_3	Z^*	$W_{Z^*}(\times 10^2)$	$c_3/(c_2-c_1)$	$C_{Z^*}(\times 10^2)$
1	2	180	0	1.437	180	104.4
1	2	150	17	1.479	150	102.6
1	2	100	26	0.975	100	91.72
1	4	100	41	0.331	0.333×10^2	105.2
1	8	100	51	0.142	0.143×10^2	114.9
1	16	100	60	0.065	0.067×10^2	123.5
1	32	100	68	0.032	0.032×10^2	131.5

for the discrete exponential SRGM and the discrete inflection S-shaped SRGM, respectively, which are the same data sets used in Section 2.5. In this section, we suppose that $c_1 = 1$ to consider the relative software costs.

First, we show numerical examples for the cost-optimal software release policies discussed in Subsection 2.6.1. Tables 2.4 and 2.5 show the cost-optimal software release periods derived from the **Optimal Release Policy 1-1** and the **Optimal Release Policy 1-2**, respectively. From Tables 2.4 and 2.5, we can say that the cost-optimal software release period, Z^*, becomes large as the debugging cost per one fault in the operational phase increases. That is, it is necessary to conduct the testing more as the maintenance cost increases. Fig. 2.16 depicts the behaviors of C_Z and W_Z^e, where $c_1 = 1$, $c_2 = 4$, and $c_3 = 10$. Additionally, Fig. 2.17 shows the behaviors of C_N and W_N^I, where $c_1 = 1$, $c_2 = 4$, and $c_3 = 100$.

Next, we show numerical examples for the cost-reliability-optimal software release policies discussed in Subsection 2.6.2. For the specified operational period $h = 1$ and the reliability

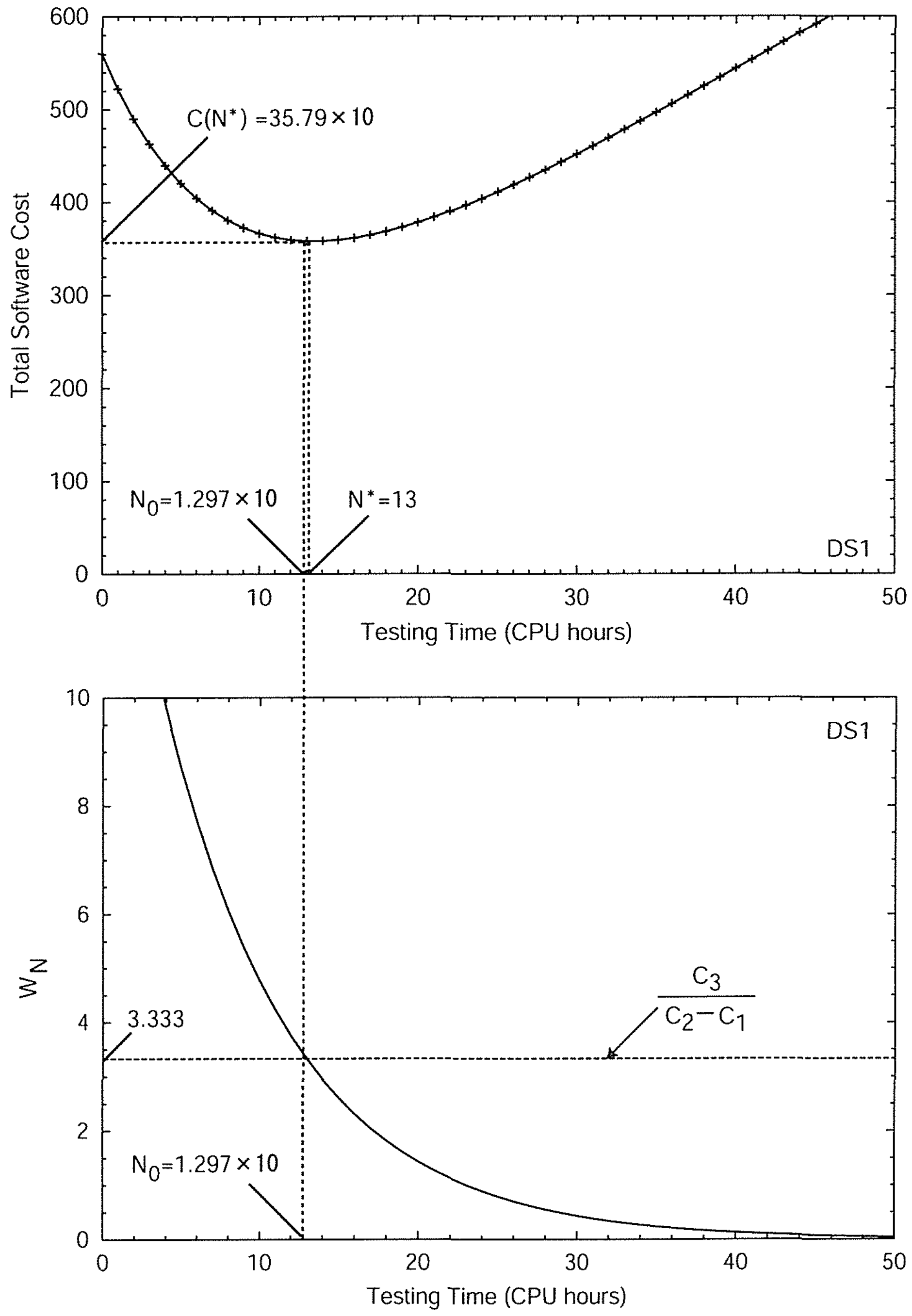


Fig. 2.16 : The cost-optimal software release policy for $c_1 = 1$, $c_2 = 4$, and $c_3 = 10$ based on the discrete exponential SRGM for DS1.

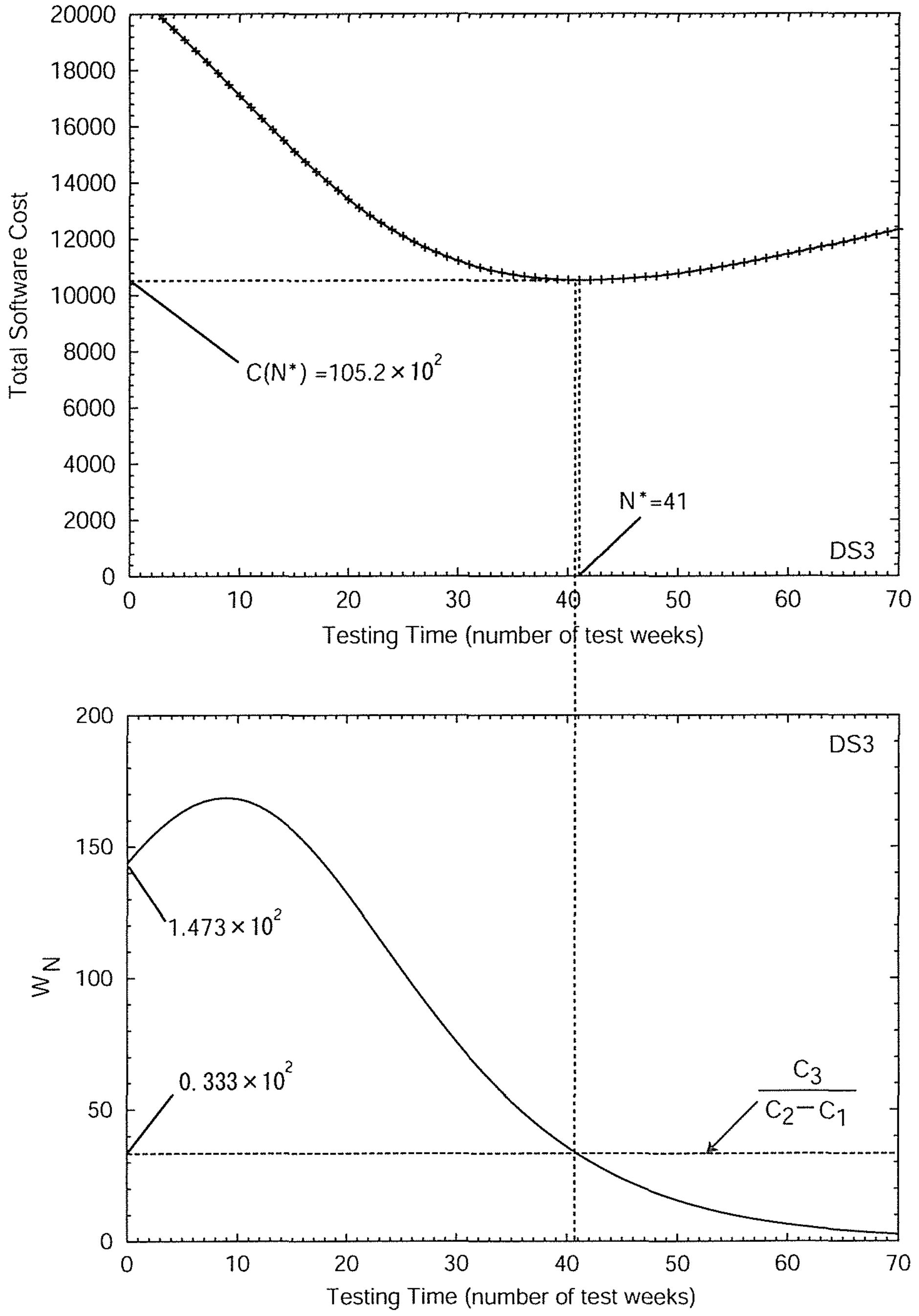


Fig. 2.17 : The cost-optimal software release policy for $c_1 = 1$, $c_2 = 4$, and $c_3 = 100$ based on the discrete inflection S-shaped SRGM for DS3.

objective $R_0 = 0 \cdot 8$, the cost-reliability-optimal software release problem can be formulated as

$$\left.\begin{array}{l}\text{Minimize } C(Z) \\ \text{subject to } R(Z,1) \geq 0.8,\ Z \geq 0\end{array}\right\}. \tag{2.41}$$

Suppose that the cost-optimal software release policies have been discussed a case of $c_1 = 1$, $c_2 = 4$, and $c_3 = 10$ for the discrete exponential SRGM and that of $c_1 = 1$, $c_2 = 4$, and $c_3 = 100$ for the discrete inflection S-shaped SRGM, respectively. Then, for the discrete exponential SRGM, we can estimate $Z_1 = 36$ because $R(35,1) < R_0$ and $R(36,1) > R_0$. And then, because $W_0^e > c_3/(c_2 - c_1)$ and $R(0,1) = 1 \cdot 296 \times 10^{-7} < R_0$, the cost-reliability-optimal software release period is estimated $Z^* = \max\{< Z_0 >,\ Z_1\} = \max\{13, 36\} = 36$ by using the **Optimal Release Policy 2-1** (3) (see Fig. 2.18). For the inflection S-shaped SRGM, we can estimate $Z_1 = 84$ because $R(83,1) < R_0$ and $R(84,1) > R_0$. Then, because $W_0^I > c_3/(c_2 - c_1)$ and $R(0,1) = 2 \cdot 364 \times 10^{-59} < R_0$, Z^* is estimated as $Z^* = \max\{< Z_0 >,\ Z_1\} = \max\{41, 84\} = 84$ by using the **Optimal Release Policy 2-2** (5) (see Fig. 2.19). In Figs. 2.18 and 2.19, we can understand the importance that the software development managers need to estimate an optimal software release period by considering not only minimizing the total expected software cost but also satisfying the reliability objective. For example, in Fig. 2.18, if the software development managers employ $< Z_0 >= 13$ minimizing the expected total software cost as the optimal software release period, the reliability objective $R_0 = 0.8$ is not satisfied because $R(13,1) = 3.611 \times 10^{-2}$. However, by considering the reliability objective $R_0 = 0.8$, the software testing has to be gone on to satisfy the reliability objective R_0.

2.7 Concluding Remarks

We have proposed discrete NHPP models which have exact solutions derived from the continuous exponential SRGM and inflection S-shaped SRGM, respectively. And we have discussed that the proposed discrete NHPP models have better performance for software reliability assessment in terms of the predicted relative error and the MSE than the discrete SDA models. And, we have derived several software reliability assessment measures, such as the number of remaining faults, the fault detection rates, the software reliability functions, and the reliability growth rates, for the discrete NHPP models. Additionally, we have discussed the cost-optimal and cost-reliability-optimal software release problems based on the discrete NHPP models. In an actual software development, the software developing manager has to spend and control the testing resources with minimizing the total software cost and satisfying the software reliabil-

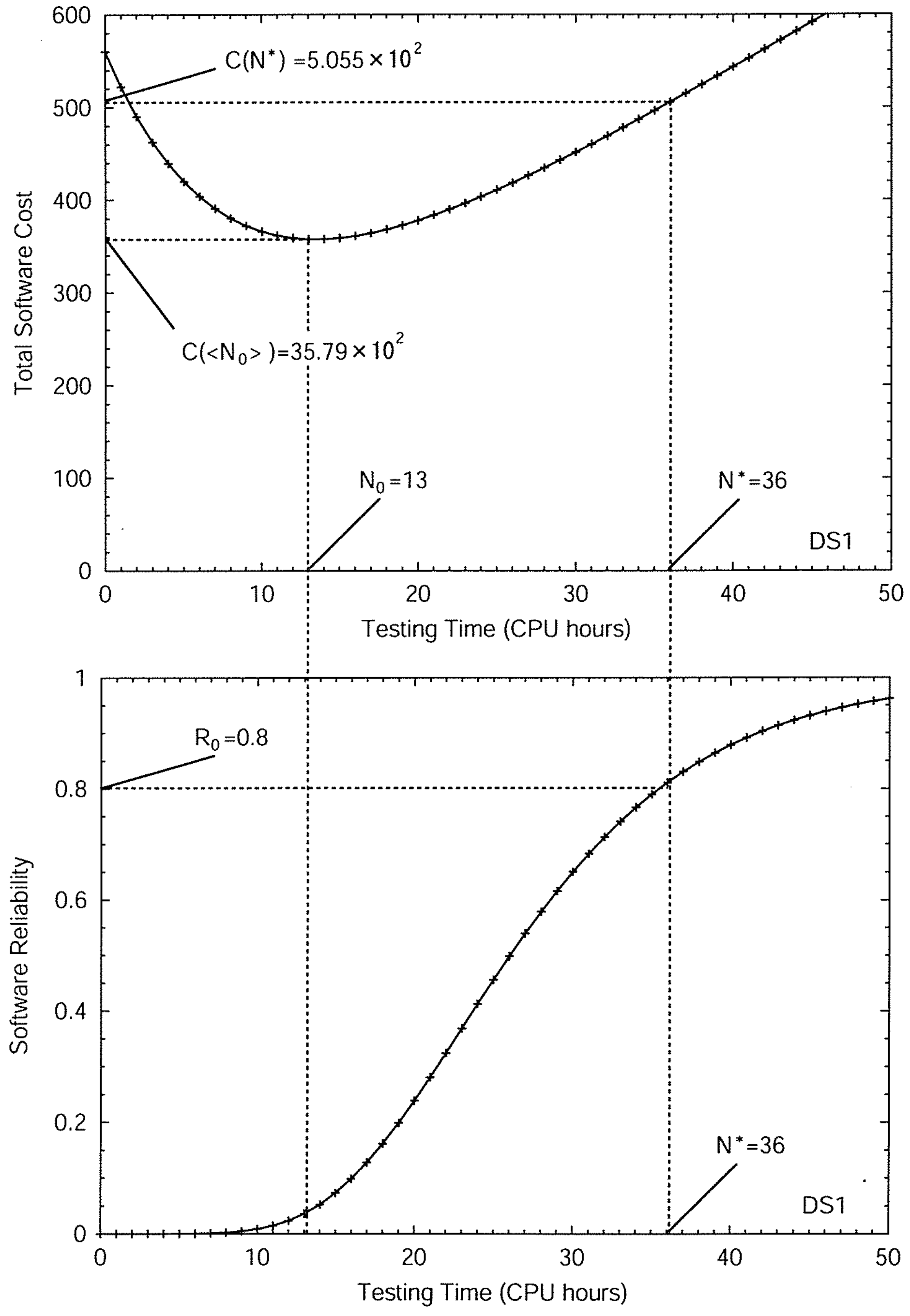


Fig. 2.18 : The cost-reliability-optimal software release policies on the subject of $R_0 = 0 \cdot 8$ based the discrete exponential SRGM for DS1.

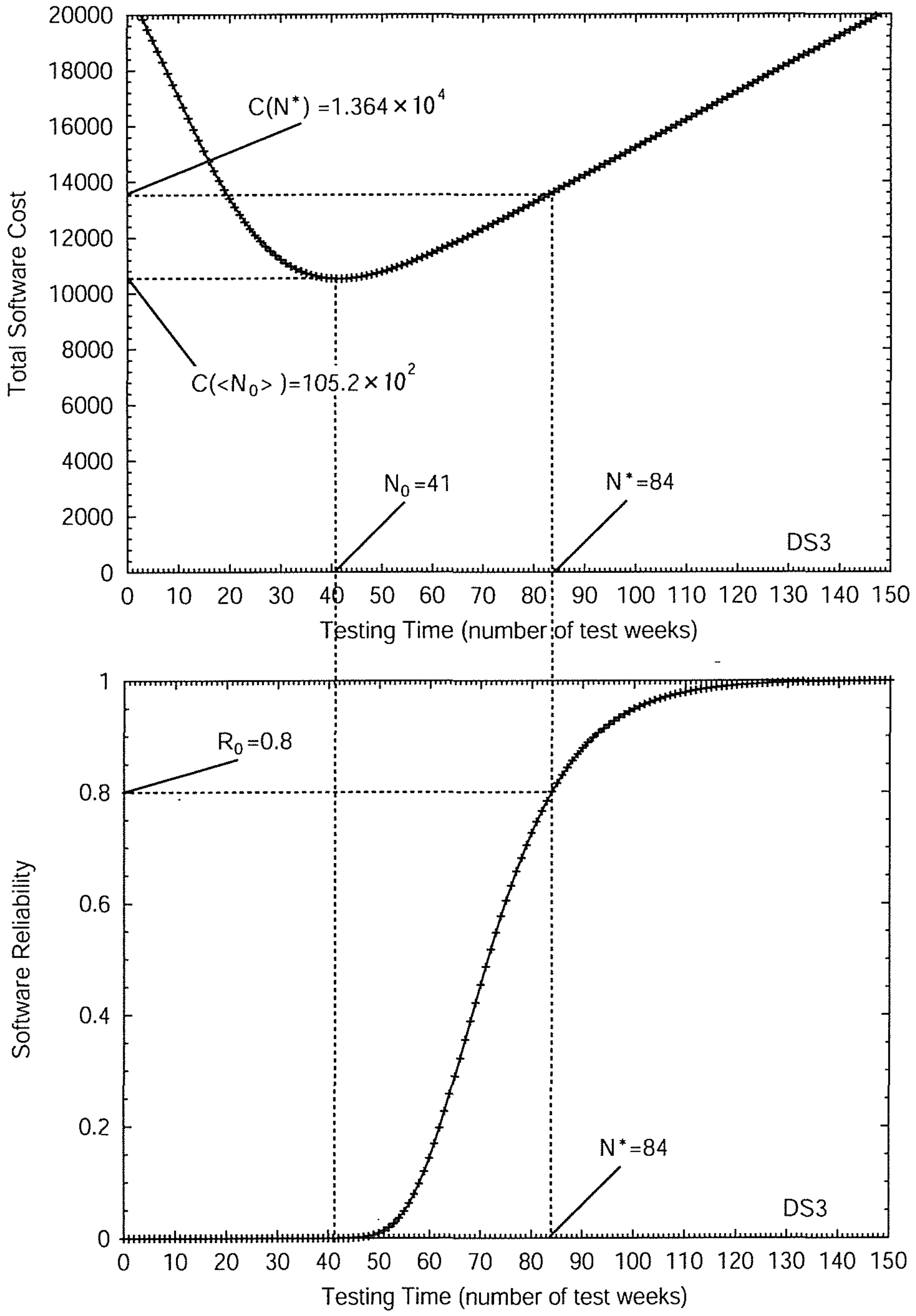


Fig. 2.19 : The cost-reliability-optimal software release policies on the subject of $R_0 = 0 \cdot 8$ based on the discrete inflection S-shaped SRGM for DS3.

ity requirements. Finally, we have shown numerical examples for our discrete NHPP models and the derived optimal software release policies by using fault count data collected in actual software development projects.

Chapter 3

Stochastic Software Reliability Modeling Based on Discretized SDA Models

3.1 Introduction

Software reliability assessment is one of the important issues to provide reliabile computer systems of which the size, complexity, and diversification are growing up drastically in recent years. An SRGM is known as one of the fundamental techniques to assess and predict software reliability quantitatively. The SRGM's have been modeled by using any stochastic processes to describe a software failure-detection phenomenon or a software failure-occurrence phenomenon in the testing or operational phases. Especially, the SRGM based on an NHPP, so-called an NHPP model, can describe software reliability growth process easily in the testing-phase by assuming a mean value function of the NHPP. Accordingly, the NHPP models have been utilized in actual software development projects.

On the other hand, deterministic SRGM's, such as logistic and Gompertz curve models, also have been practically utilized for software reliability assessment. The determinisitc SRGM's are called SDA models. These models are useful to predict the initial fault content and to describe software reliability growth process in the developed software system by regression analysis. In recent years, discretized SDA models have been proposed to assess software reliability more precisely than the continuous-time SDA models. Especially, Satoh [44] have proposed a discrete Gompertz curve model and its application to an SRGM. And, Satoh and Yamada [47] have discussed software reliability assessment methods using discrete logisitc curve models. And they have performed goodness-of-fit comparisons among these discretized SDA models by using a new goodness-of-fit evaluation criterion [46]. However, the continuous-time and the discrete-time SDA models can not provide several software reliability assessment measures, such as a software reliability function, an instantaneous MTBF, which are useful quantitative metrics for software reliability assessment.

In this chapter we discretize a modified exponential curve model, which is one of the typical SDA models, and discuss its parameter estimation method first. And, we propose stochastic software reliability growth models based on the discretized SDA models including the discrete modified exponential curve model by revising unsuitable properties of the discretized SDA models for practical application. Additionally, we perform goodness-of-fit comparisons of the proposed models with the discretized SDA models by using data sets collected in actual testing-phases. And several software reliability assessment measures derived from the proposed models are also discussed in this chapter.

3.2 Discretized SDA Models

In this section, we discuss discretized SDA models, such as logistic and Gompertz curve models, and their parameter estimation methods [44, 46, 47]. And, we derive a discrete modified exponential curve model and its parameter estimation method by discretizing the original continuous-time one. These discrete SDA models can be derived by discretizing the original continuous-time SDA models via using Hirota's bilinearization methods [14].

3.2.1 Discrete logistic curve model

Let L_n be the cumulative number of faults detected up to n-th testing-period. Then, a discrete logistic curve model [46, 47] is derived as the following difference equation based on the assumptions of the original continuous-time logistic curve model:

$$L_{n+1} - L_n = \delta \frac{\alpha}{k} L_{n+1}(k - L_n). \tag{3.1}$$

The exact solution of Eq. (3.1) is derived as

$$L_n = \frac{k}{1 + m(1 - \delta\alpha)^n} \qquad (k > 0,\ \alpha > 0,\ m > 0), \tag{3.2}$$

where k, α, and m are the constant parameters to be estimated by regression analysis, and δ the constant time-interval. We can see that parameter k in Eq. (3.2) represents the initial fault content in the software system because

$$L_n \to k \quad \text{as} \quad n \to \infty. \tag{3.3}$$

There are two types of discrete logistic curve models: Morishita type [28] and Hirota type [14]. Eq. (3.1) is the discrete logistic curve model proposed by Morishita.

A regression equation to estimate the parameters k, α, and m is obtained as

$$Y_n = A + BL_{n+1}, \tag{3.4}$$

by rewriting Eq. (3.1), where

$$\left.\begin{aligned} Y_n &= \frac{L_{n+1}}{L_n} \\ A &= \frac{1}{1-\delta\alpha} \\ B &= -\frac{\delta\alpha}{k(1-\delta\alpha)} \end{aligned}\right\}. \tag{3.5}$$

By regression analysis, we can estimate $\widehat{A}$ and $\widehat{B}$ which are the estimates of A and B in Eq. (3.4). Then, the parameter estimates $\widehat{k}$, $\widehat{\alpha}$, and $\widehat{m}$ which are the estimated value of k, α, and m can be obtained as

$$\left.\begin{aligned} \widehat{k} &= \frac{1-\widehat{A}}{\widehat{B}} \\ \widehat{\alpha} &= 1 - \frac{1}{\widehat{A}} \\ \widehat{m} &= \frac{\sum_{n=1}^{N}(\widehat{k} - L_n)}{\sum_{n=1}^{N}(L_n(1-\delta\widehat{\alpha}))} \end{aligned}\right\}. \tag{3.6}$$

3.2.2 Discrete Gompertz curve model

Satoh [44] has proposed a discrete Gompertz curve model. Let G_n be the cumulative number of faults detected up to n-th testing-period, the discrete Gompertz curve model can be derived as

$$G_{n+1} = G_n \left(\frac{G_n}{k}\right)^{\delta \log b}, \tag{3.7}$$

by discretizing the original continuous-time Gompertz curve model. The exact solution of Eq. (3.7) is obtained as

$$G_n = ka^{(1+\delta \log b)^n} \qquad (k > 0,\ 0 < a < 1,\ 0 < b < 1), \tag{3.8}$$

by solving the integrable difference equation in Eq. (3.7) with respect to G_n. In Eq. (3.8), k, a, and b are the constant parameters which are estimated by regression analysis, δ the constant time-interval. Parameter k represents the initial fault content because

$$G_n \to k \quad \text{as} \quad n \to \infty. \tag{3.9}$$

A regression equation to estimate the parameters k, a, and b is obtained as

$$Y_n = A + B \log G_n, \tag{3.10}$$

by rewriting Eq. (3.7), where

$$\left.\begin{aligned} Y_n &= \log G_{n+1} - \log G_n \\ A &= -\delta(\log b)(\log k) \\ B &= \delta \log b \end{aligned}\right\}. \tag{3.11}$$

The parameter estimates $\widehat{k}$, $\widehat{a}$, and $\widehat{b}$ can be given by

$$\left.\begin{aligned} \widehat{k} &= \exp\left(-\frac{\widehat{A}}{\widehat{B}}\right) \\ \widehat{a} &= \exp\left(\frac{\sum_{n=1}^{N} \log \frac{G_n}{k}}{\sum_{n=1}^{N} (1 + \delta \log \widehat{b})^n}\right) \\ \widehat{b} &= \exp\left(\frac{\widehat{B}}{\delta}\right) \end{aligned}\right\}, \tag{3.12}$$

where $\widehat{A}$ and $\widehat{B}$ are the estimates of A and B in Eq. (3.10), and $\widehat{k}$, $\widehat{a}$, and $\widehat{b}$ are also the estimated values of k, a, and b.

3.2.3 Discrete Modified Exponential Curve Model

We propose a discrete modified exponential curve model to discuss typical SDE models exhaustively. This discrete model is derived by discretizing the original continuous-time one. We apply the Hirota's bilinearization method [14] to derive an integrable difference equation for the modified exponential curve model. Then, we can obtain a difference equation which conserves the properties of the original differential equation for the modified exponential curve model.

Let E_n be the cumulative number of faults detected up to n-th testing-period. the discrete modified exponential curve model can be derived as

$$E_{n+1} - E_n = -\delta \log r \left(k - E_n\right), \tag{3.13}$$

based on the assumptions of the original continuous-time one. The exact solution of Eq. (3.13) is derived as

$$E_n = k - q\left(1 + \delta \log r\right)^n \qquad (k > 0,\ 0 < r < 1,\ q > 0), \tag{3.14}$$

by solving the integrable difference equation in Eq. (3.13). In Eq. (3.14), k, q, and r are the constant parameters to be estimated by regression analysis, and δ the constant time-interval. Parameter k indicates the initial fault content in the software system because

$$E_n \to k \quad \text{as} \quad n \to \infty. \tag{3.15}$$

We can see that Eq. (3.14) converges to an exact solution of the continuous-time modified exponential curve model as $\delta \to 0$. That is,

$$\begin{aligned}\lim_{\delta\to 0} E_n &= \lim_{\delta\to 0} \left[k - q\,(1+\delta\log r)^n\right] \\ &= k - qr^t.\end{aligned} \tag{3.16}$$

A regression equation to estimate the parameters k, r, and q is obtained as

$$Y_n = A + BE_{n+1}, \tag{3.17}$$

by rewriting Eq. (3.13), where

$$\left.\begin{aligned} &Y_n = E_{n+1} - E_n \\ &A = -\delta k \log r \\ &B = \delta \log r \end{aligned}\right\}. \tag{3.18}$$

By regression analysis, we can estimate $\widehat{A}$ and $\widehat{B}$ which are the estimates of A and B in Eq. (3.17). Then, the parameter estimates $\widehat{k}$, $\widehat{r}$, and $\widehat{q}$ which are the estimated values of k, r, and q, respectively, can be obtained as follows:

$$\left.\begin{aligned} &\widehat{k} = -\frac{\widehat{A}}{\widehat{B}} \\ &\widehat{r} = \exp\left(\frac{\widehat{B}}{\delta}\right) \\ &\widehat{q} = \frac{\sum_{n=1}^{N}(\widehat{k} - E_n)}{\sum_{n=1}^{N}(1+\delta\log\widehat{r})} \end{aligned}\right\}. \tag{3.19}$$

Y_n in Eq. (3.17) is independent of δ because δ is not used in calculating Y_n in Eq. (3.18). Hence, we can obtain the same parameter estimates of $\widehat{k}$, $\widehat{r}$, and $\widehat{q}$, respectively, when we choose any value of δ.

3.3 NHPP Modeling Based on Discretized SDA Models

The discretized SDA models discussed in Section 3.2 are deterministic models which do not assume stochastic properties for the fault-detection phenomenon in the testing-phase. Therefore, we can not derive typical software reliability assessment measures which are useful quantitative metrics for software reliability assessment. In this chapter, we develop stochastic SRGM's based on the discretized SDA models by incorporating discrete counting processes, which can be more widely used than the ordinary discretized SDA models. That is, we describe behavior of the expected cumulative number of faults detected up to arbitrary testing-period based on the those deterministic models.

First, we assume that the discrete counting process $\{N_n, n \geq 0\} (n = 0, 1, 2, \cdots)$ representing the cumulative number of faults detected up to n-th testing-period has the following properies based on the continuous-time NHPP:

(P-1) : $N_0 = 0$ with probability 1.

(P-2) : The discrete counting process $\{N_n, n \geq 0\}$ $(n = 0, 1, 2, \cdots)$ has independent increments.

(P-3) : For arbitrary testing-periods n_i and n_j $(0 \leq n_i \leq n_j)$,

$$\Pr\{N_{n_j} - N_{n_i} = x\} = \frac{\{\Lambda_{n_j} - \Lambda_{n_i}\}^x}{x!} \exp[-\{\Lambda_{n_j} - \Lambda_{n_i}\}] \qquad (x = 0, 1, 2, \cdots), \tag{3.20}$$

where $\Pr\{A\}$ indicates the probability of an event A.

We can rewrite Eq. (3.20) as

$$\Pr\{N_n = x \mid N_0 = 0\} = \frac{\{\Lambda_n\}^x}{x!} \exp[-\Lambda_n] \qquad (n, x = 0, 1, 2, \cdots), \tag{3.21}$$

which is the same as Eq. (2.8). In Eq. (3.21), Λ_n in Eqs. (3.20) and (3.21) is a mean value function of the discrete NHPP, which represents the expected cumulative number of faults detected up to n-th testing-period.

We derive discrete NHPP models with the mean value functions L_n, G_n, and E_n for the discrete logistic, Gompertz, and modified exponential curve models to describe the behavior of the expected cumulative number of detected faults, respectively. However, we should note that

$$L_0 = \frac{k}{1+m}, \tag{3.22}$$

$$G_0 = ka, \tag{3.23}$$

$$E_0 = k - q, \tag{3.24}$$

which imply that these discrete models do not satisfy the property of the discrete NHPP and they have unsuitable properties for practical applications. Accordingly, we have to devise L_n, G_n, and E_n as

- discrete logistic curve model based on the discrete NHPP

$$L_n^{NHPP} = k\left\{\frac{1}{1+m(1-\delta\alpha)^n} - \frac{1}{1+m}\right\}, \tag{3.25}$$

- discrete Gompertz curve model based on the discrete NHPP

$$G_n^{NHPP} = k\left\{a^{(1+\delta\log b)^n} - a\right\}, \tag{3.26}$$

- discrete modified exponential curve model on the discrete NHPP

$$E_n^{NHPP} = k\left\{1-(1+\delta\log r)^n\right\}, \tag{3.27}$$

to derive discrete NHPP models based on the discretized SDA models, respectively. Applying these models above as the mean value functions of the discrete NHPP's, we can derive useful software reliability assessment measures such as software reliability functions, instantaneous MTBF's.

Next, we discuss parameter estimation methods for the proposed stochastic models in Eqs. (3.25)-(3.27). We write these mean value functions as Λ_n generally in this section. As the parameter estimation methods in this chapter, we use the method of maximum-likelihood [38, 52,54,55]. Now, suppose that we have observed K data pairs $(n_k,\ y_k)(k=0,1,2,\cdots,K)$ with respect to the total faults, y_k, detected during constant time-interval $(0,\ n_k](0<n_1<n_2<\cdots<n_K)$. Then, we can derive the following likelihood function from the properties of the discrete NHPP:

$$\begin{aligned} L &\equiv \Pr\left[N_{n_1}=y_1, N_{n_2}=y_2,\cdots,N_{n_K}=y_K\right] \\ &= \prod_{k=1}^{K}\Pr\left[N_{n_k}-N_{n_{k-1}}=y_k-y_{k-1}\right] \\ &= \exp[-\Lambda_{n_K}]\prod_{k=1}^{K}\frac{\{\Lambda_{n_k}-\Lambda_{n_{k-1}}\}^{(y_k-y_{k-1})}}{(y_k-y_{k-1})!}, \end{aligned} \tag{3.28}$$

where $n_0=0$ and $y_0=0$. Then, the logarithmic likelihood function is derived as

$$\ln L = \sum_{k=1}^{K}(y_k-y_{k-1})\ln\left[\Lambda_{n_k}-\Lambda_{n_{k-1}}\right] - \Lambda_{n_K} - \sum_{k=1}^{K}\ln[(y_k-y_{k-1})!]. \tag{3.29}$$

The parameter estimates of each discrete NHPP model proposed in this chapter can be obtained by solving the simultaneous logarithmic likelihood functions which is derived by partially differentiating the logarithmic likelihood funtion in Eq. (3.29) in terms of the unknown parameters.

3.4 Model Comparisons

We perform goodness-of-fit comparisons of the proposed stochastic discretized models based on the discrete NHPP's in Section 3.3 with the discretized SDA models in Section 3.2 in terms of the predicted relative error and the MSE which have been discussed in Section 2.4. First, we arrange data sets used in model comparisons in this chapter as follows:

DS1 [56] : $(n_k, y_k)(k = 1, 2, \cdots, 21\ ;\ t_{21} = 21, y_{21} = 46)$ where n_k is measured on the basis of days,

DS2 [38] : $(n_k, y_k)(k = 1, 2, \cdots, 10\ ;\ t_{10} = 10, y_{10} = 93)$ where n_k is measured on the basis of hours.

The data set called DS1 in this chapter indicates an S-shaped growth curve, and DS2 an exponential one. Therefore, we use DS1 for the discrete NHPP models based on the logistic and Gompertz curve models, and DS2 for the model based on the modified exponential curve model.

We show the results of model comparisons. Figs. 3.1–3.3 show the results of the model comparisons based on the predicted relative errors for the discrete NHPP models based on the logistic, Gompertz, and modified exponential curve models, respectively. From these figures,

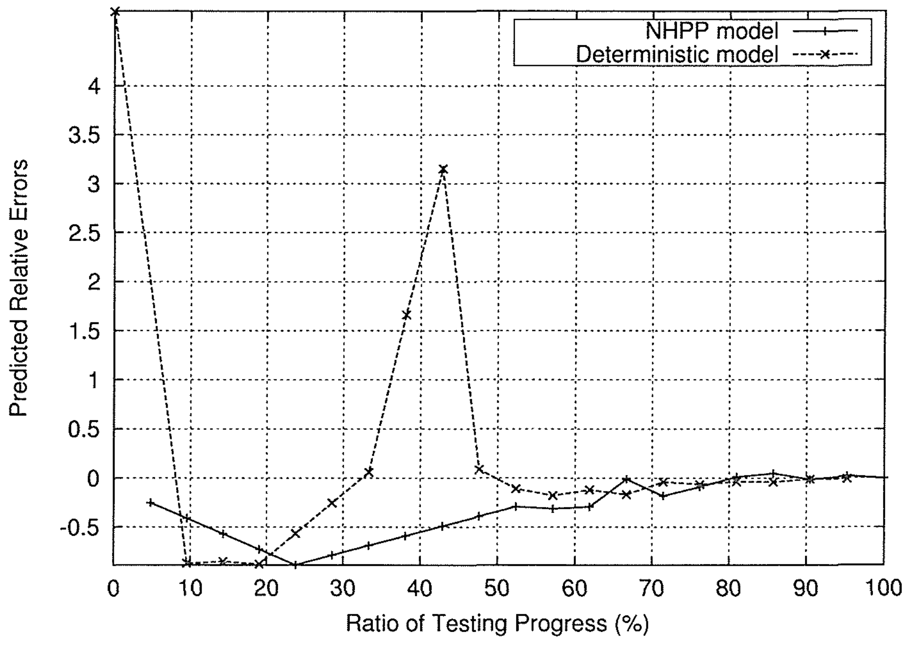


Fig. 3.1 : The predicted relative errors of L_n and L_n^{NHPP} for DS1.

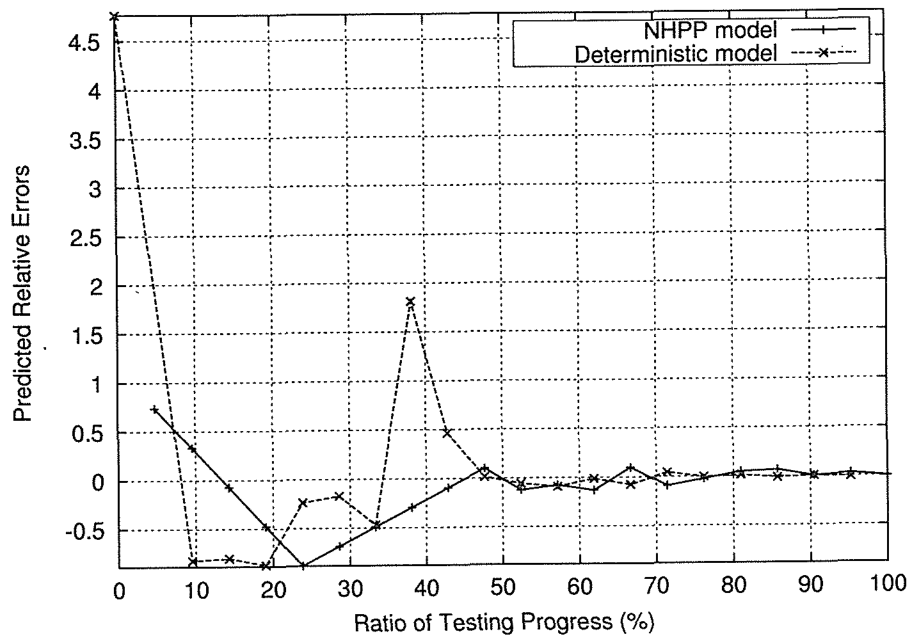


Fig. 3.2 : The predicted relative errors of G_n and G_n^{NHPP} for DS1.

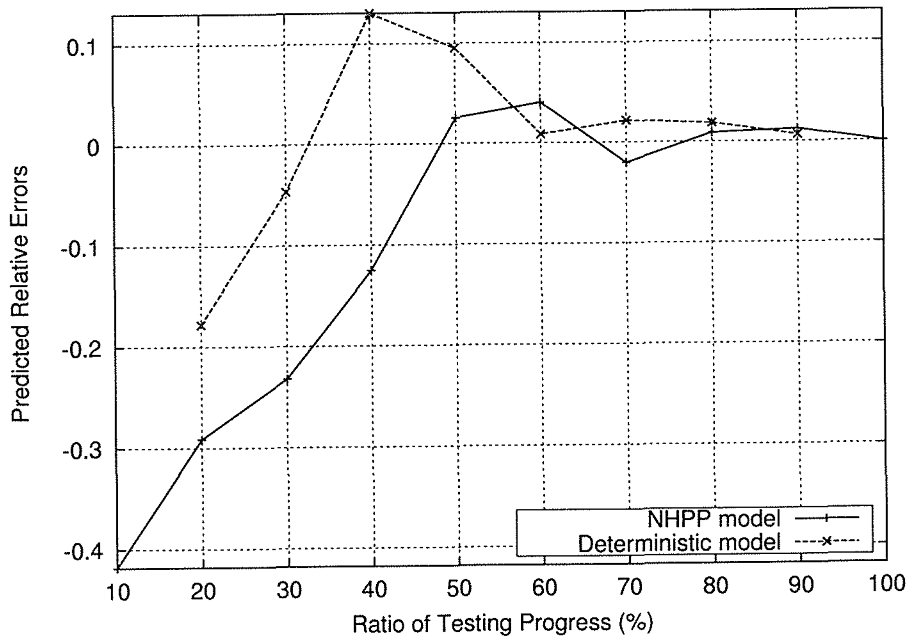


Fig. 3.3 : The predicted relative errors of E_n and E_n^{NHPP} for DS2.

Table 3.1 : The results of model comparisons based on the MSE.

Data set	Model	MSE
DS1	·Discrete logistic curve model	5.272
	·Discrete logistic curve model based on an NHPP	**1.443**
DS1	·Discrete Gompertz curve model	1.862
	·Discrete Gompertz curve model based on an NHPP	**1.497**
DS2	·Discrete modified exponential curve model	7.271
	·Discrete modified exponential curve model based on an NHPP	**2.563**

we can say that our each stochastic model propopsed in this chapter provides more stable and better prediction than the discrete SDA model. And, we can also see that stable predictions are provided after 60% of the testing-progress ratio for all models proposed in this chapter. Additionally, Table 3.1 shows the results of model comparisons based on the MSE. From the table, we can see that the proposed discretized stochastic models fit better to the actual data than the discretized SDA models. From the results of goodness-of-fit comparisons, we can say that the proposed stochastic models have better performance for the software reliability assessment in terms of the predicted relative error and the MSE.

3.5 Software Reliability Assessment Measures

We can derive useful metrics for quantitative software reliability assessment based on the proposed discretized stochastic SRGM's by using the properties of the discrete NHPP's. In this section, several software reliability assessment measures such as software reliability functions and instantaneous MTBF's are derived based on the properties of the discrete NHPP.

3.5.1 Software reliability functions

We can derive software reliability functions by using the properties of the discrete NHPP. Given that the testing has been going on up to n-th testing-period by which x software faults have been detected, the software reliability function is defined as the propability that a software failure does not occur in the time interval $(n,\ n+h](h = 0, 1, 2, \cdots)$. Then, the software reliability function, $R(n,h)$, can be formulated as

$$\begin{aligned} R(n,h) &\equiv \Pr\{N_{n+h} - N_n = 0 | N_n = x\} \\ &= \exp[-\{\Lambda_{n+h} - \Lambda_n\}] \qquad (n,\ h = 0, 1, 2, \cdots), \end{aligned} \tag{3.30}$$

which is the same as Eq. (2.29). Suppose $\delta = 1$ and that the software users operate these software under the same environments as the software testing environment after releasing those software at these testing termination time. Figs. 3.4 and 3.5 depict the estimated software reliability function for L_n^{NHPP} and G_n^{NHPP} after the testing termination time $n = 21$ (days) of DS1 and for E_n^{NHPP} after the testing termination time $n = 10$ (hours) of DS2, respectively. As to DS1, we can estimate the software reliability, $\widehat{R}(21,1)$, for the discrete logistic curve model based on the discrete NHPP to be about $0 \cdot 224$, for the discrete Gompertz curve model based on the discrete NHPP to be about $0 \cdot 164$, respectively. And we can also estimate $\widehat{R}(10,1)$ for the discrete modified exponential curve model based on the discrete NHPP to be about $0 \cdot 233$ by using DS2.

3.5.2 Instantaneous MTBF's

We also estimate *instantaneous MTBF's* (*mean time between software failures*) which have been used as one of the substitution of measures of MTBF. The instantaneous MTBF can be

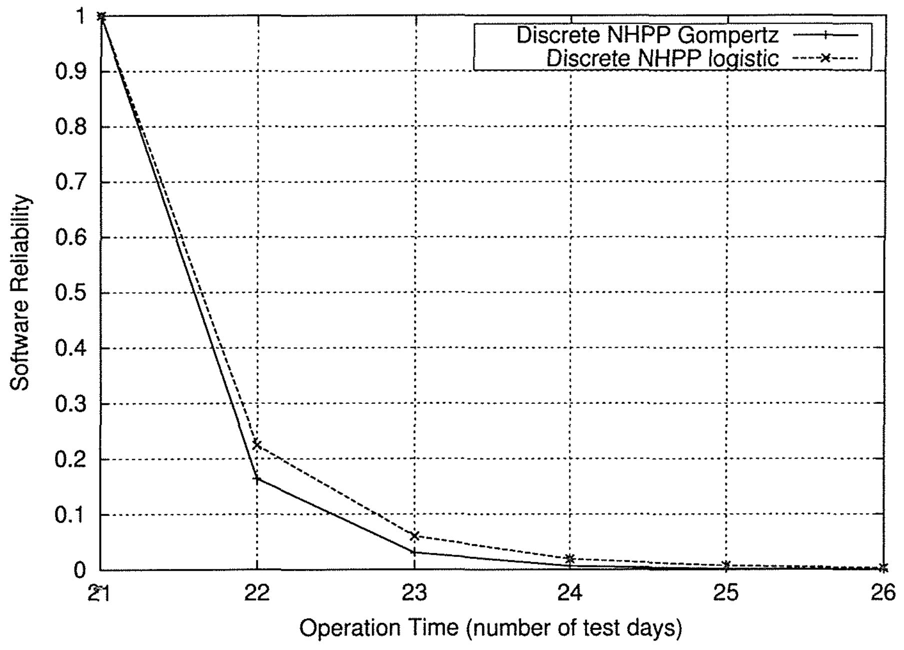


Fig. 3.4 : The estimated software reliability functions of L_n^{NHPP} and G_n^{NHPP} for DS1.

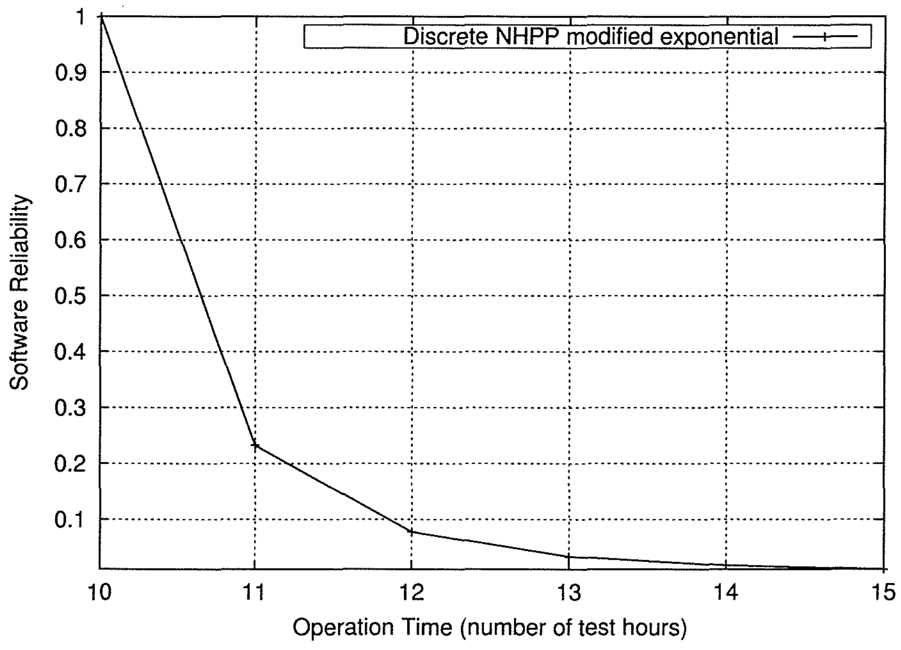


Fig. 3.5 : The estimated software reliability function of E_n^{NHPP} for DS2.

obtained as

$$\mathrm{MTBF}_I(n) = \frac{1}{z_n}, \tag{3.31}$$

where z_n is formulated as $z_n = \Lambda_{n+1} - \Lambda_n$. Figs. 3.6 and 3.7 show the time-dependent behavior of the estimated instantaneous MTBF's, respectively. We can estimate $\widehat{\mathrm{MTBF}}_I(21)$ for the discrete logistic curve model based on the discrete NHPP to be about $0 \cdot 669$ (days), for the discrete Gompertz curve model based on the discrete NHPP to be about $0 \cdot 553$ (days), respectively. And also, we can estimate $\widehat{\mathrm{MTBF}}_I(10)$ for the discrete modified exponential curve model based on the discrete NHPP to be about $0 \cdot 686$ (hours).

3.6 Concluding Remarks

In this chapter we have discretized a modified exponential curve model which is one of the typical SDA models, and developed a discrete modified exponential curve model by using the Hirota's bilinearization method. Based on the discretized SDA models including the discretized modified exponential curve model, which are the deterministic models, we have developed discretized stochastic models by incorporating counting processes such as discrete NHPP's. The proposed models can describe the time-dependent behavior of software reliability growth process

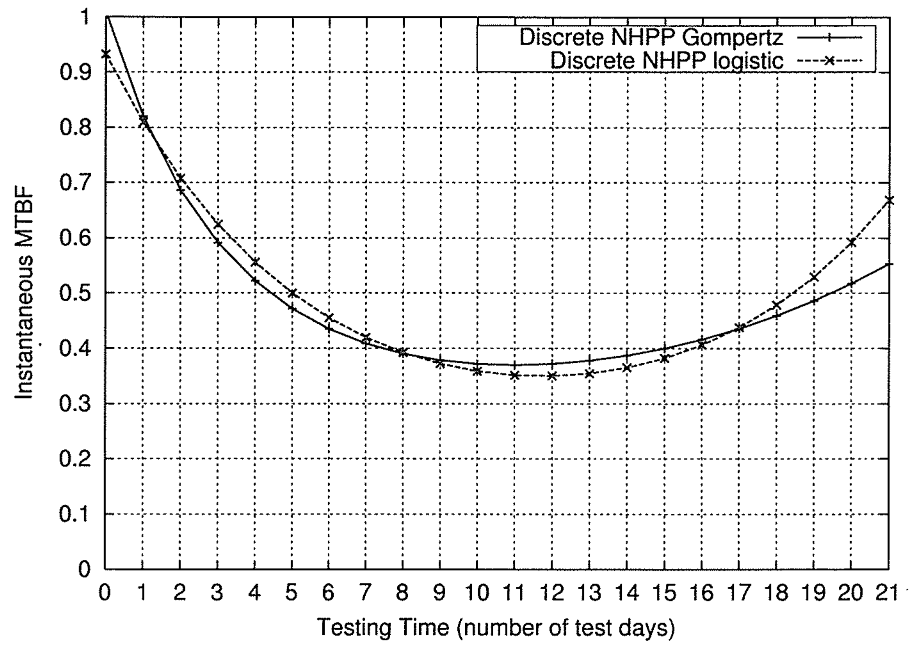


Fig. 3.6 : The estimated instantaneous MTBF's of L_n^{NHPP} and G_n^{NHPP} for DS1.

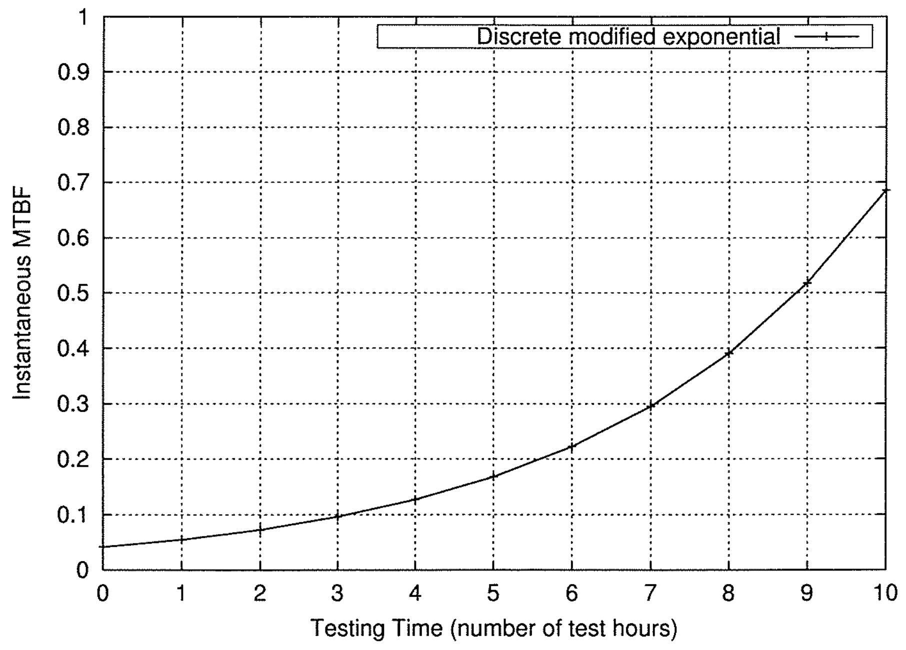


Fig. 3.7 : The estimated instantaneous MTBF of E_n^{NHPP} for DS2.

stochastically with conserving the basic properties of the discretized SDA models. Additionally, we have discussed that the proposed stochastic models based on the discretized SDA models have better performance for software reliability assessment in terms of the predicted relative error and the MSE in the goodness-of-fit comparisons in Section 3.4. Several software reliability assessment measures which are useful metrics for quantitative software reliability assessment have been derived.

Part II

MODELING WITH SEVERAL RELIABILITY FACTORS

Chapter 4

Software Reliability Modeling with Testing-Coverage

4.1 Introduction

Software reliability assessment is one of the important issues to produce reliable software systems. An SRGM has been utilized as one of the fundamental techniques for quantitative assessment of software reliability. The SRGM describes the software fault-detection phenomenon or the software failure-occurrence phenomenon by applying stochastic and statistical theories. Especially, it is well-known that NHPP model can characterize software reliability growth process simply by supposing a suitable mean value function of the NHPP. Accordingly, the NHPP model has been utilized for software reliability assessment in many practical software development project from high applicability and simplicity of the model structure of the NHPP point of view [18]. Up to now, several specific NHPP models have been proposed such as imperfect debugging SRGM's [19, 72], testing-domain dependent SRGM's [68]. The testing-domain dependent SRGM's have been derived by considering the time-dependent behavior of a testing-domain coverage which is a factor related to the software reliability growth process.

In this chapter we focus on testing-coverage as a key factor related to the software reliability growth process. The testing-coverage is one of the important measures to evaluate the quality of testing and tested software products. There are several researches on the relationship between the testing-coverage and the software reliability. Specifically, Fujiwara and Yamada [12] and Malaiya et al. [27] have proposed software reliability growth models with the testing-coverage, respectively. And Pham and Zhang [39] have also proposed an NHPP model and software cost models with the testing-coverage. However, our approach is different from these approaches above in terms of the relationship between the testing-coverage and the software reliability growth process. First we propose an alternative testing-coverage function to describe time-dependent behavior of testing-coverage maturity process. Then, we formulate the relationship between the attained testing-coverage and the number of detected faults. Estimation methods for unknown parameters of the alternative testing-coverage function and our SRGM are also discussed, respectively. Finally, we derive several software reliability assessment measures which are useful metrics for quantitative software reliability assessment, and show numerical examples of software reliability analysis based on the proposed model by using an actual data set.

4.2 Testing-Coverage

Testing-coverage is one of the important measures to evaluate the quality of testing and tested software products. The typical testing-coverage measures are classified into several types in terms of control flow testing as follows:

- Statement coverage : It is measured on the basis of the statement-paths that have been executed at least once by the test-cases. This is called C0 testing-coverage measure. The C0 testing-coverage measure should satisfy 100% testing-coverage in actual testing.
- Branch coverage : It is measured on the basis of the branch-paths that have been executed at least once by the test-cases. This is called C1 testing-coverage measure. It is said that at least 85% of C1 testing-coverage should be attained from a quality assurance point of view [64].
- Path coverage : It is measured on the basis of the all distinct program paths that have been executed at least once by the test-cases.

Most SRGM's are ordinally developed by characterizing the relationship between the testing-time and the number of detected software faults. Accordingly, we need to characterize the relationship between the testing-time and the testing-coverage to develop an SRGM considering with testing-coverage maturity process first. In this section we discuss basic concepts to describe time-dependent behavior of testing-coverage maturity process.

4.2.1 Formulation

First we propose a basic equation to describe time-dependent behavior of testing-coverage maturity process, which is called an *alternative testing-coverage function* in this chapter. For developing the equation, we assume that the testing-coverage maturity rate at any testing-time is proportional to the difference between the target value and the current one of testing-coverage. Letting $C(t)$ be the ratio of testing-coverage attained by arbitrary testing-time t, we can derive the following differential equation from the assumption:

$$\frac{dC(t)}{dt} = \beta(t)[\alpha - C(t)] \qquad (0 < \alpha \leq 1.0,\ \beta(t) \geq 0), \tag{4.1}$$

where α indicates the target value of testing-coverage to be attained, and $\beta(t)$ the testing-coverage maturity ratio at arbitrary testing-time t. We can easily obtain the alternative testing-coverage function by solving the above differential equation with respect to $C(t)$.

4.2.2 Formulation with testing-skill

The testing-coverage helps software development managers to evaluate whether the test-cases have been designed to detect faults effectively. Accordingly, the time-dependent behavior of the testing-coverage depends on a testing-skill of test-case designers. In this chapter we assume that the testing-skill of test-case designers increases as the ratio of testing-progress goes on. Suppose that the testing-skill factor for the test-case designers is given as

$$r = \frac{b_{ini}}{b_{sta}}. \tag{4.2}$$

Then, we extend $\beta(t)$ in Eq. (4.1) as follows:

$$\beta(t) \equiv B(C(t)) = b_{sta}\{r + (1-r)\frac{C(t)}{\alpha}\}, \tag{4.3}$$

where b_{ini} represents the initial testing-skill factor of the test-case designers, b_{sta} the steady-state one, and r the inflection coefficient. Substituting Eq. (4.3) into Eq. (4.1), we can obtain the following equation by solving the differential equation in Eq. (4.1):

$$C(t) = \frac{\alpha(1 - e^{-b_{sta}\cdot t})}{1 + z \cdot e^{-b_{sta}\cdot t}}, \tag{4.4}$$

where $z = (1-r)/r$. We call Eq. (4.4) an *alternative testing-coverage function with testing-skill* in this chapter. The inflection point of the alternative testing-coverage function in Eq. (4.4) is derived as

$$t^* = \frac{\log z}{b_{sta}}. \tag{4.5}$$

Then, we have

$$C(t^*) = \frac{\alpha}{2}\left(1 - \frac{1}{z}\right). \tag{4.6}$$

In Eq. (4.4), $C(t)$ indicates an exponential growth curve when $r = 1$ for the case that the internal program structure is simple, and indicates an S-shaped one called a logistic curve as $r \to 0$ for the case that the internal program structure is complex and the testing-effort increases more and more as the testing-time goes on.

4.3 Software Reliability Modeling

4.3.1 NHPP model

Time-dependent behavior of a software fault-detection phenomenon or a failure-occurrence phenomenon, i.e., the software reliability growth process, has been formulated by using a counting process ordinarily. An NHPP which is one of the counting processes is widely used for software reliability growth modeling. A counting process $\{N(t),\ t \geq 0\}$ is said to be an NHPP with mean value function $H(t)$ if $N(t)$ obeys the following distribution:

$$\left.\begin{aligned} \Pr\{N(t)=n\} &= \frac{\{H(t)\}^n}{n!}\exp\{-H(t)\} \qquad (n=0,1,2,\cdots), \\ H(t) &= \int_0^t h(\tau)d\tau, \end{aligned}\right\}, \tag{4.7}$$

where $h(\tau)$ is the intensity function representing the instantaneous fault-detection rate. The time-dependent behavior of the fault-detection process is characterized by the mean value function $H(t)$ which means the expected number of faults detected in the time-interval $(0,\ t]$. In this chapter we assume that the expected number of faults detected at testing-time t is proportional to the expected current fault content. Accordingly, we can obtain the following differential equation:

$$\frac{dH(t)}{dt} = b(t)[a - H(t)], \tag{4.8}$$

where a represents the expected initial fault content in the software system, and $b(t)$ the fault-detection rate per fault at testing-time t. The mean value function can be derived as

$$H(t) = a\left(1 - \exp[-\int_0^t b(\tau)d\tau]\right), \tag{4.9}$$

by solving the differential equation in Eq. (4.8). Most NHPP models can be characterized by $b(t)$ in Eq. (4.8) or Eq. (4.9), e.g., Eq. (4.9) is so-called a Goel-Okumoto SRGM [13] when $b(t) \equiv b$.

4.3.2 Modeling with testing-coverage

In this section we propose a software reliability growth model considering with testing-coverage maturity process based on the NHPP. First we formulate the relationship between the testing-coverage maturity process and the expected number of detected faults.

Supposing that the expected number of faults detected at testing-time t is proportional to the expected current fault content and the attained testing-coverage at testing-time t, we can formulate the relationship as the following equation by using Eq. (4.4):

$$\frac{dH_C(t)}{dt} \bigg/ \frac{dC(t)}{dt} = s[a - H_C(t)], \tag{4.10}$$

where s is the fault-detection rate per attained testing-coverage and per fault. That is, $b(t)$ in Eq. (4.8) or (4.9) is given as $b(t) \equiv b_C(t) = s \cdot c(t)$ in which $c(t) \equiv dC(t)/dt$. Then, we can obtain the following solution by solving the differential equation in Eq. (4.10) with respect to $H_C(t)$:

$$H_C(t) = a\left[1 - \exp\left\{-s \cdot C(t)\right\}\right]. \tag{4.11}$$

We define the NHPP model with mean value function in Eq. (4.11) as an SRGM with the testing-coverage. It is noted that the mean value function with the testing-coverage in Eq. (4.11) has the following property:

$$\lim_{t\to\infty} H_C(t) = a\left(1 - \exp[-s \cdot \alpha]\right), \tag{4.12}$$

which implies that $H_C(t)$ in Eq. (4.11) does not converge on the initial fault content a in the software system even if $t \to \infty$. Therefore, $\{a - H_C(\infty)\}$ represents the expected total fault content to be detected on the other testing-coverage factors.

4.4 Parameter Estimation

We discuss methods of parameter estimation for the alternative testing-coverage function in Eq. (4.4) and the SRGM in Eq. (4.11), respectively. We suppose that K data pairs $(t_k,\ x_k,\ y_k)(k = 0, 1, 2, \cdots, K)$ with respect to the total number of detected faults, y_k, and the total attained testing-coverage, x_k, during the time-interval $(0,\ t_k](0 < t_1 < t_2 < \cdots < t_K)$ are observed.

We first discuss an estimation method for the alternative testing-coverage function in Eq. (4.4). The method of least-squares is applied to Eq. (4.1) transformed into an integrable difference equation. Concretely speaking, first we derive the following integrable difference equation via using the Hirota's bilinearization methods [14] from the differential equation in Eq. (4.1) with $\beta(t)$ in Eq. (4.3):

$$C_{n+1} - C_n = \delta r \alpha b_{sta} + \frac{\delta b_{sta}(1-2r)}{2}\left[C_n + C_{n+1}\right] - \frac{\delta b_{sta}(1-r)}{\alpha} C_n C_{n+1}. \tag{4.13}$$

Solving the above difference equation yields an exact solution of C_n representing the testing-coverage attained by n-th testing-period as

$$C_n = \frac{\alpha\left[1 - \left(\frac{1-\frac{1}{2}\delta b_{sta}}{1+\frac{1}{2}\delta b_{sta}}\right)^n\right]}{1 + z\left(\frac{1-\frac{1}{2}\delta b_{sta}}{1+\frac{1}{2}\delta b_{sta}}\right)^n} \qquad (z > 0,\ 0 \leq r \leq 1), \tag{4.14}$$

where δ represents the constant time-interval, that is, $t = n\delta$. We should note that Eq. (4.13) conserves the characteristics of the differential equation in Eq. (4.1) with $\beta(t)$ in Eq. (4.3). That is, the difference equation in Eq. (4.13) has an exact solution, and, as $\delta \to 0$, Eqs. (4.13) and (4.14) converge on the original differential equation in Eq. (4.1) with $\beta(t)$ in Eq. (4.3) and the exact solution in Eq. (4.4) of the differential equation, respectively. These properties above are features of the integrable difference equation derived by using the Hirota's bilinearization methods. From Eq. (4.13), a regression equation to get parameter estimates can be derived as

$$Y_n = A + B_1 K_n + B_2 L_n, \tag{4.15}$$

where

$$\begin{cases} Y_n &= C_{n+1} - C_n \\ K_n &= C_n + C_{n+1} \\ L_n &= C_n C_{n+1} \\ A &= \delta \alpha r b_{sta} \\ B_1 &= \delta b_{sta}\left(1 - 2r\right)/2 \\ B_2 &= -\delta b_{sta}\left(1 - r\right)/\alpha. \end{cases} \tag{4.16}$$

Using Eq. (4.15), we can estimate $\widehat{A}$, $\widehat{B_1}$, and $\widehat{B_2}$ by using the observed testing-coverage data, which are the estimates of A, B_1, and B_2, respectively. Therefore, we can obtain the parameter estimates $\widehat{\alpha}$, $\widehat{b_{sta}}$, and $\widehat{r}$ from Eq. (4.16) as follows:

$$\begin{cases} \widehat{\alpha} &= \widehat{A}\Big/\left(\sqrt{\widehat{B_1}^2 - \widehat{A}\widehat{B_2}} - \widehat{B_1}\right) \\ \widehat{b_{sta}} &= 2\sqrt{\widehat{B_1}^2 - \widehat{A}\widehat{B_2}}\Big/\delta \\ \widehat{r} &= \left(1 - \widehat{B_1}/\sqrt{\widehat{B_1}^2 - \widehat{A}\widehat{B_2}}\right)\Big/2. \end{cases} \tag{4.17}$$

Y_n, K_n, and L_n in Eq. (4.15) are independent of δ because δ is not used in calculating Y_n, K_n, and L_n in Eq. (4.15). Hence, we can obtain the same parameter estimates $\widehat{\alpha}$, $\widehat{b_{sta}}$, and $\widehat{r}$ which are the estimated values α, b_{sta}, and r, respectively, when we choose any value of δ. It is said that this method can get more accurate parameter estimates than the ordinary method of least-squares.

Second we discuss an estimation method for the mean value function in Eq. (4.11). We use the method of maximum-likelihood to get the parameter estimates, $\widehat{a}$ and $\widehat{s}$ which are the estimated values of a and s, respectively, in the mean value function. Then, the logarithmic likelihood function is given as

$$\ln L = \sum_{k=1}^{K}(y_k - y_{k-1}) \cdot \ln[H_C(t_k) - H_C(t_{k-1})] - H_C(t_K) - \sum_{k=1}^{K}\ln[y_k - y_{k-1}], \tag{4.18}$$

from the properties of the NHPP. Furthermore, we can derive the following simultaneous equations by partially differentiating the logarithmic likelihood function, $\ln L$, with respect to parameters a and s:

$$\frac{\partial \ln L}{\partial a} = \frac{\partial \ln L}{\partial s} = 0. \tag{4.19}$$

By solving the above simultaneous equations numerically, we can estimate $\widehat{a}$ and $\widehat{s}$ which are the estimates of a and s, respectively.

4.5 Software Reliability Assessment Measures

In this section we derive several software reliability assessment measures which are useful for quantitative assessment of software reliability and the progress of the software testing. Specifically, we derive a software reliability function, instantaneous and cumulative mean times between software failures (abbreviated as MTBF's).

4.5.1 Software reliability function

Given that the testing or the user operation has been going up to time t, the probability that a software failure does not occur in the time-interval $(t,\ t+x](x \geq 0,\ t \geq 0)$ is derived as

$$R(x \mid t) = \exp[-\{H(t+x) - H(t)\}], \tag{4.20}$$

from the properties of the Eq. (4.7). $R(x \mid t)$ in Eq. (4.20) is called a software reliability function. We can estimate the software reliability by using this equation.

4.5.2 Instantaneous MTBF

We discuss an instantaneous MTBF which has been used as one of the substitution for MTBF. An instantaneous MTBF is approximately given by

$$\mathrm{MTBF}_I(t) = \frac{1}{h(t)}. \tag{4.21}$$

4.5.3 Cumulative MTBF

A cumulative MTBF is also the substitution for the MTBF. The cumulative MTBF is approximately derived as

$$\mathrm{MTBF}_C(t) = \frac{t}{H(t)}. \tag{4.22}$$

If the instantaneous MTBF in Eq. (4.21) and the cumulative MTBF in Eq. (4.22) take on a large value, respectively, then we decide that the software system becomes more reliable.

4.6 Numerical Examples

For evaluating the performance of our model, we show numerical examples and a result of goodness-of-fit test for our model by using C0 testing-coverage measure data recorded along

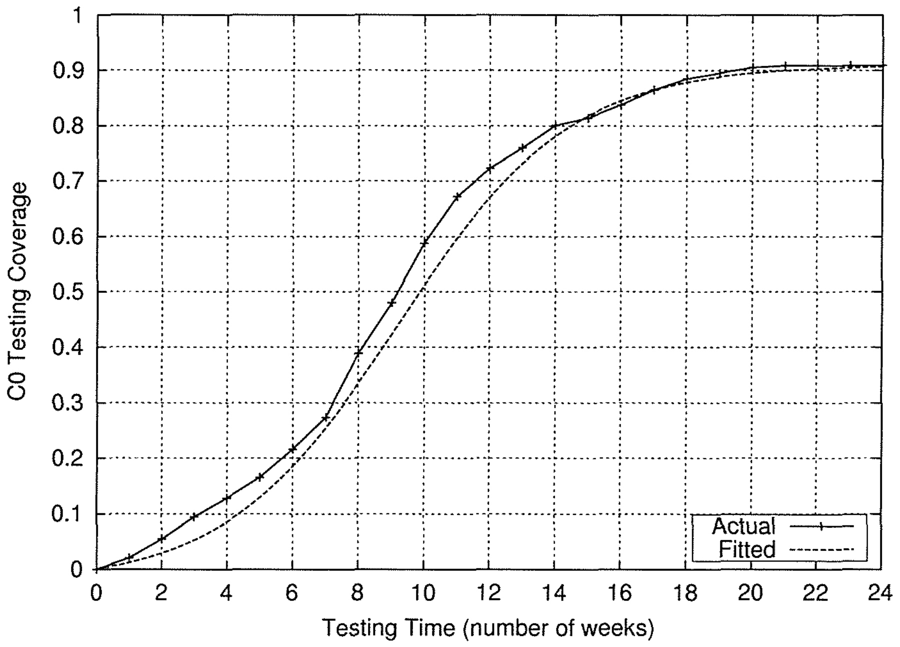


Fig. 4.1 : The estimated alternative testing-coverage function (on the C0 testing-coverage measure).

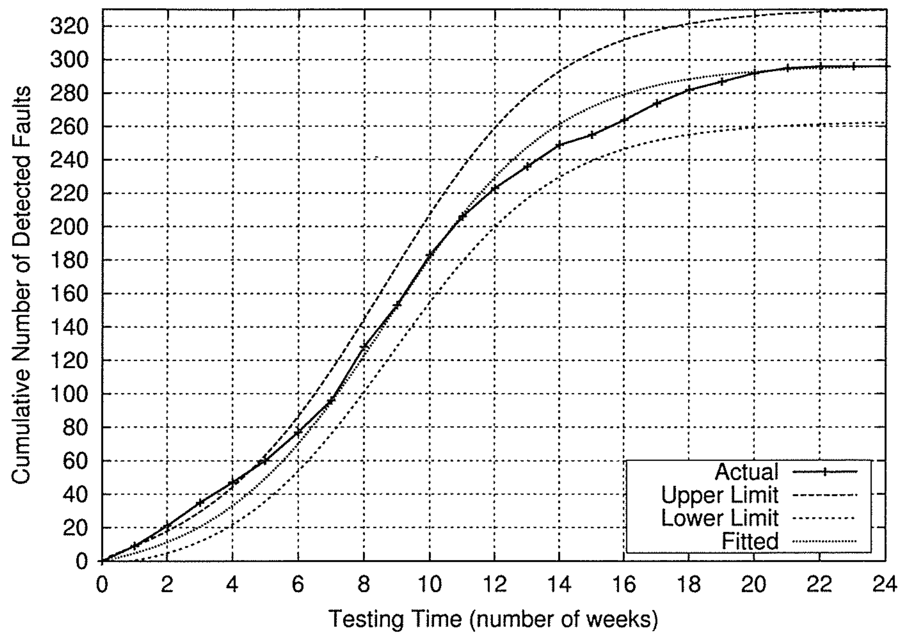


Fig. 4.2 : The estimated mean value function with its 95% confidence limits.

with fault count data collected from a practical software development project for an embedded software system. There are totally 296 faults detected and 90.6% of the C0 testing-coverage measure attained within 24 weeks.

Fig. 4.1 shows the estimated alternative testing-coverage function $\widehat{C}(t)$ of Eq. (4.4) in which the parameter estimates are obtained as $\widehat{\alpha} = 90.796$, $\widehat{b_{sta}} = 0.388$, and $\widehat{z} = 52.338$. From Fig. 4.1, we can see that the estimated alternative testing-coverage function fits well to the actual C0 testing-coverage measure data. Next, Fig. 4.2 shows the estimated mean value function $\widehat{H_C}(t)$ of Eq. (4.11) and its 95% confidence limits, where the parameter estimates of $\widehat{H_C}(t)$ are obtained as $\widehat{a} = 919.9$ and $\widehat{s} = 0.4282$. And the $100\gamma\%$ confidence limits for $\widehat{H_C}(t)$ are derived as

$$\widehat{H_C}(t) \pm K_\gamma \sqrt{\widehat{H_C}(t)}, \tag{4.23}$$

where K_γ indicates the $100(1+\gamma)/2$ percent point of the standard normal distribution. Eq. (4.23) is derived by using the asymptotic normality of the maximum-likelihood estimates [58].

Furthermore, Fig. 4.3 depicts the estimated software reliability function at the termination of the testing in Eq. (4.20). If we assume that the developed software system is used in the operation phase which has the same environment as the testing-phase, we can estimate the software reliability after 3 weeks from the termination time of the testing, $R(3 \mid 24)$, to be

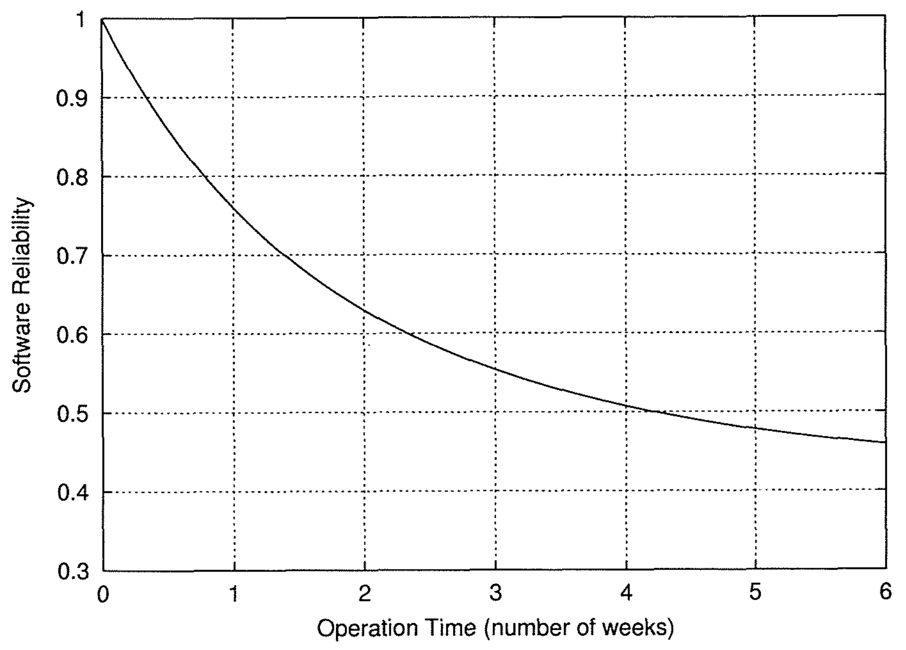


Fig. 4.3 : The estimated software reliability.

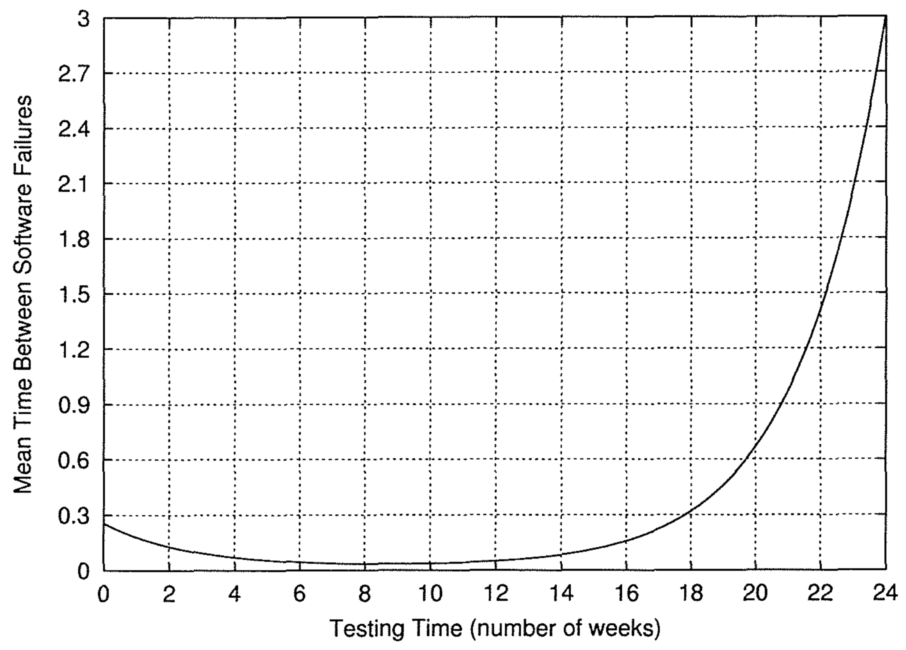


Fig. 4.4 : The estimated instantaneous MTBF.

about $0 \cdot 554$. And Fig. 4.4 shows the estimated instantaneous MTBF. From Fig. 4.4, we can estimate the instantaneous MTBF at the termination time of the testing, $\widehat{\mathrm{MTBF}}_I(24)$, to be about $3 \cdot 024$ (weeks).

4.7 Goodness-of-Fit Test

We conduct a statistical goodness-of-fit test of our model for the observed fault count data. That is, we compare the observed empirical (or sample) distribution with the theoretical distribution. In the fields of software reliability growth modeling, *Chi-Squared* (χ^2) *Test* and *Kolmogorov-Smirnov Test* have been used as the nonparametric goodness-of-fit test techniques [4,38,51]. The χ^2 goodness-of-fit test is applicable to both continuous-time and discrete-time theoretical distributions, and be used when the parameters of the distribution are estimated based on the method of maximum-likelihood. However, this test is valid only for large sample size since this test assumes large sample normality of the observed number of data pairs. On the other hand, the Kolmogorov-Smirnov (abbreviated as K-S) goodness-of-fit test assumes a continuous-time theoretical distribution only, and is valid for both small and large sample size. Therefore, we apply the K-S goodness-of-fit test as the goodness-of-fit technique in this chapter.

The K-S goodness-of-fit test is conducted along with the following procedure [38,50–52,56]. Suppose that $F(x)$ is a continuous-time theoretical distribution function of the random variable X and $(x_1, x_2, \cdots, x_n\,;\, x_1 \le x_2 \le \cdots \le x_n)$ are the order statistics which are the realization of X. Then, the K-S test statistic, D, is given by

$$\left.\begin{aligned} D &= \max_{1\le i\le n} D_i \\ D_i &= \max\left\{\left|F(x_i) - \tfrac{i}{n}\right|,\ \left|F(x_i) - \tfrac{i-1}{n}\right|\right\} \end{aligned}\right\}, \tag{4.24}$$

where, $F(x_i)$ corresponds to

$$F(x_i) = \frac{H(x_i)}{H(x_n)}, \tag{4.25}$$

in the case of the NHPP model with the mean value function $H(t)$ in Eq. (4.7). Thus, supposing that we have observed n data pairs $(t_i,\ y_i)(i = 0, 1, 2, \cdots, n)$ with respect to the total faults, y_i, detected during constant time-interval $(0,\ t_i](0 < t_1 < t_2 < \cdots < t_n)$, the K-S test statistic, D, can be rewritten as

$$\left.\begin{aligned} D &= \max_{1\le i\le n} D_i \\ D_i &= \max\left\{\left|\tfrac{H(t_i)}{H(t_n)} - \tfrac{y_i}{y_n}\right|,\ \left|\tfrac{H(t_i)}{H(t_n)} - \tfrac{y_{i-1}}{y_n}\right|\right\} \end{aligned}\right\}. \tag{4.26}$$

The K-S test statistic D is needed to compared with a critical value $d_{n;\alpha}$, where n represents the number of data pairs and α a level of significance which is ordinally given as $0 \cdot 01$ or $0 \cdot 05$. That is, we judge that the applied NHPP model fits to the observed data at a level of significance α if $D < d_{n;\alpha}$. A table of the critical values $d_{n;\alpha}$ are provided in the related books (see [38, 50–52, 56]).

The result of the goodness-of-fit test based on the K-S goodness-of-fit test of our model for the observed data introduced in Section 4.6 can be obtained as $D = 0 \cdot 0947 < d_{24;0.05} = 0 \cdot 2693$. Then, we can verify that $\widehat{H_C}(t)$ fits to the applied observed data at the 5% level of significance from the result of the K-S goodness-of-fit testing.

4.8 Concluding Remarks

We have discussed software reliability growth modeling with testing-coverage which is one of the key factors related to the software reliability growth process in this chapter. And we have also discussed parameter estimation methods of our models. Specifically, we have obtained the parameter estimates of the alternative testing-coverage function by using the regression analysis based on the integrable difference equation derived from the original differential equation. Then, we have derived several software reliability assessment measures, such as the software reliability function, the instantaneous and cumulative MTBF's. After that, we have shown numerical examples of the estimated alternative testing-coverage function and the estimated our SRGM by using C0 testing-coverage measure data recorded along with fault count data collected in an actual testing-phase. Finally, we have conducted a statistical goodness-of-fit test of our model for the actual data based on the K-S goodness-of-fit test.

Chapter 5

Lognormal Process Software Reliability Modeling with Testing-Effort

5.1 Introduction

Quantitative software reliability assessment is one of the most important issues to produce reliable software systems. An SRGM has been utilized to assess software reliability quantitatively since 1970's. The SRGM can describe a software fault-detection phenomenon or a software failure-occurrence phenomenon in the testing or operational phase by applying stochastic and statistical theories. Especially, an NHPP which treats the fault-detection phenomenon as a discrete-state space has been often applied to software reliability growth modeling. The NHPP model enables us to characterize software reliability growth process simply by supposing an appropriate mean value function of the NHPP.

In contrast with discrete-state space SRGM's such as NHPP models, continuous-state space SRGM's to assess software reliability for large scale software systems have been proposed so far. Specifically, Yamada et al. [71] have discussed a framework of continuous-state space software reliability growth modeling based on stochastic differential equations of Itô type, and have compared the continuous-state space SRGM with the NHPP models. Recently, Yamada et al. [65] have proposed several SRGM's based on stochastic differential equations of Itô type, such as exponential, delayed S-shaped, inflection S-shaped stochastic differential equation models, of which the fault-detection rates per unit time per one fault have been characterized by using basic assumptions of existing exponential, delayed S-shaped, and inflection S-shaped SRGM's, respectively. And, Tamura et al. [49] have proposed continuous-state space SRGM for distributed development environment and its parameter estimation.

However, these continuous-state space SRGM's have not taken the effect of testing-effort into consideration. The testing-effort, such as number of executed test-cases, attained testing-coverage, or CPU hours expended in the testing-phase, is well-known as one of the most important factors related to the software reliability growth process. Yamada et al. [67] has proposed a testing-effort dependent SRGM's based on the NHPP's. Under the above background, there is necessity to discuss a testing-effort dependent SRGM on a continuous-state space for the purpose of developing a plausible continuous-state space SRGM.

This chapter proposes a continuous-state space software reliability growth model with a testing-effort factor by applying a mathematical technique of stochastic differential equations of Itô type, and conducts goodness-of-fit comparisons between our model and existing continuous-state space SRGM's. First we extend a basic differential equation describing the behavior of the cumulative number of detected faults to stochastic differential equations of Itô type by considering with the testing-effort expenditures through the testing-phase, and derive its solution process which represents the fault-detection process. Then, we discuss estimation methods for unknown parameters in our models. And we then compare our model with existing continuous-state space SRGM's by using several goodness-of-fit evaluation criteria. Finally, we derive software reliability assessment measures based on a probability distribution of the solution process, and show numerical examples for derived software reliability assessment measures by using an actual fault count data.

5.2 Framework of Modeling

In this section we discuss a framework of continuous-state space software reliability growth modeling. Letting $N(t)$ be a random variable which represents the number of faults detected up to time t, we can derive the following linear differential equation from the common assumptions for software reliability growth modeling [29, 38, 54]:

$$\frac{dN(t)}{dt} = b(t)\{a - N(t)\} \qquad (a > 0,\, b(t) > 0), \tag{5.1}$$

where $b(t)$ indicates the fault-detection rate at testing-time t and is assumed to be a non-negative function, and a the initial fault content in the software system. Eq. (5.1) describes the behavior of the decrement of the fault content in the software system.

Especially, in large-scale software development, the fault-detection process in the actual testing-phase is influenced by several uncertain testing-factors such as testing-skill and debugging environment. Accordingly, we should take these uncertain factors into consideration in software reliability growth modeling. Thus, we extend Eq. (5.1) to the following equation:

$$\frac{dN(t)}{dt} = \{b(t) + \xi(t)\}\{a - N(t)\}, \tag{5.2}$$

where $\xi(t)$ is a noise that describes an irregular fluctuation. For the purpose of making its solution a Markov process, we assume that $\xi(t)$ in Eq. (5.2) is given as

$$\xi(t) = \sigma\gamma(t) \qquad (\sigma > 0), \tag{5.3}$$

where σ indicates a positive constant representing the magnitude of the irregular fluctuation and γ a standardized Gaussian white noise.

We transform Eq. (5.2) into the following stochastic differential equation of Itô type:

$$dN(t) = \{b(t) - \frac{1}{2}\sigma^2\}\{a - N(t)\}dt + \sigma\{a - N(t)\}dW(t), \tag{5.4}$$

where $W(t)$ is a one-dimensional Wiener process which is formally defined as an integration of the white noise $\gamma(t)$ with respect to time t. The Wiener process $W(t)$ is a Gaussian process, and has the following properties:

(a) $\Pr[W(0) = 0] = 1$,

(b) $\mathrm{E}[W(t)] = 0$,

(c) $\mathrm{E}[W(t)W(t')] = \min[t, t']$,

where $\Pr[\,\cdot\,]$ and $\mathrm{E}[\,\cdot\,]$ represent the probability and expectation, respectively. Next, we derive a solution process $N(t)$ by using the Itô's formula. The solution process $N(t)$ can be derived as

$$N(t) = a\left[1 - \exp\left\{-\int_0^t b(\tau)d\tau - \sigma W(t)\right\}\right]. \tag{5.5}$$

Eq. (5.5) implies that the solution process $N(t)$ obeys a geometric Brownian motion or a lognormal process [21,26,37]. And the transition probability distribution of the solution process $N(t)$ is derived as

$$\Pr[N(t) \le n | N(0) = 0] = \Phi\left(\frac{\log\frac{a}{a-n} - \int_0^t b(\tau)d\tau}{\sigma\sqrt{t}}\right), \tag{5.6}$$

consequently, by the properties (a)–(c) and the assumption that $W(t)$ is a Gaussian process. $\Phi(\cdot)$ in Eq. (5.6) indicates a standardized normal distribution function defined as

$$\Phi(x) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x} \exp(-\frac{y^2}{2})dy. \tag{5.7}$$

By assuming a specific appropriate function $b(t)$ in Eq. (5.5) which characterizes the software reliability growth process, we can derive several SRGM's.

5.3 Lognormal Process SRGM with Testing-effort

We develop a continuous-state space SRGM with the effect of testing-effort based on stochastic differential equations which obey the lognormal process in this chapter. The testing-effort, such as the number of executed test-cases, attained testing-coverage, or CPU hours expended in the testing-phase, is one of the important factors which influence a software reliability growth process in an actual testing phase. Therefore, the testing-effort should be taken into consideration in software reliability growth modeling.

5.3.1 Modeling

For the purpose of developing a continuous-state space SRGM with the effect of the testing-effort, we characterize $b(t)$ in Eq. (5.5) as

$$b(t) \equiv b_T(t) = r \cdot s(t) \qquad (0 < r < 1), \tag{5.8}$$

where r represents the fault-detection rate per expended testing-effort at testing-time t and $s(t) \equiv dS(t)/dt$ in which $S(t)$ is the amount of testing-effort expended by arbitrary testing-time t. Then, based on the framework of continuous-state space modeling in the previous section, we can obtain the following solution process:

$$\begin{aligned} N(t) &\equiv N_T(t) \\ &= a\left[1 - \exp\left\{-r\int_0^t s(\tau)d\tau - \sigma W(t)\right\}\right] \\ &= a\left[1 - \exp\left\{-rS(t) - \sigma W(t)\right\}\right]. \end{aligned} \tag{5.9}$$

The transition probability distribution function of the solution process in Eq. (5.9) can be derived as

$$\Pr[N_T(t) \le n | N_T(0) = 0] = \Phi\left(\frac{\log\frac{a}{a-n} - rS(t)}{\sigma\sqrt{t}}\right). \tag{5.10}$$

We should specify the testing-effort function $s(t)$ in Eq. (5.8) to utilize the solution process $N_T(t)$ in Eq. (5.9) as an SRGM.

5.3.2 Testing-effort function

We need to specify a suitable function $s(t)$ in Eq. (5.8) to describe the time-dependent behavior of testing-effort expenditures in the testing-phase. In this chapter we apply a Weibull curve to the testing-effort function $s(t)$. The Weibull curve is formulated as

$$s(t) = \alpha\beta m t^{m-1} \exp\{-\beta t^m\} \qquad (\alpha > 0,\ \beta > 0,\ m > 0). \tag{5.11}$$

Then,

$$S(t) = \int_0^t s(\tau)d\tau = \alpha\left[1 - \exp\{-\beta t^m\}\right], \tag{5.12}$$

where α is the total amount of testing-effort expenditures, β the scale parameter, and m the shape parameter characterizing the shape of the testing-effort function.

The Weibull curve has a useful property to describe the time-dependent behavior of the expended testing-effort expenditures during the testing-phase approximately. We can obtain the exponential curves when $m = 1$ in Eqs. (5.11) and (5.12). And when $m = 2$, we can derive Rayleigh curves. Accordingly, we can see that the Weibull curve is a useful function as a testing-effort function which can describe the time-dependent behavior of the testing-effort expenditures through the testing-phase flexibly.

5.4 Estimation Methods for Unknown Parameters

We discuss estimation methods of unknown parameters of the testing-effort function in Eq. (5.11) and the solution process in Eq. (5.9), respectively. Suppose that K data pairs $(t_j,\ y_j,\ n_j)(j = 0, 1, 2, \cdots, K)$ with respect to the total number of faults, n_j, detected during the time-interval $(0,\ t_j](0 < t_1 < t_2 < \cdots < t_K)$, and the amount of testing-effort expenditures, y_j, expended at t_j are observed.

5.4.1 Testing-effort function

As to a parameter estimation method for the testing-effort function in Eq. (5.11), we apply the method of least squares. First we take the natural logarithm of Eq. (5.11) as

$$\log s(t) = \log\alpha + \log\beta + \log m + (m-1)\log t - \beta t^m. \tag{5.13}$$

Then, the sum of the squares of vertical distances from the data points to the estimated values is formulated as

$$S(\alpha,\ \beta,\ m) = \sum_{j=1}^{K}\{\log y_j - \log s(t_j)\}^2. \tag{5.14}$$

The parameter estimates $\widehat{\alpha}$, $\widehat{\beta}$, and $\widehat{m}$ of the parameter α, β, and m which minimize $S(\alpha,\ \beta,\ m)$ in Eq. (5.14) can be obtained by solving the following simultaneous equations:

$$\frac{\partial S}{\partial \alpha} = \frac{\partial S}{\partial \beta} = \frac{\partial S}{\partial m} = 0. \tag{5.15}$$

5.4.2 Solution process

Next we discuss a parameter estimation method for the solution process in Eq. (5.9) by using the method of maximum-likelihood. Let us denote the joint probability distribution function of the process $N_T(t)$ as

$$\begin{aligned} &P(t_1, n_1; t_2, n_2; \cdots ; t_K, n_K) = \\ &\qquad \Pr[N_T(t_1) \le n_1, N_T(t_2) \le n_2, \cdots N_T(t_K) \le n_K | N_T(0) = 0], \end{aligned} \tag{5.16}$$

and also denote its density as

$$p(t_1, n_1; t_2, n_2; \cdots ; t_K, n_K) = \frac{\partial^K P(t_1, n_1; t_2, n_2; \cdots ; t_K, n_K)}{\partial n_1 \partial n_2 \cdots \partial n_K}. \tag{5.17}$$

Then, we can constract the likelihood function l for the observed data pairs $(t_j, n_j)(j = 0, 1, 2, \cdots , K)$ as

$$l = p(t_1, n_1; t_2, n_2; \cdots ; t_K, n_K). \tag{5.18}$$

For convenience in mathematical manipulations, we need to derive the logarithmic likelihood function by taking the natural logarithm of Eq. (5.18). The logarithmic likelihood function is denoted by

$$L \equiv \log l. \tag{5.19}$$

The likelihood function l in Eq. (5.18) can be reduced to the following equation by using the Bayes' formula and a Markov property [36, 50]:

$$l = \prod_{j=1}^{K} p(t_j, n_j \mid t_{j-1}, n_{j-1}), \tag{5.20}$$

where $p(\cdot \mid t_0, n_0)$ is the conditional probability density under the condition of $N_T(t_0) = n_0$. The transition density $p(t_j, n_j \mid t_{j-1}, n_{j-1})$ in Eq. (5.20) can be obtained by partially differentiating the following transition probability of $N_T(t)$ under the condition $N_T(t_{j-1}) = n_{j-1}$,

$$\Pr[N_T(t_j) \le n_j | N_T(t_{j-1}) = n_{j-1}] = \Phi\left(\frac{\log\left(\frac{a - n_{j-1}}{a - n_j}\right) - r\{S(t_j) - S(t_{j-1})\}}{\sigma\sqrt{t_j - t_{j-1}}}\right), \tag{5.21}$$

with respect to n_j. Consequently, the likelihood function l in Eq. (5.20) can be rewritten as

$$l = \prod_{j=1}^{K} \frac{1}{(a-n_j)\sigma\sqrt{2\pi(t_j - t_{j-1})}} \cdot \exp\left[-\frac{\left[\log\left(\frac{a-n_{j-1}}{a-n_j}\right) - r\{S(t_j) - S(t_{j-1})\}\right]^2}{2\sigma^2(t_j - t_{j-1})} \right], \tag{5.22}$$

and the logarithmic likelihood function is then derived as

$$L = -\sum_{j=1}^{K} \log(a-n_j) - K\log\sigma - \frac{K}{2}\log 2\pi - \frac{1}{2}\log(t_j - t_{j-1})$$
$$- \frac{1}{2\sigma^2}\sum_{j=1}^{K} \frac{\left[\log\left(\frac{a-n_{j-1}}{a-n_j}\right) - r\{S(t_j) - S(t_{j-1})\}\right]^2}{t_j - t_{j-1}}. \tag{5.23}$$

From Eq. (5.23), we can obtain the following simultaneous logarithmic likilihood equations:

$$\frac{\partial L}{\partial a} = -\sum_{j=1}^{K}\frac{1}{a-n_j} + \frac{1}{\sigma^2}\sum_{j=1}^{K}\frac{(n_j - n_{j-1})\left[\log\left(\frac{a-n_{j-1}}{a-n_j}\right) - r\{S(t_j) - S(t_{j-1})\}\right]}{(t_j - t_{j-1})(a-n_j)(a-n_{j-1})} = 0, \tag{5.24}$$

$$\frac{\partial L}{\partial r} = \frac{1}{\sigma^2 t_K}\left[r\sum_{j=1}^{K}\{S(t_j) - S(t_{j-1})\}^2 - \sum_{j=1}^{K}\{S(t_j) - S(t_{j-1})\}\log\left(\frac{a-n_{j-1}}{a-n_j}\right)\right] = 0, \tag{5.25}$$

$$\frac{\partial L}{\partial \sigma} = -\frac{K}{\sigma} + \frac{1}{\sigma^3}\sum_{j=1}^{K}\frac{\left[\log\left(\frac{a-n_{j-1}}{a-n_j}\right) - r\{S(t_j) - S(t_{j-1})\}\right]^2}{t_j - t_{j-1}} = 0. \tag{5.26}$$

Eqs. (5.25) and (5.26) can be transformed into

$$r = \sum_{j=1}^{K}\frac{\{S(t_j) - S(t_{j-1})\}\log\left(\frac{a-n_{j-1}}{a-n_j}\right)}{\{S(t_j) - S(t_{j-1})\}^2}, \tag{5.27}$$

$$\sigma^2 = \frac{1}{K}\sum_{j=1}^{K}\frac{\left[\log\left(\frac{a-n_{j-1}}{a-n_j}\right) - r\{S(t_j) - S(t_{j-1})\}\right]^2}{t_j - t_{j-1}}, \tag{5.28}$$

respectively. Therefore, we can eliminate the two parameters, r and σ, in Eq. (5.24) by substituting Eqs. (5.27) and (5.28) into Eq. (5.24). Consequently, the maximum likelihood esitmates $\widehat{a}$, $\widehat{r}$, and $\widehat{\sigma}$ of the parameters a, r, and σ by solving the nonlinear equations with one variable, respectively.

5.5 Software Reliability Assessment Measures

In this section we derive several software reliability assessment measures which are useful for quantitative assessment of software reliability and the progress control of the software testing-process. Specifically, we derive instantaneous and cumulative MTBF's in this chapter.

5.5.1 Instantaneous MTBF

We discuss an instantaneous MTBF which has been used as one of the substitution for an MTBF. The instantaneous MTBF is approximately given by

$$\mathrm{MTBF}_I(t) = \frac{dt}{\mathrm{E}[dN_T(t)]}. \tag{5.29}$$

We need to derive $\mathrm{E}[N_T(t)]$ which represents the expected number of faults detected up to arbitrary testing-time t to obtain $\mathrm{E}[dN_T(t)]$ in Eq. (5.29). By noting that $W(t) \sim N(0,\, t)$, the expected number of faults detected up to arbitrary testing-time t is obtained as

$$\mathrm{E}[N_T(t)] = a\left[1 - \exp\left\{-(rS(t) - \frac{1}{2}\sigma^2 t)\right\}\right]. \tag{5.30}$$

Since the Wiener process has the independent increment property, $W(t)$ and $dW(t)$ are statistically independent with each other, and $\mathrm{E}[dW(t)] = 0$, $\mathrm{E}[dN_T(t)]$ in Eq. (5.29) is finally derived as

$$\mathrm{E}[dN_T(t)] = a\{rs(t) - \frac{1}{2}\sigma^2\}\exp\{-(rS(t) - \frac{1}{2}\sigma^2 t)\}dt. \tag{5.31}$$

The instantaneous MTBF in Eq. (5.29) can be calculated by substituting Eq. (5.31) into Eq. (5.29).

5.5.2 Cumulative MTBF

A cumulative MTBF is also the substitution for the MTBF. The cumulative MTBF is approximately derived as

$$\mathrm{MTBF}_C(t) = \frac{t}{\mathrm{E}[N_T(t)]}. \tag{5.32}$$

If the instantaneous MTBF in Eq. (5.29) and the cumulative MTBF in Eq. (5.32) take on large values, respectively, then we decide that the software system becomes more reliable.

Table 5.1 : The results of model comparisons.

		Proposed model	Exponential SDE model	Delayed S-shaped SDE model	Inflection S-shaped SDE model
MSE	DS1	**1367.63**	22528	6018.65	6550.37
	DS2	1370.8	**1332.34**	36549	1986.8
AIC	DS1	**306.15**	325.32	315.98	318.57
	DS2	125.51	125.18	131.65	126.47

(SDE : stochastic differential equation)

5.6 Model Comparisons

We show results of goodness-of-fit comparisons between our model and other continuous-state space SRGM's [65], such as exponential, delayed S-shaped, and inflection S-shaped stochastic differential equations, in terms of the MSE and AIC introduced in Section 2.4. As to the goodness-of-fit comparisons, we use two actual data sets [6] named as DS1 and DS2, respectively. DS1 and DS2 indicate an S-shaped and exponential reliability growth curves, respectively.

Table 5.1 shows the results of model comparisons based on the MSE and the AIC, respectively. However the model comparisons based on the AIC is not significant only for DS2, we can see that our model improves performance of the MSE and the AIC as compared with other continuous-state space SRGM's used in these goodness-of-fit comparisons in this section.

5.7 Numerical Examples

We show numerical examples by using testing-effort data recorded along with detected fault count data collected from the actual testing. In this testing, 1301 faults are totally detected and 1846.92 (testing-hours) are totally expended as the testing-effort within 35 months [6].

Fig. 5.1 shows the estimated testing-effort function $\widehat{s}(t)$ in Eq. (5.11) in which the parameter estimates are obtained as $\widehat{\alpha} = 2253.2$, $\widehat{\beta} = 4.5343 \times 10^{-4}$, and $\widehat{m} = 2.2580$. Fig. 5.2 shows the estimated expected number of detected faults in Eq. (5.30) where the parameter estimates are obtained as $\widehat{a} = 1435.3$, $\widehat{r} = 1.4122 \times 10^{-3}$, and $\widehat{\sigma} = 3.4524 \times 10^{-2}$. Furthermore, Fig. 5.3 shows the time-depedent behavior of the estimated instantaneous and cumulative MTBF's in Eqs. (5.29) and (5.32), respectively. From Fig. 5.3, we can see that the software reliability decreases in the early testing-phase, and then, it grows as the testing procedures go on. We can

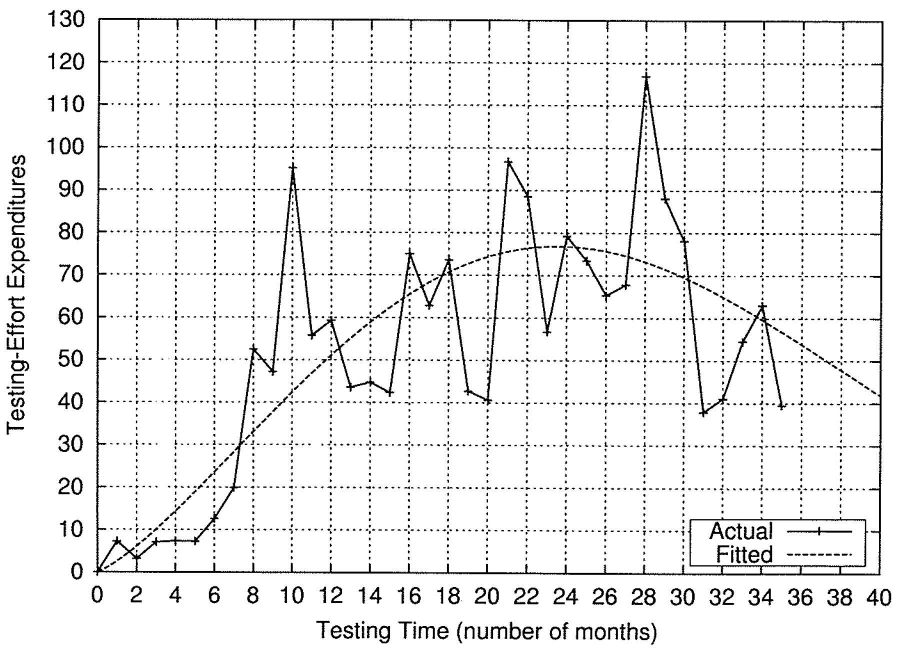


Fig. 5.1 : The estimated testing-effort function.

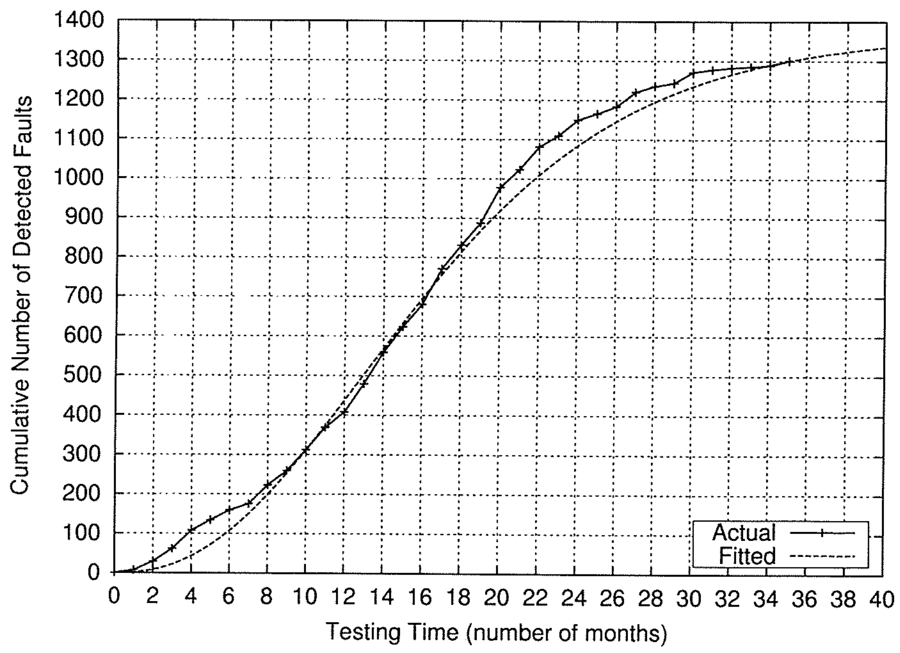


Fig. 5.2 : The estimated expected number of detected faults.

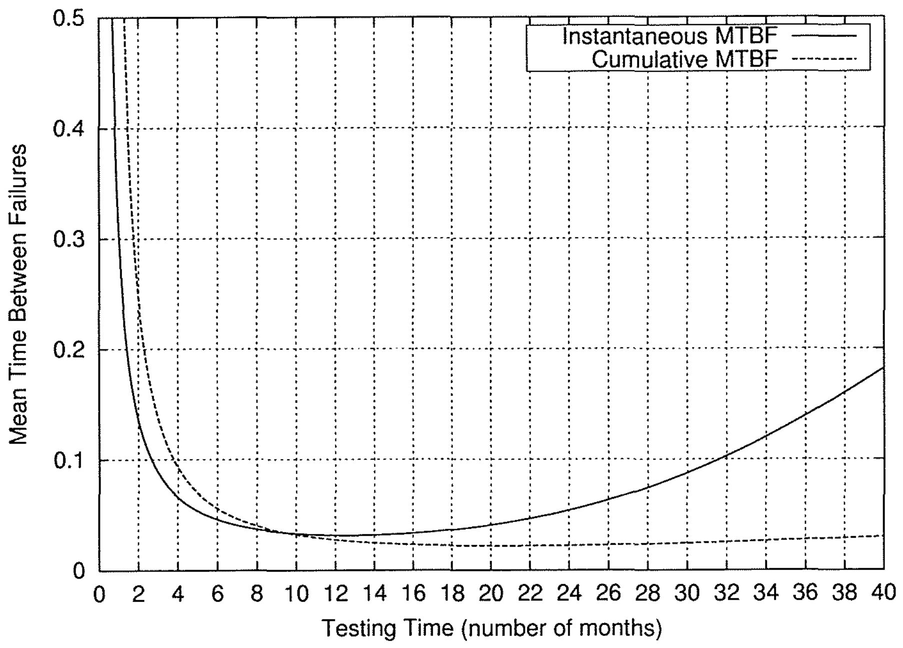


Fig. 5.3 : The estimated instantaneous and cumulative MTBF's.

estimate the instantaneous MTBF at the termination time of the testing, $\mathrm{MTBF}_I(35)$, to be about 0.1297 (about 4.5 months), and the cumulative MTBF, $\mathrm{MTBF}_C(35)$, to be about 0.0269 (about 0.9 months).

5.8 Concluding Remarks

In this chapter we have discussed continuous-state space software reliability growth modeling based on a lognormal process with the effect of testing-effort by using a mathematical technique of stochastic differential equations of Itô type and estimation methods for the parameters of the testing-effort function and the solution process, respectively. Then, we have conducted goodness-of-fit comparisons between our model and existing continuous-state space SRGM's in terms of the MSE and the AIC by using actual data sets, respectively. Finally, we have also shown numerical illustrations for the software reliability assessment measures such as the instantaneous and cumulative MTBF's. We believe that software development managers can grasp the relationship between the attained software reliability and the testing-effort expenditures through the testing-phase by using our software reliability growth model, and our model also enables software development managers to decide how much testing-effort are expended to attain the reliability objective.

Part III

GENERALIZED MODELING

Chapter 6

Generalized Discrete Software Reliability Modeling with Program Size

6.1 Introduction

Assessing software reliability in a testing-phase is one of the important issues to develop a highly reliable software system. During the testing-phase, an implemented software system is tested to detect and correct faults latent in the software system. We can describe the software failure-occurrence or fault-detection phenomenon by analyzing the related actual data collected in the testing-phase. An SRGM is known as a useful mathematical tool to describe these phenomena above in the testing-phase, to assess software reliability quantitatively, to decide the time to release for operational use, and to evaluate the maintenance cost for faults undetected during the testing-phase.

As a role of software systems is expanding rapidly, the size, complexity, and diversification of software systems are growing drastically in recent years. Accordingly, we need to develop more plausible SRGM's which enable us to assess software reliability more accurately. As one of the solutions, generalized approach for software reliability growth modeling have been proposed so far based on order-statistics [24], infinite server queueing theory [8], Markov processes [23,29,48], and so on. Especially, Langberg and Singpurwalla [24] have proposed a generalized SRGM by using the assumption that the fault-detection times can be regard as order-statistics. And they have also discussed that several NHPP models can be classified by the fault-detection times distribution. These generalized SRGM's has a useful characteristic that we can easily obtain a suitable SRGM by reflecting the software failure-occurrence or fault-detection phenomenon to the generalized assumptions. However, most of the generalized SRGM's have been discussed in terms of continuous-time SRGM's. Considering that there are discrete software reliability growth models to describe the software reliability growth process depending on discrete testing-time such as the number of days (or weeks), the number of executed test-cases [60], we need to discuss a generalized discrete software reliability growth modeling approach. In recent researches, Huang et al. [15] have discussed a unified scheme of discrete NHPP models by applying the concept of weighted arithmetic, weighted geometric, or weighted harmonic means. Dohi et al. [9] have proposed a generalized discrete software reliability model based on the idea of cumulative Bernoulli trials. And, Okamura et al. [34] have discussed a unified framework of discrete NHPP modeling based on the concepts of cumulative binomial trials and order-statistics.

In this chapter we discuss a unified framework for discrete software reliability growth modeling in which the software failure-occurrence times obey a discrete-time probability distribution. Based on the framework, we then discuss a generalized discrete SRGM which enables us to assess software reliability in consideration of the effect of the program size. We can consider that the effect of program size is one of the important factor influencing not only the testing-coverage maturity process and also the software reliability growth process. And we derive several generalized software reliability assessment measures based on the concept of the generalization framework. After that, we discuss parameter estimation based on the method of maximum-likelihood for our generalized discrete SRGM. Additionally, we discuss optimal software release problems with simultaneous cost and reliability objectives based on our generalized discrete SRGM. Finally, we depict numerical illustrations of our generalized discrete model and its application to derived optimal release policies by using actual fault count data.

6.2 Generalized Modeling

We discuss a unified framework for discrete software reliability growth modeling in which the probability distribution of each fault-detection time obey a discrete-time probability distribution. Based on the framework, we develop a generalized discrete binomial process model with the effect of the program size.

6.2.1 Unified framework

In a testing-phase the software failure-occurrence times can be regarded as order-statistics. Okamura et al. [34] have discussed a unified framework for discrete software reliability growth modeling based on the order-statistics. The unified framework is based on the following assumptions:

(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-detection procedure,

(A2) Each software failure occurs at independently and identically distributed time with the discrete probability distribution $P(i)(i = 0, 1, 2, \cdots)$,

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

We can develop a generalized discrete SRGM based on the assumptions above. First, let $\{N(i),\ i = 0, 1, \cdots\}$ denote a discrete stochastic process representing the number of faults detected up to i-th testing-period. Then, the conditional probability that m faults are detected up to i-th testing-period given that $N_0 = n$ is derived as

$$\Pr\{N(i) = m \mid N_0 = n\} = \binom{n}{m} \{P(i)\}^m \{1 - P(i)\}^{n-m}. \tag{6.1}$$

Accordingly, we have the probability mass function that m faults are detected up to i-th testing-period as

$$\Pr\{N(i) = m\} = \sum_n \binom{n}{m} \{P(i)\}^m \{1 - P(i)\}^{n-m} \Pr\{N_0 = n\} \qquad (m = 0, 1, 2, \cdots). \tag{6.2}$$

The stochastic behavior of the software fault-detection or the failure-occurrence phenomenon in the testing-phase can be characterized by giving a suitable probability mass function of the initial fault content N_0. Okamura et al. [34] have discussed a generalized discrete NHPP model for software reliability assessment by assuming that the initial fault content, N_0, obeys a Poisson distribution, and proposed a parameter estimation method based on the EM algorithm.

6.2.2 Generalized discrete binomial process modeling

In this chapter we propose a generalized discrete binomial process model for software reliability assessment by considering the case that the probability distribution of the initial fault content, N_0, obeys a binomial distribution with parameters (K, λ), which is given as

$$\Pr\{N_0 = n\} = \binom{K}{n} \lambda^n (1 - \lambda)^{K-n} \qquad (0 < \lambda < 1\ ;\ n = 0, 1, \cdots, K). \tag{6.3}$$

Eq. (6.3) has the following physical assumptions:

(a) The software system consists of K lines of code (LOC) at the beginning of the testing-phase,

(b) Each code has a fault with a constant probability λ,

(c) Each software failure caused by a fault remaining in the software system occurs independently and randomly.

These assumptions are useful to apply a binomial distribution as a probability mass function of an initial fault content in the software system to software reliability growth modeling, and to incorporate the effect of the program size into software reliability growth modeling.

Substituting Eq. (6.3) into Eq. (6.2), we can derive the probability mass function of the number of faults detected up to i-th testing-period as

$$\Pr\{N_B(i) = m\} = \binom{K}{m} \{\lambda P(i)\}^m \{1 - \lambda P(i)\}^{K-m} \qquad (m = 0, 1, 2, \cdots K). \tag{6.4}$$

From Eq. (6.4), we can see that the number of faults detected up to i-th testing-period obeys a binomial process if the probability mass function of the initial fault content is the binomial distribution as Eq. (6.3).

6.2.3 Discrete failure-occurrence times distribution

We need to specify a discrete failure-occurrence times distribution to develop an SRGM. In this chapter we apply a discrete Weibull distribution [30] to the discrete failure-occurrence times distribution. Its probability distribution function is given as

$$P(i) = 1 - (1 - p)^{i^\beta} \qquad (i = 1, 2, \cdots ; \ \beta > 0, \ 0 < p < 1). \tag{6.5}$$

In Eq. (6.5), p represents the probability that a software failure caused by a fault is observed per one testing-period, and β the shape parameter.

We focus on the cases that $\beta = 1$ and $\beta = 2$, respectively, as the special cases for the discrete Weibull distribution in Eq. (6.5). When $\beta = 1$ in Eq. (6.5), the distribution becomes a geometric distribution:

$$P(i) = 1 - (1 - p)^i \qquad (i = 1, 2, \cdots ; \ 0 < p < 1), \tag{6.6}$$

which has the constant failure rate. The geometric distribution means that a software failure occurs at any testing-period decreases geometrically, which represents the case that the internal program structure is simple and the testing-skill of test-case designers is high. When $\beta = 2$, the distribution can be regard as a discrete Rayleigh distribution:

$$P(i) = 1 - (1-p)^{i^2} \qquad (i = 1, 2, \cdots ; \ 0 < p < 1), \tag{6.7}$$

which has the increasing failure rate. Applying the discrete Rayleigh distribution to a software failure-occurrence times distribution means that the internal program structure is complex and the initial testing-skill of test-case designers is low, however, the testing-skill of them improves more and more as the testing-period goes on.

6.3 Generalized Reliability Assessment Measures

Software reliability assessment measures are well-known as useful metrics which enable us to assess software reliability quantitatively. In this section we derive several generalized software reliability assessment measures based on the unified framework of discrete software reliability growth modeling.

6.3.1 Expectation and variance of the number of detected faults

Information on the current number of detected faults is one of the important metrics to estimate the degree of testing-progress. Therefore, the expectation and variance of the number of detected faults are useful measures because the number of faults detected up to i-th testing-period, $N(i)$ in Eq. (6.2), is treated as a random variable.

The expectation of the number of detected faults, $\mathrm{E}[N(i)]$, is derived as

$$\begin{aligned}\mathrm{E}[N(i)] &= \sum_{z=0}^{n} z \sum_{n} \binom{n}{z} \{P(i)\}^z \{1 - P(i)\}^{n-z} \Pr\{N_0 = n\} \\ &= \mathrm{E}[N_0] P(i).\end{aligned} \tag{6.8}$$

And the variance, $\mathrm{Var}[N(i)]$, is also derived as

$$\begin{aligned}\mathrm{Var}[N(i)] &= \mathrm{E}[N(i)^2] - (\mathrm{E}[N(i)])^2 \\ &= \mathrm{Var}[N_0]\{P(i)\}^2 + \mathrm{E}[N_0]P(i)\{1 - P(i)\}.\end{aligned} \tag{6.9}$$

Therefore, if N_0 obeys the binomial distribution in Eq. (6.3), they are given as

$$\mathrm{E}[N_B(i)] = K\lambda P(i), \tag{6.10}$$

$$\mathrm{Var}[N_B(i)] = K\lambda P(i)\{1 - \lambda P(i)\}, \tag{6.11}$$

respectively. We can see that $K\lambda$ in Eq. (6.10) represents the expected initial fault content when N_0 obeys the binomial distribution.

6.3.2 Software reliability function

A software reliability function is one of the well-known software reliability assessment measures. Given that the testing or the operation has been going up to i-th testing-period, the discrete software reliability function is defined as the probability that a software failure does not occur in the time-interval $(i, i+h](i, h = 0, 1, \cdots)$ [60]. Accordingly, we can formulate the discrete software reliability function $R(i, h)$ as

$$\begin{aligned} R(i,h) &= \sum_k \Pr\{N(i+h) = k \mid N(i) = k\}\Pr\{N(i) = k\} \\ &= \sum_k \left[\{P(i)\}^k\{1 - P(i+h)\}^{-k}\sum_n \binom{n}{k}\{1 - P(i+h)\}^n \cdot \Pr\{N_0 = n\}\right], \end{aligned} \tag{6.12}$$

by using Eq. (6.2). Therefore, if N_0 obeys the binomial distribution in Eq. (6.3), the discrete software reliability function can be derived as

$$R_B(i,h) = [1 - \lambda\{P(i+h) - P(i)\}]^K, \tag{6.13}$$

by using Eq. (6.12).

6.3.3 Instantaneous and cumulative MTBF's

As substitutions for an ordinary MTBF, we derive instantaneous and cumulative MTBF's. The ordinary MTBF can not be derived because the generalized SRGM has the following properties:

$$F(i,0) = 1 - R(i,0) = 0, \tag{6.14}$$

$$\begin{aligned} F(i,\infty) &= 1 - R(i,\infty) \\ &= 1 - \sum_n \{P(i)\}^n \cdot \Pr\{N_0 = n\}, \end{aligned} \tag{6.15}$$

where $F(i, h)$ represents the probability that a software failure occurs in the time-interval $(i, i+h]$. These equations above implies that the probability distribution function, $F(i, h)$, does not satisfy the properties of the ordinary probability distribution function. Accordingly, we need to utilize discrete instantaneous and cumulative MTBF's as substitutions for the ordinary MTBF.

Using Eq. (6.8), we can formulate the discrete instantaneous MTBF as

$$\mathrm{MTBF}_I(i) = \frac{1}{\mathrm{E}[N(i+1)] - \mathrm{E}[N(i)]}. \tag{6.16}$$

And the discrete cumulative MTBF can also given as

$$\mathrm{MTBF}_C(i) = \frac{i}{\mathrm{E}[N(i)]}. \tag{6.17}$$

By substituting Eq. (6.10) into Eqs. (6.16) and (6.17), we can obtain specified instantaneous and cumulative MTBF's, respectively.

6.4 Parameter Estimation

In this section we discuss parameter estimation for the generalized discrete binomial process model in Eq. (6.4) based on the method of maximum-likelihood. Suppose that we have observed J data pairs $(t_j, y_j)(j = 0, 1, 2, \cdots, J)$ with respect to the cumulative number of faults, y_j, detected during a constant time-interval $(0,\ t_j](0 < t_1 < t_2 < \cdots < t_J)$. The likelihood function l for the generalized discrete binomial process model, $N_B(i)$, can be derived as

$$\begin{aligned} l &\equiv \Pr\{N_B(t_1) = y_1, N_B(t_2) = y_2, \cdots, N_B(t_J) = y_J\} \\ &= \prod_{j=2}^{J} \Pr\{N_B(t_j) = y_j \mid N_B(t_{j-1}) = y_{j-1}\} \Pr\{N_B(t_1) = y_1\}, \end{aligned} \tag{6.18}$$

by using the Bayes' formula and the Markov property [36, 50]. The conditional probability in Eq. (6.18), $\Pr\{N_B(t_j) = y_j \mid N_B(t_{j-1}) = y_{j-1}\}$, can be shown as

$$\begin{aligned} &\Pr\{N_B(t_j) = y_j \mid N_B(t_{j-1}) = y_{j-1}\} \\ &\qquad = \binom{K - y_{j-1}}{y_j - y_{j-1}} \{z(t_{j-1}, t_j)\}^{y_j - y_{j-1}} \{1 - z(t_{j-1}, t_j)\}^{K - y_j}, \end{aligned} \tag{6.19}$$

by considering that we can regard t_{j-1} as the initial time and that the distribution range of $N_B(i)$ is $0 \leq N_B(i) \leq K - y_{j-1}$. In above equation, setting

$$z(t_{j-1}, t_j) = \frac{\lambda\{P(t_j) - P(t_{j-1})\}}{1 - \lambda P(t_{j-1})}, \tag{6.20}$$

we can rewrite Eq. (6.18) as

$$l = \prod_{j=1}^{J} \binom{K - y_{j-1}}{y_j - y_{j-1}} \{z(t_{j-1}, t_j)\}^{y_j - y_{j-1}} \{1 - z(t_{j-1}, t_j)\}^{K - y_j}, \tag{6.21}$$

by using Eq. (6.19), where $t_0 = 0$, $y_0 = 0$, and $P(t_0) = 0$. Accordingly, the logarithmic likelihood function can be derived as

$$\begin{aligned}\log l &\equiv L \\ &= \log K! - \log\{(K - y_J)!\} - \sum_{j=1}^{J} \log\{(y_j - y_{j-1})!\} + y_J \log\lambda \\ &\quad + \sum_{j=1}^{J}(y_j - y_{j-1})\log\{P(t_j) - P(t_{j-1})\} + (K - y_J)\log\{1 - \lambda P(t_J)\}, \qquad (6.22)\end{aligned}$$

by taking the natural logarithm of Eq. (6.21).

When we apply the discrete Weibull distribution in Eq. (6.5) to the software failure-occurrence times distribution, i.e., $P(i) = 1 - (1-p)^{i^\beta}$, the logarithmic likelihood function can be given as

$$\begin{aligned}L = \log K! &- \log\{(K - y_J)!\} + y_J \log\lambda - \sum_{j=1}^{J}\{(y_j - y_{j-1})!\} \\ &+ \sum_{j=1}^{J}(y_j - y_{j-1})\log\{(1-p)^{t_{j-1}^\beta} - (1-p)^{t_j^\beta}\} \\ &\qquad + (K - y_J)\log\left[1 - \lambda\{1 - (1-p)^{t_J^\beta}\}\right], \qquad (6.23)\end{aligned}$$

by using Eq. (6.22). In this case that the value of the parameter β in Eq. (6.5) is supposed, we have to estimate the parameters λ and p if we can know the program size K. The simultaneous logarithmic likelihood equations with respect to the parameters λ and p can be derived as

$$\frac{\partial L}{\partial\lambda} = \frac{y_J}{\lambda} + (K - y_J)\cdot\{(1-p)^{t_J^\beta} - 1\}\frac{1}{1 - \lambda\{1 - (1-p)^{t_J^\beta}\}} = 0, \qquad (6.24)$$

$$\begin{aligned}\frac{\partial L}{\partial p} &= \sum_{j=1}^{J}(y_j - y_{j-1})\frac{1}{\{(1-p)^{t_{j-1}^\beta} - (1-p)^{t_j^\beta}\}}\{t_j^\beta(1-p)^{t_j^\beta - 1} - t_{j-1}^\beta(1-p)^{t_{j-1}^\beta - 1}\} \\ &\quad - (K - y_J)\{t_J^\beta\lambda(1-p)^{t_J^\beta - 1}\}\frac{1}{1 - \lambda\{1 - (1-p)^{t_J}\}} = 0, \qquad (6.25)\end{aligned}$$

respectively. Solving Eq. (6.24) with respect to λ, we can obtain

$$\lambda = \frac{y_J}{K\{1 - (1-p)^{t_J^\beta}\}}. \qquad (6.26)$$

Substituting Eq.(6.26) into Eq.(6.25), we can obtain the following equation:

$$\begin{aligned}\frac{t_J^\beta y_J(1-p)^{t_J^\beta - 1}}{1 - (1-p)^{t_J^\beta}} &= \sum_{j=1}^{J}(y_j - y_{j-1})\frac{1}{\{(1-p)^{t_{j-1}^\beta} - (1-p)^{t_j^\beta}\}} \\ &\qquad \cdot\{t_j^\beta(1-p)^{t_j^\beta - 1} - t_{j-1}^\beta(1-p)^{t_{i-1}^\beta - 1}\}. \qquad (6.27)\end{aligned}$$

Accordingly, we can obtain the maximum-likelihood estimates $\widehat{\lambda}$ and $\widehat{p}$ of the unknown parameters λ and p, respectively, by solving the simultaneous likelihood functions in Eqs.(6.26) and (6.27) numerically.

6.5 Optimal Software Release Problems

Software developing managers have a great interest in how to develop the reliable software product economically and when to release the software to the customers [59]. In this section we discuss discrete cost-optimal software release policies based on our generalized discrete binomial process model. And then, we also discuss discrete optimal software release policies with simultaneous cost and reliability requirements in consideration of software quality control point of view. In this chapter we discuss the case that the geometric distribution in Eq. (6.6) is applied to the software failure-occurrence times distribution.

6.5.1 Cost-optimal software release policies

We discuss cost-optimal software release policies based on our generalized discrete binomial process model. First of all, the following notations are defined:

c_1 : debugging cost per one fault in the testing-phase,

c_2 : debugging cost per one fualt in the operational phase, where $c_1 < c_2$,

c_3 : testing cost per constant period.

Let Z denote the software release period. Then, the expected total software cost $C(Z)$ which indicates the expected total cost during the testing and operational phases is formulated as

$$C(Z) = c_1\mathrm{E}[N_B(Z)] + c_2(K\lambda - \mathrm{E}[N_B(Z)]) + c_3 Z. \tag{6.28}$$

The cost-optimal software release period is derived by minimizing the expected total software cost $C(Z)$ in Eq. (6.28). From Eq. (6.28), we can derive the following equations:

$$\frac{C(Z+1)-C(Z)}{\delta} = \frac{c_2-c_1}{\delta}\left[\frac{c_3}{c_2-c_1} - W(Z)\right], \tag{6.29}$$

where $W(Z)$ represents the expected number of detected faults during a Z-th testing-period. And, we need to define the following notation to discuss the discrete software release policies:

$$<n>=\begin{cases} [n] & (\text{if } C([n]) \leq C([n]+1) \\ [n]+1 & (\text{otherwise}), \end{cases} \tag{6.30}$$

where $[\,n\,]$ represents the Gaussian symbol for any real number n.

In the case that the software failure-occurrence times distribution obeys the geometric distribution, we can confirm that $W(Z)$ has the following properties:

$$\left.\begin{array}{l} W(Z+1) < W(Z) \\ W(0) = K\lambda p \\ W(\infty) = 0 \end{array}\right\}, \tag{6.31}$$

for any nonnegative interger $Z(\geq 0)$ since $0 < p < 1$. That is, we can see that $W(Z)$ is a monotonically decreasing function in terms of the testing-period $Z(\geq 0)$. Therefore, we can obtain the cost-optimal software release policies as follows:

[Cost-Optimal Release Policy]

Suppose that $c_2 > c_1 > 0$ and $c_3 > 0$.

(1) If $W(0) \leq \frac{c_3}{c_2 - c_1}$, then the cost-optimal software release period is $Z^* = 0$.

(2) If $\frac{c_3}{c_2 - c_1} < W(0)$, then we have the following an only solution $Z = Z_0$ minimizing Eq.(6.28):

$$Z_0 = \frac{\log\left[\frac{c_3}{(c_2 - c_1)K\lambda p}\right]}{\log(1-p)}. \tag{6.32}$$

Thus, the optimal software release period $Z^* =< Z_0 >$.

6.5.2 Cost-reliability-optimal software release policies

Further, we discuss the optimal software release problems which take both total software cost and reliability criteria into consideration simultaneously. In the actual software development, the software developing manager has to spend and control the testing resources under both minimizing the total software cost and satisfying the software reliability requirement rather than only minimizing the cost.

Now, let R_0 $(0 < R_0 \leq 1)$ be the software reliability objective. Using the discrete software reliability function in Eq. (6.13), we can discuss the optimal software release policies which minimize the total expected software cost in Eq. (6.28) with satisfying the software reliability objective R_0. That is, the cost-reliability-optimal software release problem can be formulated as follows:

$$\left.\begin{array}{l} \text{minimize } C(Z) \\ \text{subject to } R_B(Z,h) \geq R_0,\ Z \geq 0 \end{array}\right\}. \tag{6.33}$$

Supposing h is a constant value, we can see that the discrete software reliability function $R_B(Z,h)$ is a monotonically increasing function in terms of the testing-period Z when the software failure-occurrence times obeys the geometric distribution. Accordingly, if $R_B(0,h) < R_0$, then we have only finite solution Z_1 satisfying $R_B(Z-1,h) < R_0$ and $R_B(Z,h) \geq R_0$. Furthermore, if $R_B(0,h) \geq R_0$, then $R_B(Z,h) \geq R_0$ for any nonnegative integer Z. Therefore, in this case, we have to discuss optimal software release policies based on only the cost criterion. From the discussion above, the cost-reliability-optimal software release policies can be obtained as follows:

[Cost-Reliability-Optimal Release Policy]

Suppose that $c_2 > c_1 > 0$, $c_3 > 0$, $0 < R_0 \leq 1$, and $h \geq 0$.

(1) If $W(0) \leq \frac{c_3}{c_2-c_1}$ and $R_B(0,h) \geq R_0$, then the cost-reliability-optimal software release period $Z^* = 0$.

(2) If $W(0) \leq \frac{c_3}{c_2-c_1}$ and $R_B(0,h) < R_0$, then the cost-reliability-optimal software release period $Z^* = Z_1$.

(3) If $W(0) > \frac{c_3}{c_2-c_1}$ and $R_B(0,h) \geq R_0$, then the cost-reliability-optimal software release period $Z^* =< Z_0 >$.

(4) If $W(0) > \frac{c_3}{c_2-c_1}$ and $R_B(0,h) < R_0$, then the optimal software release period $Z^* = \max\{< Z_0 >, Z_1\}$.

6.6 Numerical Examples

We show numerical examples for our generalized discrete binomial process model in Eq.(6.4) by using actual fault count data cited by Ohba [32] and RADC [6]. We call these data DS1 and DS2 in this chapter, respectively. DS1 consists of 19 data pairs $(t_j,\ y_j)(j = 0,1,2,\cdots,19\,;\ t_{19} = 19, y_{19} = 328)$ and the program size K of this software system is 1.317×10^6 (LOC). And DS2 consists of 35 data pairs $(t_j,\ y_j)(j = 0,1,2,\cdots,35\,;\ t_{35} = 35, y_{35} = 1301)$ and its program size K is 1.240×10^6 (LOC). DS1 and DS2 show exponential and S-shaped reliability growth curves, respectively. Therefore, we use DS1 in the case that the software failure-occurrence times distribution obeys the geometric distribution, and DS2 in the case that the distribution obeys the discrete Rayleigh distribution, respectively.

Figs. 6.1 and 6.2 depicts the estimated expected number of detected faults, $\widehat{\mathrm{E}}[N_B(i)]$'s, and its 95% confidence limits in case of the geometric and the Rayleigh software failure-occurrence times distributions, respectively. The 100γ% confidence limits for $\widehat{\mathrm{E}}[N_B(i)]$ are derived as

$$\widehat{\mathrm{E}}[N_B(i)] \pm K_\gamma \sqrt{\widehat{\mathrm{Var}}[N_B(i)]}, \tag{6.34}$$

where K_γ indicates the $100(1+\gamma)/2$ percent point of the standard normal distribution [58]. As to Fig. 6.1 the estimates of the unknown parameters λ and p have been obtained that

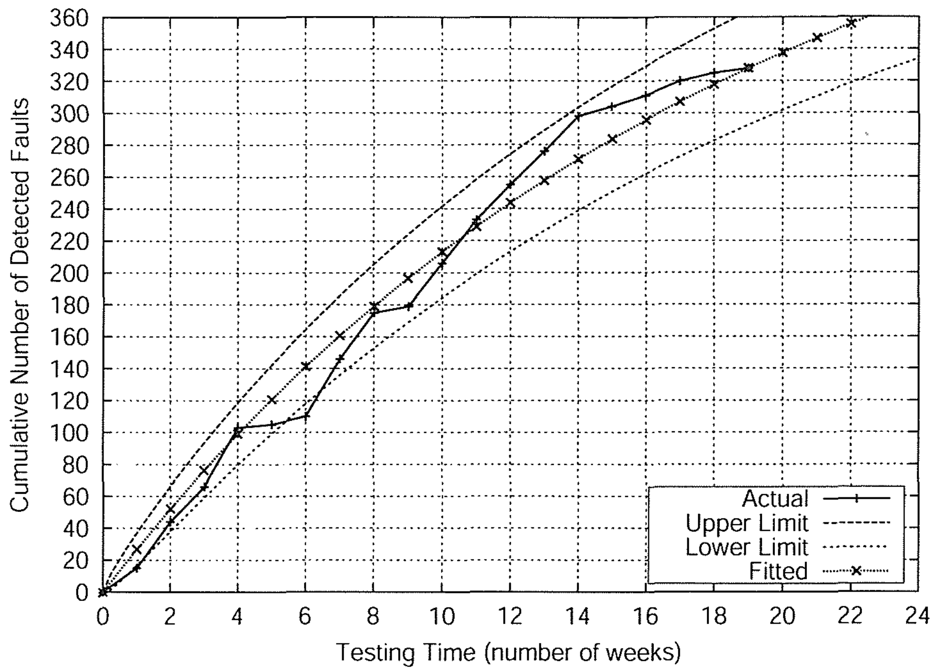


Fig. 6.1 : The estimated expected number of detected faults in the case of the geometric software failure-occurrence times distribution and its 95% confidence limits for DS1.

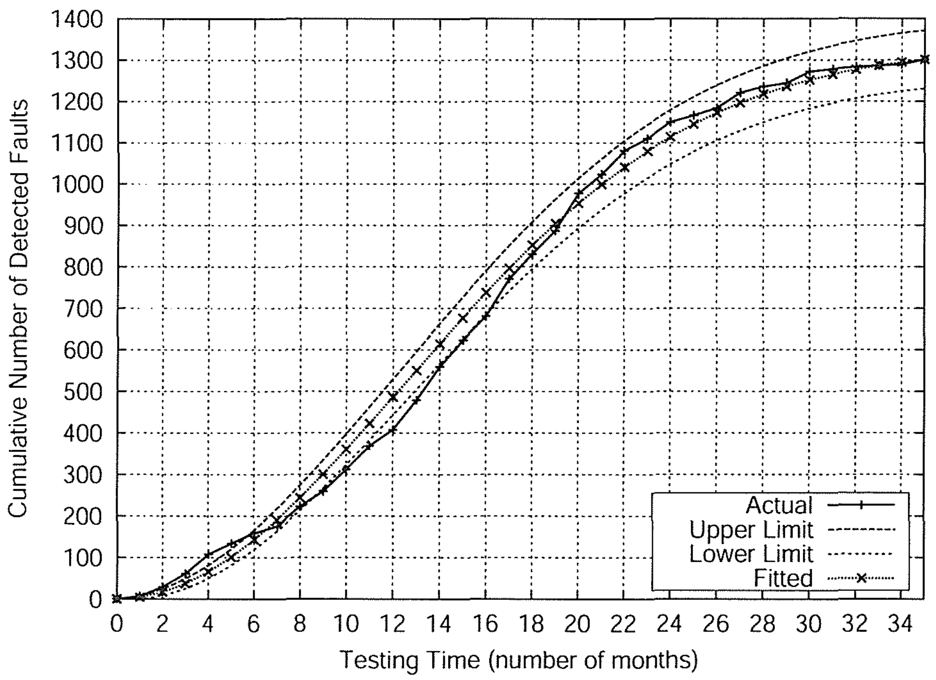


Fig. 6.2 : The estimated expected number of detected faults in the case of the Rayleigh software failure-occurrence times distribution and its 95% confidence limits for DS2.

$\widehat{\lambda} = 0.340 \times 10^{-3}$ and $\widehat{p} = 0.052$, respectively, by using the method of maximum-likelihood discussed in Section 6.4. By using the estimates, the expected initial fault content can be estimated as $\widehat{K} \cdot \widehat{\lambda} \approx 513$. In Fig. 6.2 the estimates of the unknown parameters λ and p have been obtained that $\widehat{\lambda} = 0.316 \times 10^{-2}$ and $\widehat{p} = 0.107 \times 10^{-1}$, respectively. Accordingly, we can estimate the expected initial fault content to be about 1328.

Figs. 6.3 and 6.4 show the estimated software reliability functions, $\widehat{R_B}(i,1)$'s, by using the estimated parameters, respectively. By using the estimated software reliability functions, $\widehat{R_B}(i,1)$, we can estimate the software reliabilities at the 60-th testing-period to be about 0·342 in Fig. 6.3 and at the 45-th testing-period to be about 0 · 579 in Fig. 6.4, respectively.

Figs. 6.5 and 6.6 depict the estimated instantaneous MTBF's, $\widehat{MTBF_B}(i)$'s, respectively. In Fig. 6.5, we can estimate the instantaneous MTBF at the 60-th testing-period to be about 0·933 (weeks) or to be about 157 (hours). And we also estimate one at the 45-th testing-period to be about 0 · 1.833 (months) in Fig. 6.6.

Next we show numerical examples for discussed optimal software release policies, such as the cost-optimal and the cost-reliability-optimal release policies, respectively. Fig. 6.7 depicts derived cost-optimal software release policy for $c_1 = 1$, $c_2 = 32$, and $c_3 = 10$. In this case, **Cost-Optimal Release Policy** (2) is applied, then we can estimate that the cost-optimal software

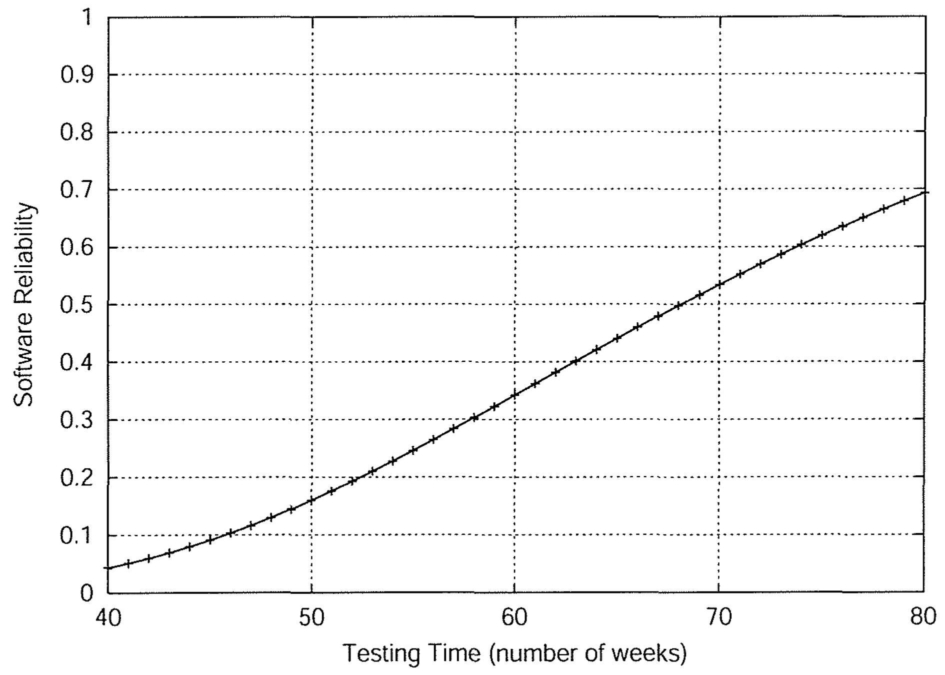


Fig. 6.3 : The estimated sotware reliability function in the case of the geometric software failure-occurrence times distribution for DS1.

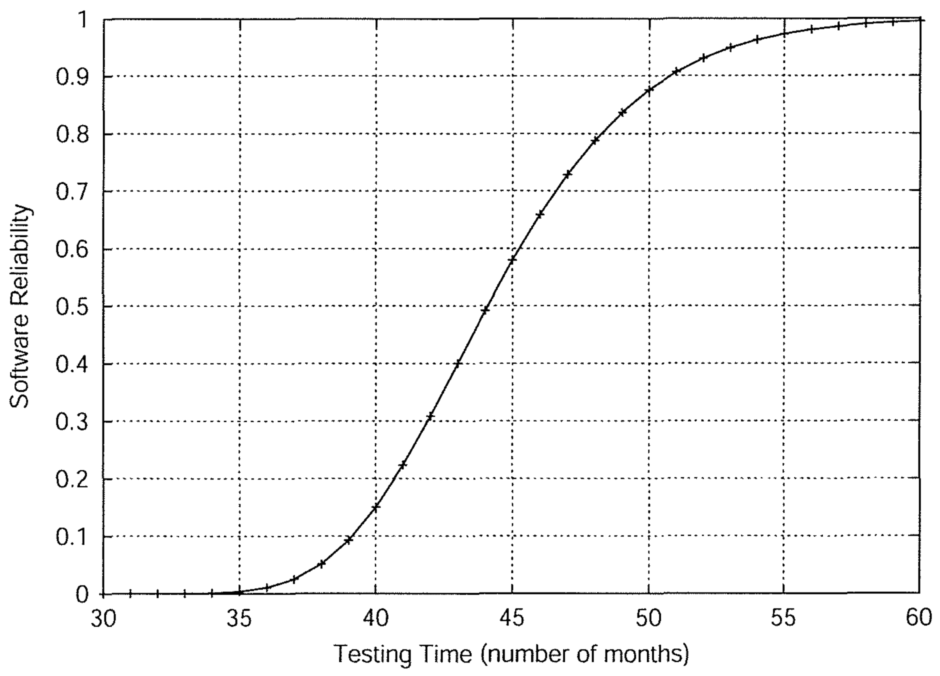


Fig. 6.4 : The estimated sotware reliability function in the case of the Rayleigh software failure-occurrence times distribution for DS2.

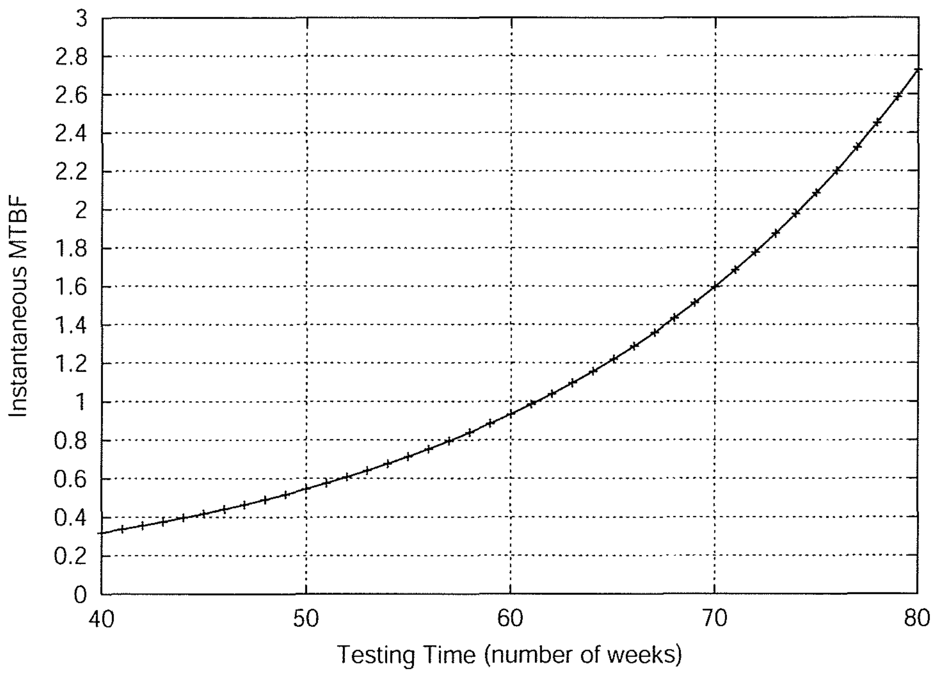


Fig. 6.5 : The estimated instantaneous MTBF in the case of the geometric software failure-occurrence times distribution for DS1.

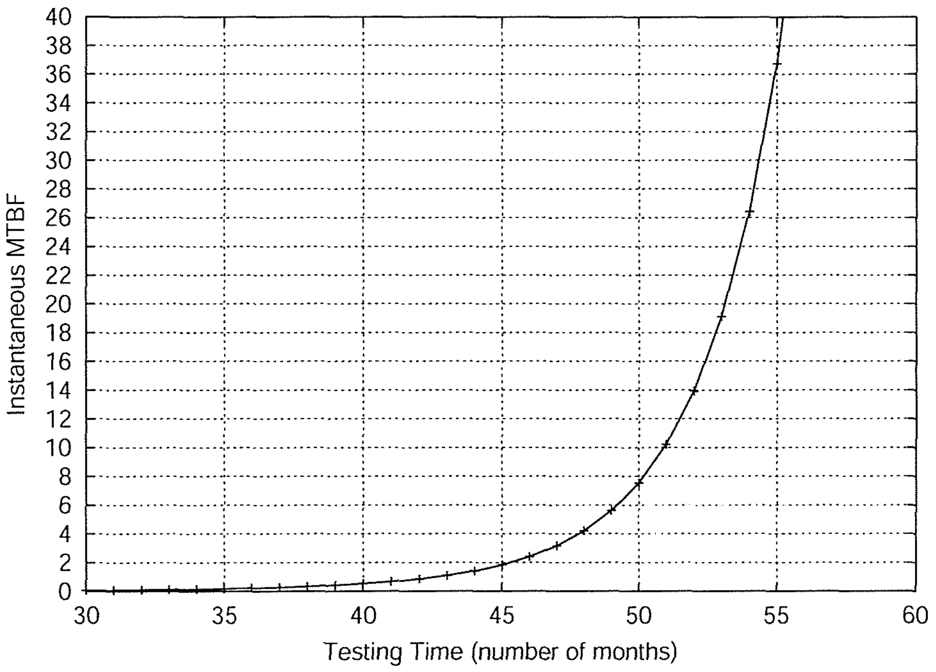


Fig. 6.6 : The estimated instantaneous MTBF in the case of the Rayleigh software failure-occurrence times distribution for DS2.

release period $Z^* = 83$ (weeks). And Fig. 6.8 shows the numerical examples for derived cost-reliability-optimal software release policy. For the specific operational period $h = 1$ and the reliability objective $R_0 = 0 \cdot 8$, the cost-reliability-optimal software release problem can be discussed in the followings. Suppose that the cost-optimal software release policy have been discussed about the case of $c_1 = 1$, $c_2 = 32$, and $c_3 = 10$. We can estimate $Z_1 = 90$ because $R(89, 1) = 0.798 < R_0$ and $R(90, 1) = 0.807 > R_0$. Since $W(Z) > c_3/(c_2 - c_1)$ and $R(0, 1) = 2 \cdot 280 \times 10^{-12} < R_0$, Z^* is estimated as $Z^* = \max\{< Z_0 >, Z_1\} = \max\{83, 90\} = 90$ by using the **Cost-Reliability-Optimal Release Policy** (4). In Fig. 6.8, we can understand an importance that the software development managers should estimate the optimal software release period by considering not only minimizing the total expected software cost but also satisfying the reliability objective.

6.7 Concluding Remarks

We have discussed a generalization for discrete software reliability growth modeling. We have then developed generalized discrete binomial process model based on the unified framework by assuming that a probability distribution of the initial fault content obeys the binomial

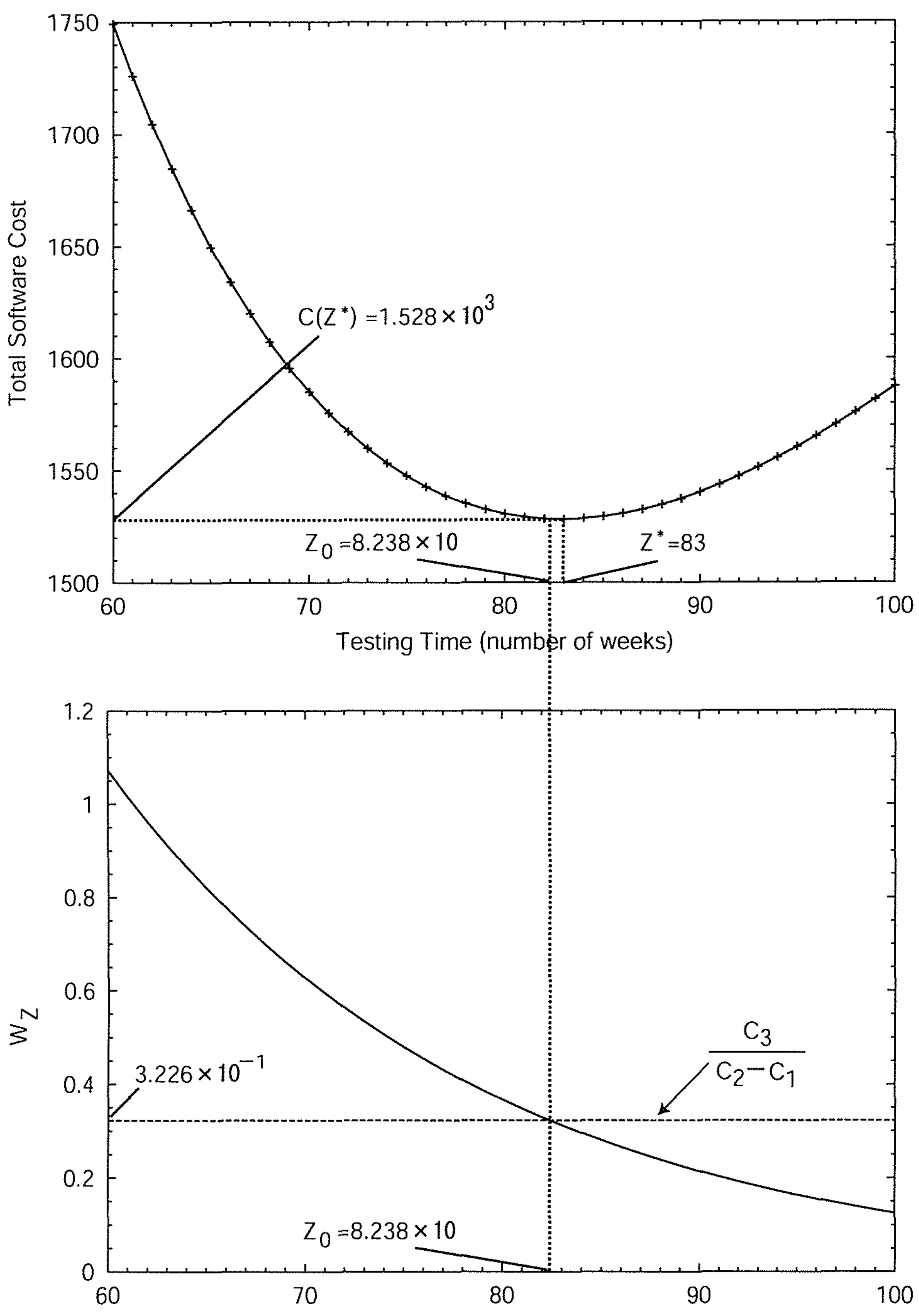


Fig. 6.7 : The cost-optimal software release policy in the case of $c_1 = 1$, $c_2 = 32$, and $c_3 = 10$ for DS1.

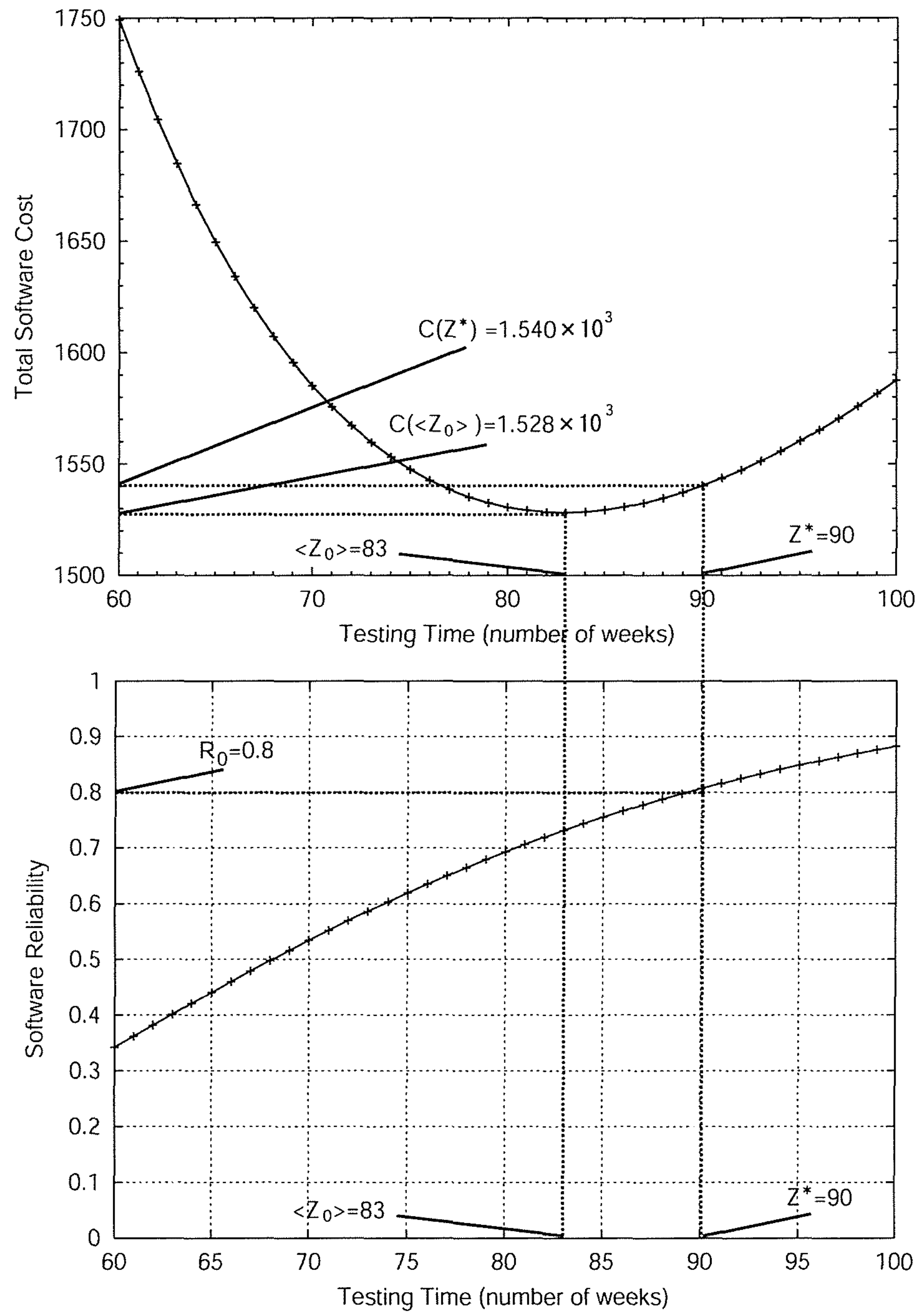


Fig. 6.8 : The cost-reliability-optimal software release policy in the case of $c_1 = 1$, $c_2 = 32$, and $c_3 = 10$ for DS1.

distribution, which enables us to assess software reliability in consideration of the effect of the program size. After that, we have derived generalized discrete software reliability assessment measures based on the concept of the unified framework. And, parameter estimation method based on the method of maximum-likelihood for our generalized discrete binomial process model have been discussed. Additionally, as one of the applications of our generalized discrete binomial process model, we have discussed optimal software release problems with simultaneous cost and a reliability objective. Finally we have shown numerical examples for software reliability assessment based on our generalized discrete binomial process models and its application to derived optimal release policies by using actual fault count data.

Chapter 7

Infinite Server Queueing Modeling for Software Reliability Assessment

7.1 Introduction

Quantitative assessment of software reliability in a testing-phase is important to provide a software keeping high degree of reliability for users because a testing-phase is located in the final stage of the software development process. Up to now, as mathematical models to assess software reliability, several SRGM's have been utilized for assessing the degree of the achievement of software quality, deciding the time to release for operational use, and evaluating the maintenance cost for faults undetected during the testing-phase. Most of SRGM's have been modeled by any stochastic processes to describe the software fault-detection phenomenon or the software failure-occurrence phenomenon. Especially, it is known that an NHPP model can describe software reliability growth process easily by supposing the mean value function of the NHPP intuitively. On the other hand, for most of NHPP models, it has been pointed out that it is difficult to understand physical interpretation for the fault-detection phenomenon by many researchers. As one of the methods for solving this problem, several generalization methods for SRGM's have been proposed. In recent years, Dohi et al. [8] has proposed a general approach to existing SRGM's by regarding the software failure-occurrence phenomenon as an infinite server queue.

In this chapter we discuss generalized software reliability growth modeling by applying an infinite server queueing thoery to basic assumptions of a delayed S-shaped SRGM [58,66], which is a different approach from the infinite server queueing models proposed by Dohi et al. [8]. The delayed S-shaped SRGM is one of the SRGM's which can analyze the physical interpretation for the fault-detection phenomenon. This SRGM has been developed by supposing that the fault-detection phenomenon consists of successive software failure-detection and fault-isolation processes. In an actual testing-phase, we can consider that the time for analyzing or isolating the causes of software failures do not always take a constant values.

We discuss a concept of conditional distribution of arrival times, which is utilized for developing our infinite server queueing model. After that, based on the concepts of the delayed S-shaped SRGM and the conditional distribution of arrival times, we develop an infinite server queueing model considering the time distribution of the fault-isolation process for software reliability assessment. Finally, we mention that our infinite server queueing model is a general approach for several SRGM's described by NHPP's, and show numerical examples for our model by using actual fault count data.

7.2 Delayed S-shaped SRGM

In this section we discuss an SRGM based on an NHPP and a concept of the delayed S-shaped software reliability growth modeling [58, 66] which is the basic concept for developing our infinite server queueing model.

First, let $\{Z(t), t \geq 0\}$ be the counting process representing the cumulative number of faults detected up to time t $(t \geq 0)$. Supposing that $Z(t)$ obeys an NHPP, we can formulate the fault-detection phenomenon as

$$\left.\begin{aligned} \Pr\{Z(t)=n\} &= \frac{\{H(t)\}^n}{n!}\exp\{-H(t)\} \qquad (n=0,1,2,\cdots) \\ H(t) &= \int_0^t h(t)dt \end{aligned}\right\}, \tag{7.1}$$

where $H(t)$ is the mean value function which indicates the expectation of $Z(t)$, i.e., the expected cumulative number of faults detected up to time t, and $h(t)$ called an intensity function which indicates the instantaneous fault-detection rate at time t. Eq. (7.1) implies that the software reliability growth process in the testing-phase is characterized by the mean value function $H(t)$ or the intensity function $h(t)$.

Generally, the cause analysis to detect software faults occurring software failures is conducted in the testing-phase. Accordingly, the delayed S-shaped SRGM has been developed by supposing that the fault-detection process is consisted of successive software failure-detection and fault-isolation processes. That is, this SRGM regards analyzing the software failure-occurrence phenomenon and isolating the faults causing software failures as the fault-detection.

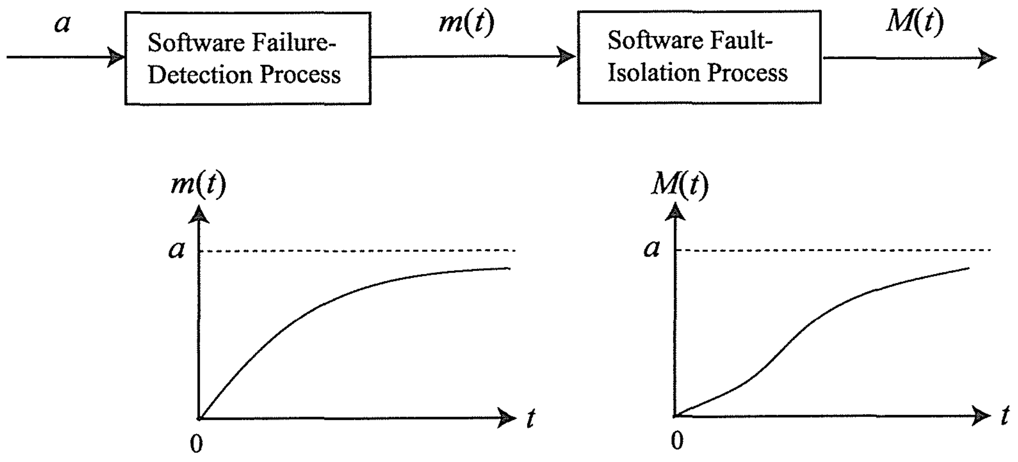

a
Software Failure-Detection Process
m(t)
Software Fault-Isolation Process
M(t)
m(t)
a
0
t
M(t)
a
0
t

Fig. 7.1 : A basic concept of delayed S-shaped software reliability growth modeling.

The delayed S-shaped SRGM is derived by the following procedure. First of all, let $m(t)$ be the expected cumulative number of software failures detected up to time t in the software failure-detection process. Then, we can obtain the following differential equation by the assumptions of the delayed S-shaped SRGM:

$$\frac{dm(t)}{dt} = b_1[a - m(t)], \tag{7.2}$$

where a indicates the expected initial fault content in the software system, and $b_1(> 0)$ the failure-occurrence rate. Next, letting $M(t)$ be the expected cumulative number of faults isolated (or detected) up to time t in the fault-isolation process, we can also obtain the following differential equation by assumptions of the delayed S-shaped SRGM:

$$\frac{dM(t)}{dt} = b_2[m(t) - M(t)], \tag{7.3}$$

where $b_2(> 0)$ represents the fault-detection rate. Suppose that $b = b_1 = b_2$ approximately. Then, $M(t)$ can be derived from Eqs. (7.2) and (7.3) as

$$M(t) = a[1 - (1 + bt)\exp\{-bt\}]. \tag{7.4}$$

The mean value funtion $M(t)$ in Eq. (7.4) is called a *delayed S-shaped software reliability growth model* [58,66]. Fig. 7.1 shows the concept of the delayed S-shaped SRGM. Additionally, let $M_G(t)$ be the expected cumulative number of faults detected up to time t in case of $b_1 \neq b_2$. Then the mean value function $M_G(t)$ is called a *generalized delayed S-shaped software reliability growth model* [62], and is derived as

$$M_G(t) \;=\; a\left[1 - \frac{1}{1-v}(\exp\{-b_1 v t\} - v\exp\{-b_1 t\})\right], \tag{7.5}$$

where $v = b_2/b_1$ which represents the relative measure between the frequency of the fault-occurrence and the isolation progress rate.

7.3 Infinite Server Queueing Modeling

Two events of software failure-occurrence phenomenon and fault-detection process have different meaning each other. Thus, the faults are not always detected even if the software failures are occurred. And the time spent by analyzing the causes of each software failure is randomly behaved by difference of the difficulty of isolating and detecting each fault. First of all, in this section we introduce a concept of conditional distribution of arrival times which is need for developing an infinite server queueing model. After that, utilizing the concept, we develop an infinite server queueing model [36,41,42,50] to treat above situation comprehensively. And we also propose SRGM's considering the time distribution of fault-isolation process.

7.3.1 Conditional arrival times distribution

Before developing an infinite server queueing model, we need to discuss a concept of conditional distribution of arrival times. In this section we discuss the conditional arrival times distribution in case that the events occur in accordance with an NHPP formulated as Eq. (7.1).

Let $S_1, S_2, \cdots, S_n$ be the n arrival times of a counting process $\{Z(t),\ t \geq 0\}$ which obeys an NHPP with its mean value function $H(t)$ and intensity function $h(t)$ in Eq. (7.1). Now we consider the conditional distribution of the first arrival time, S_1, given that there was an event in the time-interval $[0,\ t]$. For $s < t$, the conditional distribution is derived as

$$\begin{aligned}\Pr\{S_1 \leq s_1 \mid Z(t) = 1\} &= \frac{H(s_1)}{H(t)} \\ &= \int_0^{s_1} \frac{h(x)}{H(t)} dx. \end{aligned} \tag{7.6}$$

Similarly, we can derive the joint conditional distribution of S_1 and S_2 as follows:

$$\begin{aligned}&\Pr\{S_1 \leq s_1, S_2 \leq s_2 \mid Z(t) = 2\} \\ &\quad = 2! \frac{H(s_1)[H(s_2) - H(s_1)]}{[H(t)]^2} \\ &\quad = 2! \int_{s_1}^{s_2} \int_0^{s_1} \frac{\prod_{i=1}^2 h(x_i)}{[H(t)]^2} dx_1 dx_2, \end{aligned} \tag{7.7}$$

where $s_1 < s_2 \le t$. Then, if we condition that $Z(t) = n$, the joint conditional distribution of n arrival times is given by

$$\begin{aligned}\Pr\{S_1 \le s_1,\ S_2 \le s_2, \cdots, S_n \le s_n \mid Z(t) = n\} \\ = n! \int_{s_0}^{s_1} \int_{s_1}^{s_2} \cdots \int_{s_{n-1}}^{s_n} \frac{\prod_{i=1}^{n} h(x_i)}{[H(t)]^n} dx_1 dx_2 \cdots dx_n.\end{aligned} \tag{7.8}$$

Therefore, given that $Z(t) = n$, the joint conditional density of n arrival times is derived as follows:

$$f(t_1,\ t_2,\ \cdots,\ t_n \mid Z(t) = n) = n! \frac{\prod_{i=1}^{n} h(t_i)}{[H(t)]^n}. \tag{7.9}$$

Eq. (7.9) implies that unordered random variables of n arrival times $S_1, S_2, \cdots, S_n$ are independent and identically distributed with the density

$$f(x) = \begin{cases} \dfrac{h(x)}{H(t)} & (0 \le x \le t) \\ 0 & (\text{otherwise}), \end{cases} \tag{7.10}$$

if we condition that $Z(t) = n$ [41]. Of course, if $Z(t)$ obeys a *homogeneous Poisson process* (abbreviated as *HPP*) which is the special case for the NHPP, the n arrival times given $Z(t) = n$ are independent and identically distributed uniformly on the interval $[0,\ t]$.

Additionally, we also introduce a useful conditional probability related to the conditional arrival times distribution discussed above. If $s < t$ and $0 \le m \le n$, then

$$\begin{aligned}\Pr\{Z(s) = m \mid Z(t) = n\} &= \frac{\Pr\{Z(t-s) = n-m,\ Z(s) = m\}}{\Pr\{Z(t) = n\}} \\ &= \binom{n}{m} \left(\frac{H(s)}{H(t)}\right)^m \left(1 - \frac{H(s)}{H(t)}\right)^{n-m}.\end{aligned} \tag{7.11}$$

Eq. (7.11) implies that m events occured by time $s(< t)$ are independent and each event is occurred with a probability $H(s)/H(t)$ respectively, given that $Z(t) = n$. That is, the conditional distribution of $Z(s)$ given that $Z(t) = n$ obeys a binomial distribution with parameters $(n,\ H(s)/H(t))$.

7.3.2 Infinite server queueing modeling

We develop an infinite server queueing model for software reliability assessment based on the following assumptions:

(A-1) The expected cumulative number of software failures are observed according to an NHPP with the mean value function $\Lambda(t)$ and the intensity function $\lambda(t)$.

(A-2) The observed software failure is directly analyzed in the fault-isolation process when the software failure is observed. After the software failure analysis, the software fault is detected.

(A-3) The fault-isolation times are assumed to be independent with a common distribution $F(t)$.

Let a counting process $\{X(t),\ t \geq 0\}$ be the random variable indicating the cumulative number of software failures observed up to time t, and also a counting process $\{N(t),\ t \geq 0\}$ be the one indicating the cumulative number of faults detected up to time t. Suppose that the test has been begun at $t = 0$. Then, the distribution function of $N(t)$ is given by

$$\Pr\{N(t) = n\} = \sum_{j=0}^{\infty} \Pr\{N(t) = n \mid X(t) = j\} \frac{[\Lambda(t)]^j}{j!} e^{-\Lambda(t)}. \tag{7.12}$$

And, for j software failures observed up to time t, the probability that n faults are detected

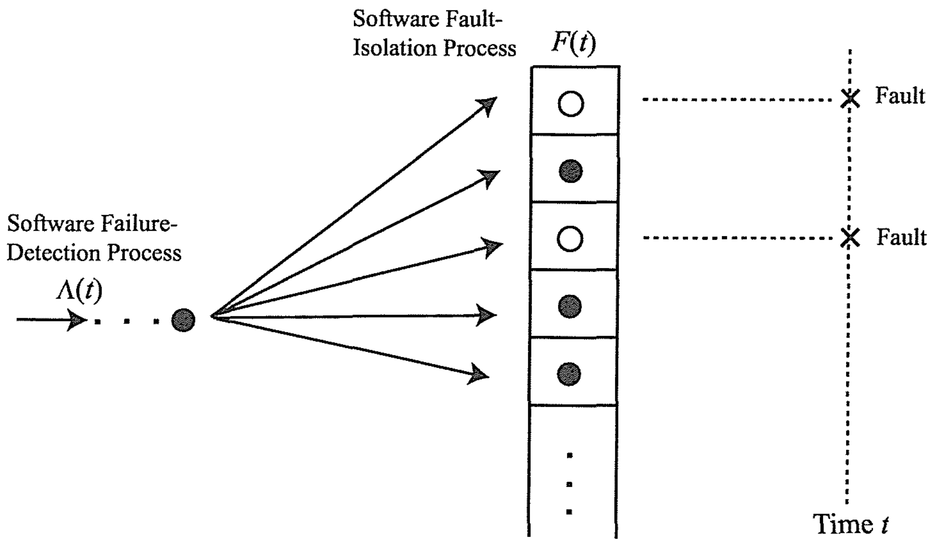


Fig. 7.2 : Our infinite server queueing model with the time distribution of fault-isolation process.

via the fault-isolation process is given as

$$\Pr\{N(t)=n \mid X(t)=j\} = \binom{j}{n}\{p(t)\}^n\{1-p(t)\}^{j-n}, \tag{7.13}$$

where $p(t)$ means the probability that an arbitrary one fault is detected by time t, and also $p(t)$ is given by

$$p(t) = \int_0^t F(t-x)\frac{d\Lambda(x)}{\Lambda(t)}, \tag{7.14}$$

by using the Stieltjes convolution and the concepts of the conditional arrival times distribution. Substituting Eqs. (7.13) and (7.14) into Eq. (7.12), we obtain the distribution function of the cumulative number of faults detected up to time t as

$$\Pr\{N(t)=n\} = \frac{[\int_0^t F(t-x)d\Lambda(x)]^n}{n!}\exp[-\int_0^t F(t-x)d\Lambda(x)]. \tag{7.15}$$

Eq. (7.15) is equivalent to an NHPP with mean value function $\int_0^t F(t-x)d\Lambda(x)$. That is, $N(t)$ has an NHPP with mean value function $\int_0^t F(t-x)d\Lambda(x)$. Fig. 7.2 shows the concept of our infinite server queueing modeling.

7.4 Relationship to Existing SRGM's

We have developed the infinite server queueing model for software reliability assessment, and derived an SRGM considering the time distribution of the fault-isolation process in the preceding section. Using Eq. (7.15), we can characterize the time-dependent behavior of fault-detection phenomenon by determining $\Lambda(t)$ and $F(t)$ which indicate the expected cumulative number of software failures observed up to time t and the time distribution function of fault-isolation process, respectively. Accordingly, we can easily reflect the physical phenomenon for the successive software failure-occurrence and the fault-detection on software reliability growth modeling concretely and simply.

And, Eq. (7.15) can be considered as an general description for several NHPP models. Specifically, for example, Eq. (7.15) is equivalent to the generalized delayed S-shaped SRGM in Eq. (7.5) essentially if $\Lambda(t)$ and $F(t)$ in Eq. (7.15) are supposed as

$$\Lambda(t) = a(1-e^{-bt}), \qquad F(t) = 1-e^{-\alpha t}, \tag{7.16}$$

respectively. In Eq. (7.16), a represents the expected initial fault content in the software system, b the failure-occurrence rate, $\alpha(>0)$ the reciprocal of the expectation of the exponential

Table 7.1 : Relationships between infinite server queueing models and existing NHPP models.

$\Lambda(t)$	$F(t)$	$M_S(t)$	Ref.
$a(1-\exp[-bt])$	$\mathbf{1}(t)$	$a(1-\exp[-bt])$	[52, 56]
$a(1-\exp[-bt])$	$T \sim EXP(\alpha)$	$a[1-\frac{1}{b-\alpha}(b\exp\{-\alpha t\}-\alpha\exp\{-bt\})]$	[62, 68]
$a(1-\exp[-\eta t])$	$T \sim EXP(\eta)$	$a[1-(1+\eta t)\exp\{-\eta t\}]$	[58, 66]
μt	$T \sim WEI(\alpha,\ m)$	$\mu[t-\frac{1}{m\alpha}\{m\Gamma_1(1+\frac{1}{m})-\Gamma_2(\frac{1}{m},(\alpha t)^m)\}]$	
$a[1-(1+bt)\exp\{-bt\}]$	$T \sim EXP(\alpha)$	$a[1-(1-bt+\frac{b^2(bt-\alpha t+1)}{(b-\alpha)^2})\exp\{-bt\}-\frac{b^2}{(b-\alpha)^2}\exp\{-\alpha t\}]$	
$a(1-r^t)$	$T \sim EXP(\alpha)$	$a\left[(1-r^t)+\frac{(r^t-\exp\{-\alpha t\})\log r}{\alpha+\log r}\right]$	

$(\alpha,\ \eta > 0,\ 0 < r < 1)$

$\mathbf{1}(t)$: unit function.

Γ_1 : gamma function, Γ_2 : incomplete gamma function.

EXP : exponential distribution.

WEI : Weibull distribution.

distribution, i.e., the expectation of the Poisson distribution. Furthermore, by supposing that $\eta = b = \alpha$ in Eq. (7.16), Eq. (7.15) is equivalent to the delayed S-shaped SRGM in Eq. (7.4). Thus, using Eq. (7.15), we can easily understand the physical interpretation of the fault-detection phenomenon, and reflect it on software reliability growth modeling. Table 7.1 summarizes the relationships between the infinite server queueing model and existing NHPP models where $M_S(t) = \int_0^t F(t-x)d\Lambda(x)$.

7.5 Numerical Examples

In this section we show several numerical examples for software reliability assessment by using actual observed data. We suppose that $\Lambda(t)$ and $F(t)$ are given as

$$\Lambda(t) = a(1 - r^t) \qquad (0 < r < 1), \tag{7.17}$$

and

$$F(t) = 1 - \exp[-\alpha t], \tag{7.18}$$

respectively. This case is one of the cases listed in Table 7.1. We apply the MLE to estimate the model parameters. Supposing that we observed K data pairs $(t_k,\ y_k)(k = 0, 1, 2, \cdots, K\ ;\ 0 < t_1 < t_2 < \cdots < t_K)$ with respect to the total faults, y_k, detected during constant time-interval $(0,\ t_k]$, we can derive the following logarithmic likelihood function from the properties of the NHPP:

$$\ln L = \sum_{k=1}^{K}(y_k - y_{k-1}) \cdot \ln[M_S(t_k) - M_S(t_{k-1})] - M_S(t_K) - \sum_{k=1}^{K} \ln[y_k - y_{k-1}]. \tag{7.19}$$

We can derive the following simultaneous equations by partially differentiating the logarithmic likelihood funtion in Eq. 7.19 with respect to the parameters a, r, and α, respectively:

$$\frac{\partial \ln L}{\partial a} = \frac{\partial \ln L}{\partial r} = \frac{\partial \ln L}{\partial \alpha} = 0. \tag{7.20}$$

Accordingly, by solving numerically the above equations, we can estimate $\widehat{a}$, $\widehat{r}$, and $\widehat{\alpha}$, which are the estimates of a, r, and α, respectively.

We use a PL/I application program test data consisting of 19 data pairs $(t_k,\ y_k)(k = 1, 2, \cdots, 19\ ;\ t_{19} = 19, y_{19} = 328)$ [32]. Fig. 7.3 shows the estimated mean value function $\widehat{M_S}(t)$ and the 95% confidence limits of it where the estimated parameters of $\widehat{M_S}(t)$ are $\widehat{a} = 459.08$, $\widehat{r} = 0.1916$, and $\widehat{\alpha} = 0.0682$. The $100\gamma\%$ confidence limits is derived as

$$\widehat{M_S}(t) \pm K_\gamma \sqrt{\widehat{M_S}(t)}, \tag{7.21}$$

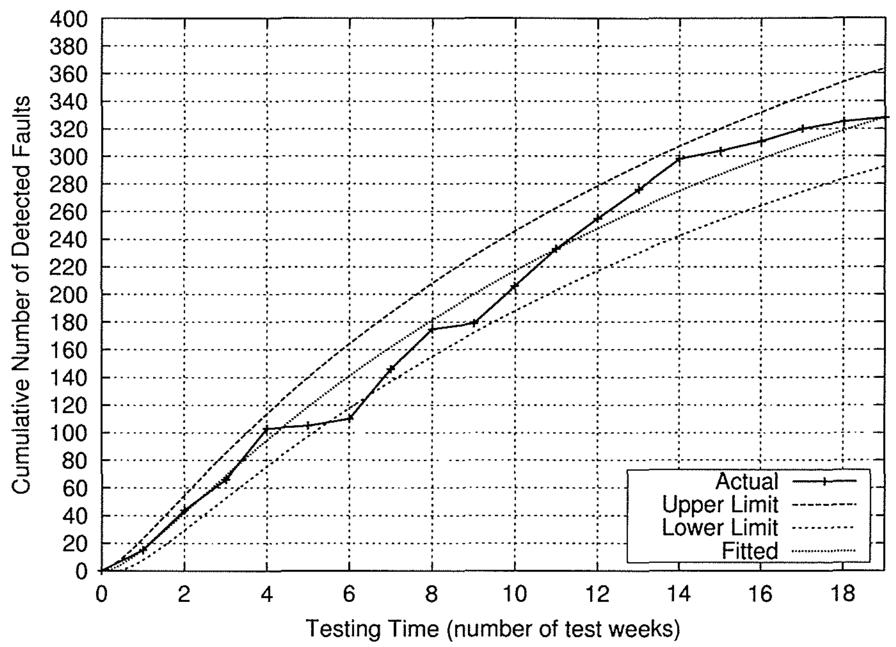


Fig. 7.3 : The estimated mean value function, $\widehat{M_S}(t)$.

where K_γ indicates the $100(1+\gamma)/2$ percent point of the standard normal distribution [58]. We also apply the K-S goodness-of-fit test discussed in Section 4.7 to evaluate whether $\widehat{M_S}(t)$ fits statistically to the observed data. We verified that $\widehat{M_S}(t)$ fits to the observed data with the 5% level of significance by the K-S goodness-of-fit testing.

7.5.1 Software reliability function

Given that the testing or the operation has been going up to time t, the probability that a software failure does not occur in the time-interval $[t,\ t+x)(x \geq 0,\ t \geq 0)$ is derived as

$$R_S(x \mid t) = \exp[-\{M_S(t+x) - M_S(t)\}], \tag{7.22}$$

from Eq. (7.1). Eq. (7.22) is called a software reliability function. We can estimate the software reliability by using this equation. Fig. 7.4 shows the estimated software reliability with respect to $t = 19$ (weeks) which is the termination time of the testing. Assuming that the software users operate the software product under the same environment as the testing, for example, we can estimate the software reliability $\widehat{R_S}(0.1 \mid 19)$ to be about $0 \cdot 410$.

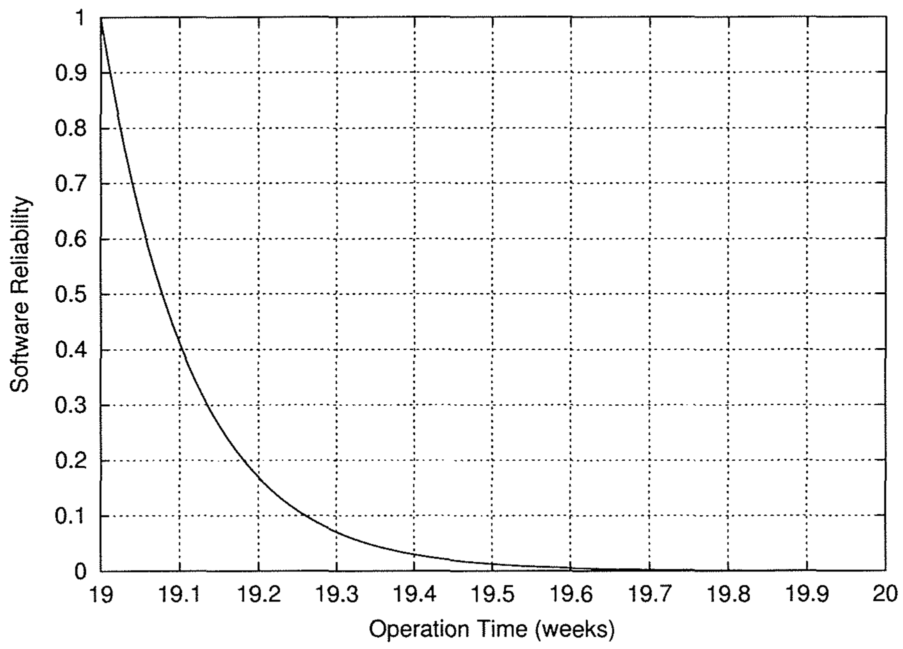


Fig. 7.4 : The estimated software reliability $\widehat{R_S}(x \mid 19)$.

7.5.2 Instantaneous MTBF

We also estimate an instantaneous MTBF which has been used as one of the substitution of measures of MTBF. An instantaneous MTBF can be obtained as

$$\mathrm{MTBF}_I(t) = \frac{1}{h_S(t)}, \tag{7.23}$$

where $h_S(t)$ indicates the intensity function of the NHPP. Fig. 7.5 shows the time-dependent behavior of the instantaneous MTBF in Eq. (7.23). Using Eq. (7.23), we can estimate the mean time between failures, $\mathrm{MTBF}_I(19)$, to be about $0 \cdot 112$(weeks).

7.6 Concluding Remarks

In this chapter, we have discussed an infinite server queueing model considering the time distribution of the fault-isolation process based on the concept of the delayed S-shaped software reliability growth modeling. Generally, it is considered that the time spent by analyzing the causes of each software failure is randomly behaved by difference of the difficulty of isolating and detecting each fault. Accordingly, this chapter has treated with the random behavior of the isolation times for each software fault by developing the infinite server queueing model. Additionally, this chapter has shown that this model can easily express the physical description

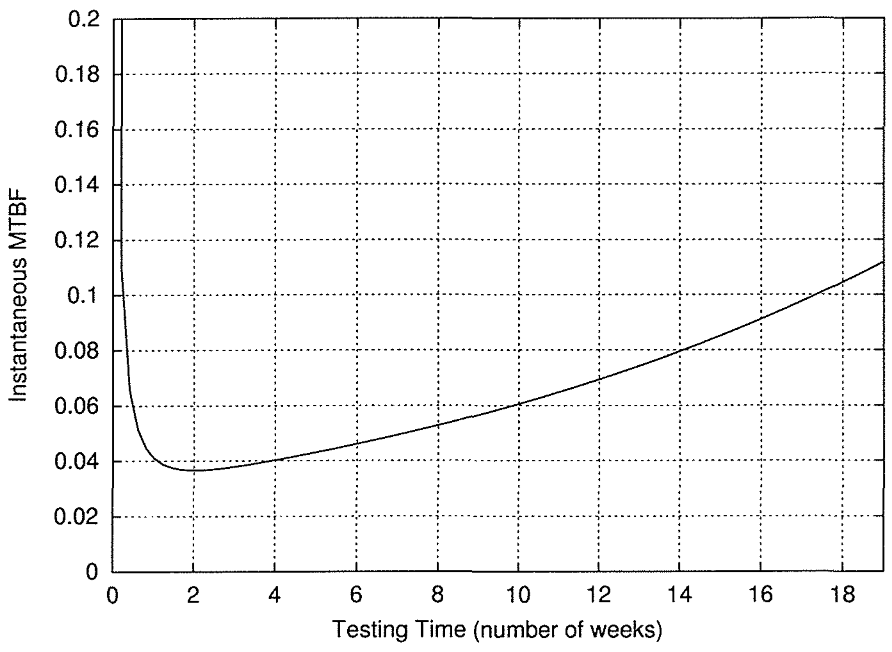


Fig. 7.5 : The estimated instantaneous MTBF, $\widehat{\mathrm{MTBF}}_I(t)$.

for the fault-detection phenomenon, and can describe several NHPP models as the special cases. Finally, having assumed that $\Lambda(t) = a(1 - r^t)$ and $F(t) = 1 - \exp[-\alpha t]$, we have shown the result of goodness-of-fit testing to the actual data and several numerical examples by using fault count data observed in the actual testing-phase.

Part IV

IMPERFECT DEBUGGING MODELING

Chapter 8

Software Reliability Modeling with Imperfect Debugging Activities

8.1 Introduction

Assessing software reliability in a testing-phase located in the final phase of the software development is one of the important project management activities to produce highly-reliable software systems. A software reliability assessment method based on a software reliability growth model has been known as one of the fundamental technologies for assesssing software reliability quantitatively, and applied for practical use. The SRGM is a mathematical tool based on probability and statistical theories, which can describe the software reliability growth process by regarding the number of faults detected up to arbitrary testing-time as a random variable. Therefore, most of the SRGM's have been developed by applying counting processes such as NHPP's.

The SRGM's can be classified into the following two models: perfect and imperfect debugging models. The perfect debugging models are modeled by assuming a perfect debugging environment in which faults latent in a software system are always detected and corrected perfectly by the debugging activities. On the other hand, the imperfect debugging models are developed by assuming an imperfect debugging environment where faults are not always detected and corrected perfectly and the debugging activities have a possibility that new faults are introduced. We can see that the imperfect debugging environment is a suitable assumption for software reliability growth modeling since debugging activities can not always detect correct faults perfectly in the actual testing-phase. Therefore, a lot of SRGM's developed by assuming the imperfect debugging environment have been proposed so far. Yamada [53], Yamada and Sera [61], Yamada et al. [69], and Zeephongsekul [72] have been proposed several types of imperfect debugging models based on NHPP's by considering that new faults are introduced by imperfect debugging activities, respectively. And Yamada et al. [70] has extented a well-known Goel-Okumoto model to an imperfect debugging model by using only a perfect debugging rate. Thus, imperfect debugging models proposed so far treat the following two types of imperfect debugging activities separately:

(1) The imperfect correcting activities which introduce new faults,

(2) The imperfect correcting activities which introduce no new faults.

However, it is possible that these two types of imperfect debugging activities are happened simultaneously in the actual testing-phase. Accordingly, we need to consider the two types of imperfect debugging activities simultaneously for developing plausible imperfect debugging models. In this chapter we develop imperfect debugging models which incorporate the two types of imperfect debugging activities simultaneously based on the NHPP's. Then, we derive several software reliability assessment measures based on our models. Further, we discuss estimation methods of model parameters, and show numerical examples of our model by using actual fault count data. Finally, we conduct goodness-of-fit comparisons among our models by using actual data sets.

8.2 Imperfect Debugging Modeling

8.2.1 NHPP model

We develop imperfect debugging models based on NHPP's. An NHPP model is one of the SRGM's having the following stochastic properties with respect to a counting process, $\{N(t), t \geq 0\}$, representing the number of faults detected during a constant time-interval $(0, t]$:

$$\left.\begin{aligned} \Pr\{N(t)=n\} &= \frac{\{H(t)\}^n}{n!}\exp\{-H(t)\} \\ &\qquad\qquad\qquad (n=0,1,2,\cdots) \\ H(t) &= \int_0^t h(x)dx \end{aligned}\right\}, \tag{8.1}$$

where $H(t)$ is called the mean value function which represents the expected number of faults detected during the time-interval $(0, t]$, and $h(t)$ the intensity function representing the instantaneous fault-detection rate. The stochastic behavior of the fault-detection phenomenon can be characterized by assuming a suitable mean value function $H(t)$. Most of the mean value functions are developed by assuming basically that the number of faults detected at testing-time

t is proportional to the current residual fault content. Accordingly, we can obtain the following differential equation characterizing the time-dependent behavior of the expected number of detected faults:

$$\frac{dH(t)}{dt} = b(t)[a - H(t)] \qquad (a > 0,\ b(t) \geq 0), \tag{8.2}$$

if we assume the perfect debugging environment. In Eq. (8.2), a represents the expected initial fault content, and $b(t)$ the fault-detection rate per fault at testing-time t.

8.2.2 Modeling with 2-types of imperfect debugging activities

We propose NHPP models by considering two types of imperfect debugging activities, such as the activities which introduce new faults and the activities which introduce no new faults. Our imperfect debugging models are developed by expanding the basic assumptions of Eq. (8.2). That is, we expand Eq. (8.2) into the following differential equation:

$$\frac{dH(t)}{dt} = b(t)[a(t) - pH(t)] \qquad (a(t) \geq a,\ 0 < p \leq 1), \tag{8.3}$$

where p represents the perfect debugging rate, $a(t)$ the number of faults in the software system at testing-time t considering with the imperfect debugging which introduces new faults. Accordingly, $(1-p)$ denotes the probability of the imperfect debugging which introduce no new faults. Solving Eq. (8.3) with respect to $H(t)$, we obtain the following equation:

$$\left.\begin{aligned} H(t) &= e^{-p\cdot Z(t)}\left[\int_0^t a(s)b(s)e^{p\cdot Z(s)}ds\right] \\ Z(t) &= \int_0^t b(s)ds \end{aligned}\right\}. \tag{8.4}$$

We need to give suitable functions $a(t)$ and $b(t)$ in Eq. (8.4) to develop a specific imperfect debugging model, respectively. As to the function $a(t)$, it is very difficult to observe it when new faults are introduced. Therefore, we consider the following two functions which describe the time-dependent behavior of the expected total numbers of faults in the software system by taking the numbers of introduced faults into consideration:

$$a_1(t) = \alpha_1 \exp[\beta t] \qquad (\beta > 0), \tag{8.5}$$

$$a_2(t) = \alpha_2(1 + \gamma t) \qquad (\gamma > 0), \tag{8.6}$$

respectively. In Eqs. (8.5) and (8.6), α_i $(i = 1, 2)$ represent the expected numbers of initial inherent faults, β and γ new faults introduction rates for the expected numbers of initial inherent faults, respectively. We can see that Eqs. (8.5) and (8.6) mean that the expected numbers of faults in the software system increase exponentially and linearly with constant increasing rates, respectively. Substituting Eqs. (8.5) and (8.6) into Eq. (8.4) and solving Eq.(8.4), we can obtain the following equations:

$$H_1(t) = \frac{\alpha_1 b_1}{pb_1 + \beta}(\exp[\beta t] - \exp[-pb_1 t]), \tag{8.7}$$

$$H_2(t) = \frac{\alpha_2}{p}\left\{(1 - \frac{\gamma}{pb_2})(1 - \exp[-pb_2 t]) + \gamma t\right\}, \tag{8.8}$$

respectively, where we set $b(t) \equiv b_i (i = 1, 2)$ (constant values). In this chapter we call the NHPP models with Eqs. (8.7) and (8.8) as Model 1 and Model 2, respectively.

In Eqs. (8.7) and (8.8), we can see that the mean value functions, $H_i(t)$ $(i = 1, 2)$, have the following properties:

$$\left.\begin{array}{l} H_1(0) \ \ = H_2(0) = 0 \\ H_1(\infty) = H_2(\infty) = \infty \end{array}\right\}. \tag{8.9}$$

That is, the numbers of faults detected in infinitely long duration are infinity, respectively, since these functions assume that new faults are introduced when debugging activities are conducted. In existing imperfect debugging models, the Littlewood-Verrall model [25], the Weibull process model [29], and the logarithmic Poisson execution time model [29] have the same properties as Eq. (8.9) especially.

8.3 Reliability Assessment Measures

After characterizing the software failure-occurrence or fault-detection phenomenon by using an SRGM, it is important to utilize several software reliability assessment measures derived from the applied SRGM for quantitative assessment of software reliability. In this section we derive reliability assessment measures, such as software reliability functions and hazard rates, based on our proposed models.

8.3.1 Software reliability function

The software reliability function is one of the well-known software reliability assessment measures. Given that the testing or the operation has been going up to testing-time t, the software reliability function is defined as the probability that a software failure does not occur in the time-interval $(t, t+x](t \geq 0,\ x \geq 0)$. Accordingly, the software reliability function $R(x \mid t)$ can be formulated as

$$R(x \mid t) \equiv \exp[-\{H(t+x) - H(t)\}], \tag{8.10}$$

based on properties of the NHPP. Substituting Eqs. (8.7) and (8.8) into Eq. (8.10), we can derive software reliability functions as

$$R_1(x \mid t) = \exp\Big[-\frac{\alpha_1 b_1}{pb_1+\beta}(\exp[\beta(t+x)] - \exp[-pb_1(t+x)] - \exp[\beta t]) + \exp[-pb_1 t]\Big], \tag{8.11}$$

$$R_2(x \mid t) = \exp\Big[\frac{\alpha_2}{p}\Big\{(1-\frac{\gamma}{pb_2})(\exp[-pb_2(t+x)] - \exp[-pb_2 t]) - \gamma t\Big\}\Big], \tag{8.12}$$

respectively.

8.3.2 Hazard rate

The hazard rate represents the frequency of software failure-occurrence per unit testing-time, and is formulated as

$$z(x \mid t) \equiv \frac{-\frac{d}{dx}R(x \mid t)}{R(x \mid t)} = h(t+x), \tag{8.13}$$

where $z(x \mid t)\Delta t$ represents the probability that a software failiure occurs during a small time-interval $(t+x, t+x+\Delta t]$ given that a software failure has not been occurring during a time-interval $(t, t+x](t \geq 0,\ x \geq 0)$. Substituting Eqs. (8.7) and (8.8) into Eq. (8.13), we obtain the following hazard rate functions:

$$z_1(x \mid t) = \frac{\alpha_1 b_1}{pb_1+\beta}(\beta\exp[\beta(t+x)] + pb_1\exp[-pb_1(t+x)]), \tag{8.14}$$

$$z_2(x \mid t) = \frac{\alpha_2}{p}\left\{\exp[-pb_2(t+x)](pb_2-\gamma) + \gamma\right\}, \tag{8.15}$$

respectively.

8.4 Parameter Estimations

We discuss the methods of parameter estimation for our imperfect debugging models. First of all, we discuss parameters p, β, and γ in Eqs. (8.7) and (8.8). Ordinarily, estimating these parameters by using fault count data or software failure-occurrence times data is very difficult. Therefore, we specify these parameters experimentally in this case. However, we set the parameters β and γ by using fault introduction rates c_i $(i = 1, 2)$ formulated as

$$c_i = \frac{a_i(T) - \alpha_i}{\alpha_i} \qquad (i = 1, 2), \tag{8.16}$$

respectively. In Eq. (8.16), T denotes the completion time of the testing. Using Eqs. (8.5) and (8.6), we can derive the fault introduction rates as

$$c_1 = \exp[\beta T] - 1, \tag{8.17}$$

$$c_2 = \gamma T, \tag{8.18}$$

respectively. In the case that the fault introduction rates have been given as $\bar{c}_i$ $(i = 1, 2)$, the parameters β and γ can be calculated as

$$\beta = \frac{1}{T} \log(\bar{c}_1 + 1), \tag{8.19}$$

$$\gamma = \frac{1}{T} \bar{c}_2, \tag{8.20}$$

from Eqs. (8.17) and (8.18), respectively.

Next we discuss the methods of parameter estimation for α_i and b_i $(i = 1, 2)$. In this chapter we estimate the parameters α_i and $b_i (i = 1, 2)$ based on the methods of maximum-likelihood by using the set and calculated parameters p, β, and γ, respectively. Supposing that K data pairs $(t_k,\ y_k)(k = 0, 1, 2, \cdots, K)$ have been observed with respect to the cumulative number of faults, y_k, detected during a constant time-interval $(0,\ t_k](0 < t_1 < t_2 < \cdots < t_K)$, we can derive the logarithmic likelihood functions as

$$\ln L_i = \sum_{k=1}^{K} (y_k - y_{k-1}) \cdot \ln[H_i(t_k) - H_i(t_{k-1})] - H_i(t_K) - \sum_{k=1}^{K} \ln[y_k - y_{k-1}] \qquad (i = 1, 2), \tag{8.21}$$

based on the properties of NHPP's [36, 41, 50]. Accordingly, the parameter estimates $\widehat{\alpha}_i$ and $\widehat{b}_i$ $(i = 1, 2)$ of parameters α_i and b_i can be obtained by solving the following simultaneous likelihood functions numerically:

$$\frac{\partial \ln L_i}{\partial \alpha_i} = \frac{\partial \ln L_i}{\partial b_i} = 0 \qquad (i = 1, 2). \tag{8.22}$$

which are derived from Eq. (8.21), respectively.

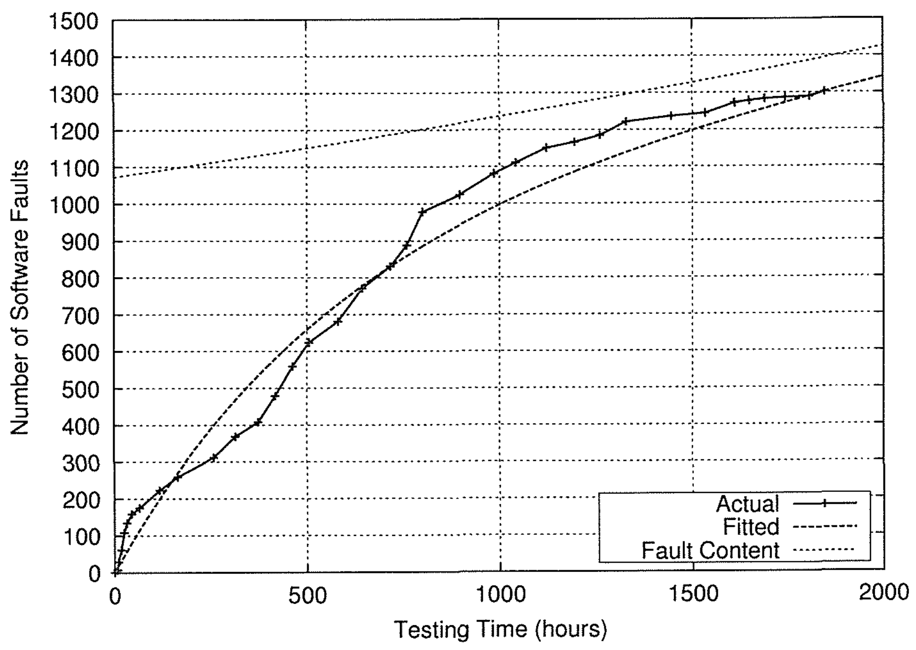


Fig. 8.1 : The estimated mean value function, $\widehat{H}_1(t)$. (Model 1 ; $p = 0.95$, $\bar{c}_1 = 0.3$, $\beta = 1.421 \times 10^{-4}$)

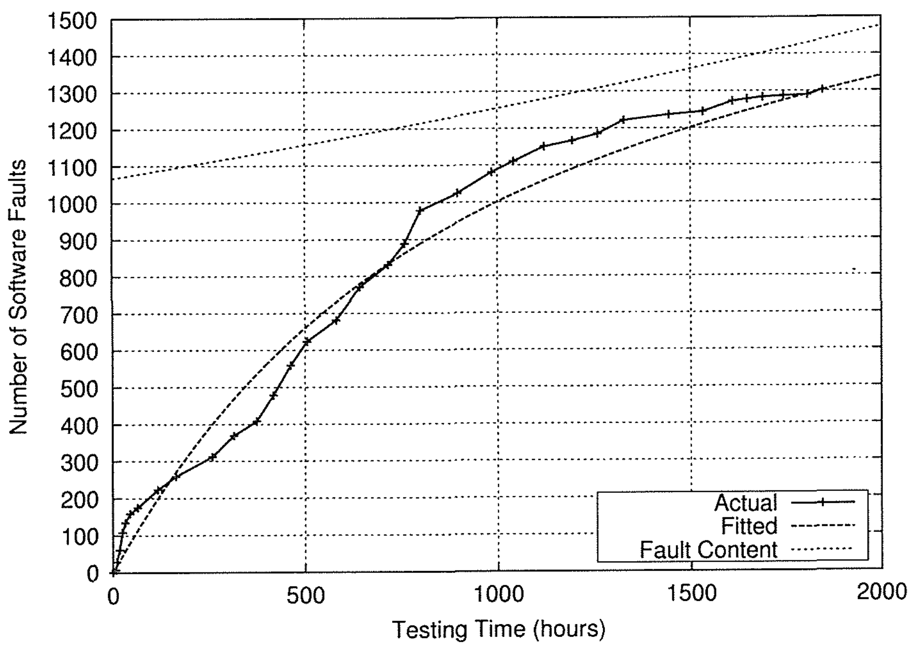


Fig. 8.2 : The estimated mean value function, $\widehat{H}_2(t)$. (Model 2 ; $p = 0.95$, $\bar{c}_2 = 0.3$, $\beta = 1.624 \times 10^{-4}$)

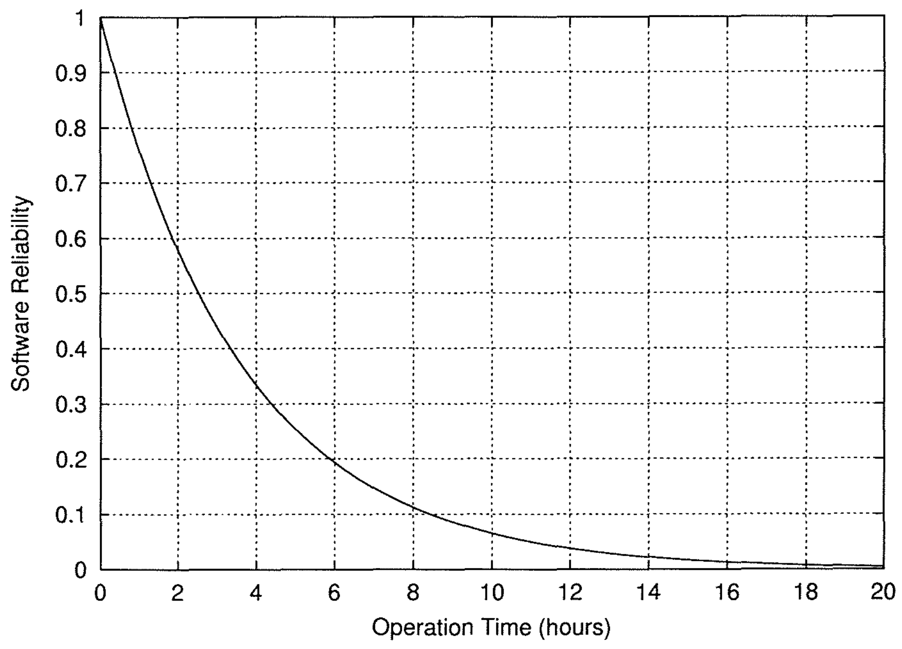


Fig. 8.3 : The estimated software reliability function, $\widehat{R}_1(x \mid 1846.92)$.

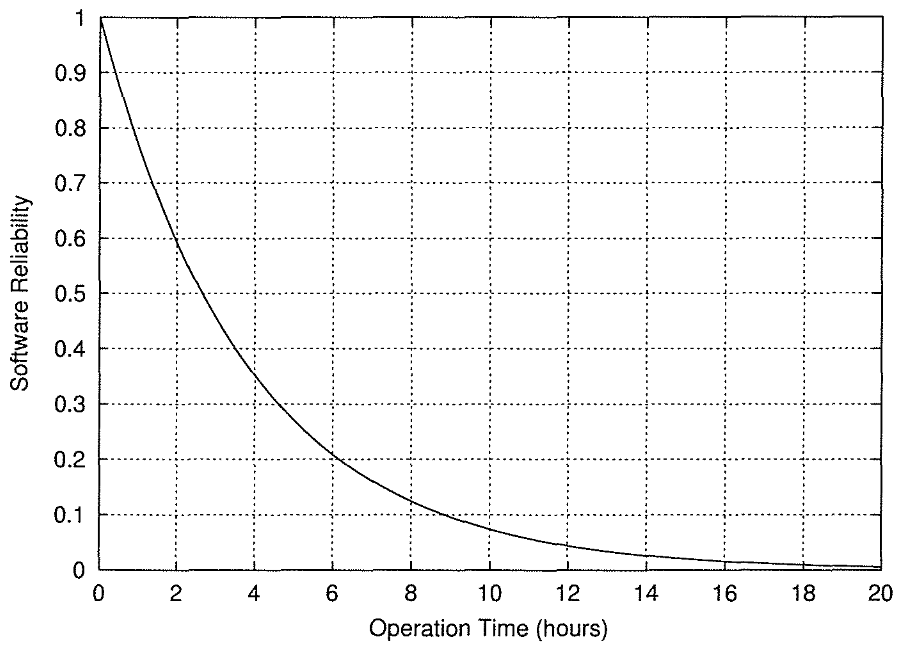


Fig. 8.4 : The estimated software reliability function, $\widehat{R}_2(x \mid 1846.92)$.

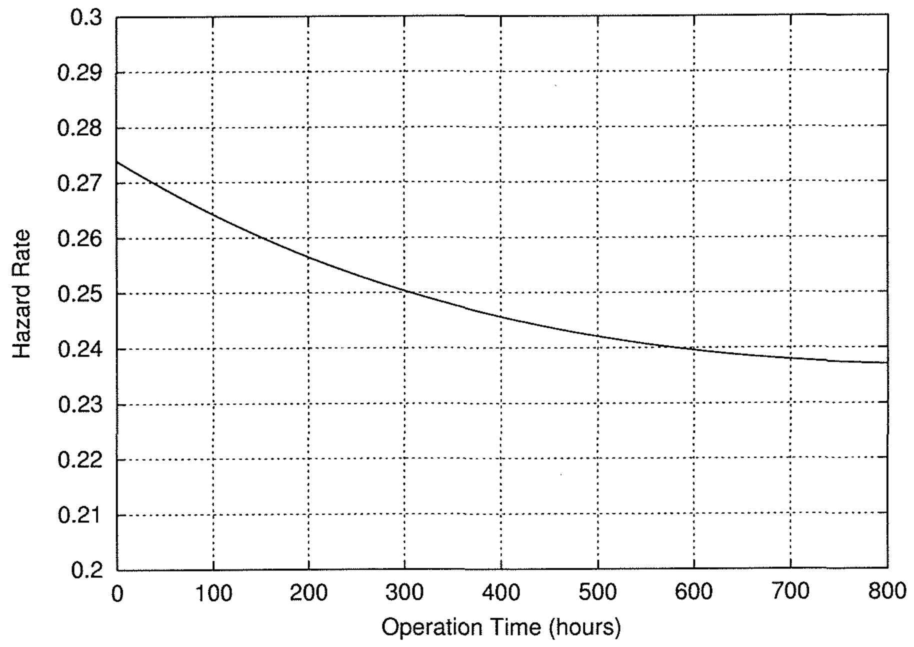


Fig. 8.5 : The estimated hazard rate function, $\widehat{z}_1(x \mid 1846.92)$.

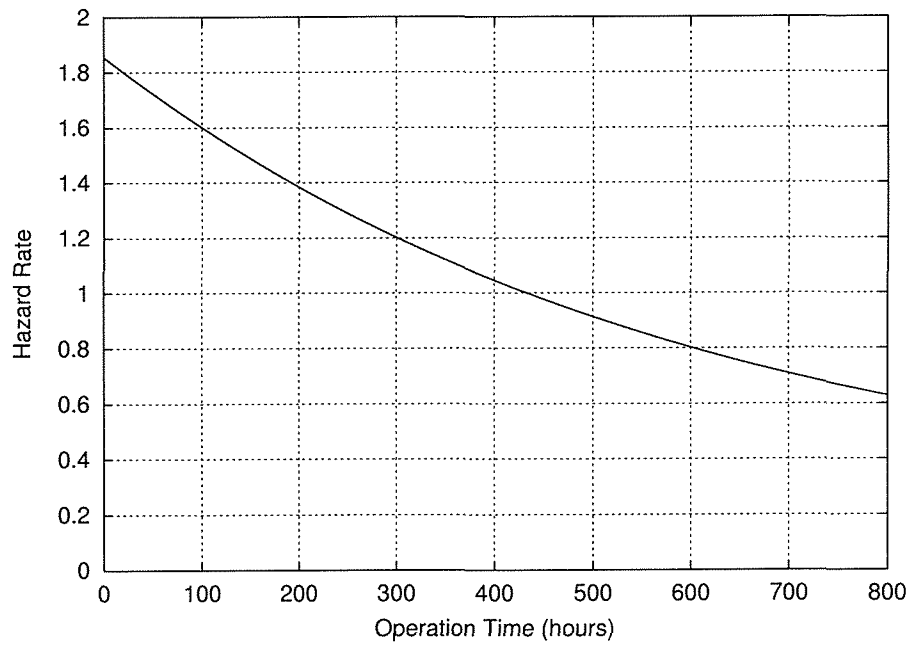


Fig. 8.6 : The estimated hazard rate function, $\widehat{z}_2(x \mid 1846.92)$.

8.5 Numerical Examples

We show numerical examples of our models by using actual fault count data. This data set consists of 35 data pairs $(t_k, y_k)(k = 1, 2, \cdots, 35;\ t_{35} = 1846.92$ (hours), $y_{35} = 1301)$ [6]. In this numerical examples we set $p = 0.95$ and $\bar{c}_i = 0.3$ $(i = 1, 2)$. Then, we can calculate $\beta = 1.421 \times 10^{-4}$ and $\gamma = 1.624 \times 10^{-4}$ by using Eqs. (8.19) and (8.20), respectively. Using these parameters set as above, we can obtain parameter estimates $\widehat{a}_1 = 1.073 \times 10^3$, $\widehat{b}_1 = 1.730 \times 10^{-3}$, $\widehat{a}_2 = 1.066 \times 10^3$, and $\widehat{b}_2 = 1.740 \times 10^{-3}$ by using the methods of maximum-likelihood, respectively.

Figs. 8.1 and 8.2 depict the estimated mean value functions, $\widehat{H}_1(t)$ and $\widehat{H}_2(t)$, respectively. And, Figs. 8.3 and 8.4 show the estimated software reliabilities $\widehat{R}_1(x \mid 1846.92)$ and $\widehat{R}_2(x \mid 1846.92)$ after the terminaiton time of the testing ($t_{35} = 1846.92$ (hours)), respectively. If we assume that the developed software system is used in the operational phase which is assumed to have the same environment of the testing-phase, we can estimate the software reliabilities after 5 hours from the termination time of the testing to be about $\widehat{R}_1(5 \mid 1846.92) \approx 0.255$ and $\widehat{R}_2(5 \mid 1846.92) \approx 0.271$ from Figs. 8.3 and 8.4, respectively. Furthermore, Figs. 8.5 and 8.6 show the estimated hazard rates, $\widehat{z}_1(x \mid 1846.92)$ and $\widehat{z}_2(x \mid 1846.92)$, respectively. From Figs. 8.5 and 8.6, we can estimate the hazard rates after 800 hours from the termination time of the testing to be about $\widehat{z}_1(800 \mid 1846.92) \approx 0.237$ and $\widehat{z}_2(800 \mid 1846.92) \approx 0.628$.

8.6 Goodness-of-fit Comparisons

We compare our two types of imperfect debugging models proposed in this chapter in terms of mean square errors (MSE) discussed in Section 2.4. In these model comparisons the parameters are set $p = 0.95, \beta = 1.421 \times 10^{-4}$, and $\gamma = 1.624 \times 10^{-4}$ which are the same as in Section 8.5, and we use the following fault count data:

- DS1 [6] : $(t_k, y_k)(k = 1, 2, \cdots, 35\ ;\ t_{35} = 35, y_{35} = 1301)$ where t_k is measured on the basis of hours.
- DS2 [6] : $(t_k, y_k)(k = 1, 2, \cdots, 19\ ;\ t_{19} = 19, y_{19} = 328)$ where t_k is measured on the basis of weeks.
- DS3 [12] : $(t_k, y_k)(k = 1, 2, \cdots, 24\ ;\ t_{24} = 24, y_{24} = 296)$ where t_k is measured on the basis of weeks.
- DS4 [44] : $(t_k, y_k)(k = 1, 2, \cdots, 59\ ;\ t_{59} = 59, y_{59} = 5186)$ where t_k is measured on the basis of weeks.

Table 8.1 : The results of model comparisons based on the MSE. ($p = 0.95$, $\beta = 1.421 \times 10^{-4}$, and $\gamma = 1.624 \times 10^{-4}$)

	DS1	DS2	DS3	DS4
Model 1	4051.39	243.375	690.438	49086.4
Model 2	**3913.79**	**237.192**	**674.719**	**45486.4**

Table 8.1 shows the results of model comparisons based on the MSE. In Table 8.1, we can say that Model 2 has better performance than Model 1 in terms of the MSE when we set the parameter as $p = 0.95, \beta = 1.421 \times 10^{-4}$, and $\gamma = 1.624 \times 10^{-4}$. That is, we can also say that it is better to assume a linearly increasing function representing the expected number of faults considering with new introduced faults like Eq. (8.6) in these model comparisons.

8.7 Concluding Remarks

We have proposed two types of imperfect debugging models based on the NHPP's, considering with two types of imperfect debugging activities simultaneously, such as the activities which introduce new faults and imperfect fault correction activities. Then, we have derived software reliability assessment measures, such as the reliability functions and the hazard rate functions, based on the stochastic properties of the NHPP's. Further, we have discussed the methods of parameter estimation for our proposed models, and shown numerical examples by using actual fault count data. Finally, we have conducted goodness-of-fit comparisons among our proposed imperfect debugging models in terms of the MSE by using four actual data sets.

Part V

Closing

Chapter 9

Conclusion

This dissertation has discussed stochastic software reliability growth modeling for the purpose of improvement of software reliability assessment. The main contributions obtained and the future studies remaining in the respective chapters are summarized as follows:

The first part has discussed discretized software reliability growth modeling and its application to optimal software release problems.

- **Chapter 2** has provided discrete NHPP models by using Hirota's bilinearization methods, which have better performance than the discretized SDA models in terms of the MSE and the predicted relative errors, and has derived optimal software release policies based on our discrete NHPP models as one of the interesting issues for an application technique to software project management. Compared with the case for the continuous NHPP models, our discrete NHPP models proposed in Chapter 2 are expected to contribute to decreasing labor of software development managers for software reliability assessment because the proposed parameter estimation procedures of our models can get estimates easily. And we can also obtain the inflection parameter along with the other parameters in applying the discrete inflection S-shaped SRGM to the observed data, simultaneously. On the continuous-time inflection S-shaped SRGM, it is known that we cannot obtain the inflection parameter along with the other parameter estimates accurately. Further studies are needed to examine the goodness-of-fit of the proposed two discrete NHPP models by using more observed data sets, and to deal with problems on software development management such as estimating the optimal allocation of testing-effort expenditures. And also, the optimal software release policies derived in Chapter 2 need to be investigated in actual software testing in terms of the validity.
- **Chapter 3** has developed stochastic models by incorporating the discrete NHPP's into

the discrete SDA models including the discrete modified exponential curve model which has been derived in this dissertation. Our proposed models in Chapter 3 can describe software reliability growth process stochastically with conserving the basic properties of the discrete SDA models. By developing stochastic models based on the discrete SDA models, several software reliability assessment measures which are useful metrics for quantitative software reliability assessment can be derived. We need further studies so that we can examine performance of the proposed model more by using many sets of faults count data collected in actual software projects.

The second part has provided software reliability growth modeling with several factors, such as testing-coverage and testing-effort expenditures, related to the software reliability growth process.

- **Chapter 4** has discussed software reliability growth modeling with testing-coverage maturity process in the testing-phase. Especially, the testing-coverage maturity process with testing-skill of test-case designers has been described by developing the alternative testing-coverage function. After that, the SRGM with testing-coverage maturity process has been developed by characterizing the relationship between the testing-coverage maturity process and the software reliability growth process. Incorporating the testing-coverage maturity process into describing the software reliability growth process is very significant for improvement of the accuracy of software reliability assessment. However, we have to research more about the useful software reliability assessment measures, and examine performance of the proposed model by using several actual data which are measured on the C1 and path testing-coverage criteria.

- **Chapter 5** has provided a lognormal process SRGM with testing-effort expenditures. The testing-effort is one of the important metrics influencing the software reliability growth process. Software developing managers can grasp the relationship between the attained software reliability and the amount of testing-effort expenditures by using our software reliability growth model proposed in Chapter 5, and our model also enables software development managers to decide how much testing-effort are expended to attain the reliability objective. However, further studies are needed to examine the validity of our model for practical applications by using more actual data.

In the third part generalization techniques for continuous or discrete-time software reliability growth modeling. Our generalization frameworks for software reliability modeling are different from the conventional framework under which an SRGM is developed by solving a differential equation modeled on suitable assumptions. Our generalized frameworks proposed in this dissertation have a useful characteristic that software development managers can easily obtain a feasible SRGM by analyzing the fault-detection phenomenon.

- **Chapter 6** has discussed generalized discrete software reliability growth modeling. Especially, generalized discrete binomial process models with the effect of the program size have been proposed. And optimal software release policies based on our models have been also derived. The discrete binomial process SRGM's proposed in Chapter 6 are suitable for software reliability assessment for a small or medium size software product since the binomial distribution in Eq. (6.3) representing the initial number of faults in the software system can be regarded as a Poisson distribution as the parameter $K \to \infty$. Our generalized discrete modeling for software reliability aseessment proposed in Chapter 6 enables us to obtain a suitable SRGM easily by analyzing the software failure-occurrence times distribution in the actual testing-phase and applying its suitable probability distribution function to our generalized discrete model. In Chapter 6, though we have applied the geometric and Rayleigh distributions as the software failure-occurrence times distributions, we have been planning to develop feasible software failure-occurrence times distributions which enable us to describe the times distribution flexibly in the future. And then, we have to discuss the validity of our generalized discrete model for actual software reliability assessment in the future studies.

- **Chapter 7** has provided a generalization framework for software reliability growth modeling based on the NHPP by using an infinite server queueing theory. By using this generalized SRGM, the time-dependent behavior of fault-detection phenomenon is characterized by $\Lambda(t)$ and $F(t)$ which indicate the mean value function of software failure-occurrence phenomenon and the distribution function of isolation time, respectively. But, there are several problems for applying this SRGM to the actual testing-phase. We have shown numerical examples in Chapter 7 by assuming that $\Lambda(t) = a(1-r^t)$ and $F(t) = 1-\exp[-\alpha t]$ intuitively. However, we have to research on how to assume $\Lambda(t)$ and $F(t)$ in actual testing-phase, which is an important issue for practical software reliability assessment as future studies.

The fourth part discusses software reliability growth modeling under imperfect debugging environment. The imperfect debugging modeling is considered as one of the effective approach for feasible software reliability growth modeling because debugging activities do not always remove faults perfectly in an actual testing-phase.

- **Chapter 8** has proposed imperfect debugging models with 2-types of imperfect debugging activities, such as the activities introducing new faults and the imperfect fault-correction activities, simultaneously based on NHPP's. Our imperfect debugging models have been developed by extending the basic assumptions of software reliability modeling based on the NHPP. In further studies, we have been planning to develop more plausible imperfect debugging models which can describe software reliability growth process with the two types of imperfect debugging activities more specifically by using other suitable stochastic processes. And we have to evaluate the validity and usefulness of our models for practical software reliability assessment.

We have discussed stochastic modeling for developing feasible software reliability growth models and accurate software reliability assessment through this dissertation comprehensively. As the future studies, we have to discuss application methods of our models to several software project mangement isssues, such as the earned value management (abbreviated as EVM) and software project risk management. And, these models proposed in this dissertation can be regard as black-box models which ignore the software architecture and the control flow of input data. Accordingly, we need to research on white-box software reliability modeling, such as the modeling with module compositions, for accurate software reliability assessment. Additionally, we have to research on software reliability assessment methodologies for recent software developent environment such as an open source software development project as the future studies.

References

[1] H. Akaike, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, Vol. AC-19, pp. 716–723, 1974.

[2] F.M. Bass, "A new product growth model for consumer durables," *Management Science*, Vol. 15, No. 5, pp. 215–227, January 1969.

[3] A. Behforooz and F.J. Hudson, *Software Engineering Fundamentals.* Oxford University Press, New York, 1996.

[4] A. Birolini, *Quality and Reliability of Technical Systems —Theory, Practice, Management.* 2nd ed., Berlin, Heidelberg, 1997.

[5] B.W. Boehm, "Software engineering," *IEEE Transactions on Computers*, Vol. C-25, pp. 1226–1241, 1976.

[6] W.D. Brooks and R.W. Motley, "Analysis of discrete software reliability models," Technical Report RADC-TR-80-84, Rome Air Development Center, New York, 1980.

[7] K.Y. Cai, *Software Defect and Operational Profile Modeling.* Kluwer Academic Publishers, Massachusetts, 1998.

[8] T. Dohi, S. Osaki, and K.S. Trivedi, "An infinite server queueing approach for describing software reliability growth ∼ unified modeling and etimation framework ∼, " *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, 2004, pp. 110–119.

[9] T. Dohi, K. Yasui, and S. Osaki, "Software reliability assessment models based on cumulative Bernoulli trial processes," *Mathematical and Computer Modelling*, Vol. 38, No. 11–13, pp. 1177–1184, 2003.

[10] W.W. Everett and J.D. Musa, "A software reliability engineering practice," *IEEE Computer Magazine*, Vol. 26, No. 3, pp. 77–79, March 1993.

[11] E.H. Foreman and N.D. Singpurwalla, "Optimal time intervals for testing-hypotheses on computer software errors," *IEEE Transactions on Reliability*, Vol. R-28, No. 3, pp. 250–253, August 1979.

[12] T. Fujiwara and S. Yamada, "C0 coverage-measure and testing-domain metrics based on a software reliability growth model," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 9, No. 4, pp. 329–340, 2002.

[13] A.L. Goel and K. Okumoto, "Time-dependent error-detection rate model for software reliability and other performance measures," *IEEE Transactions on Reliability*, Vol. R-28, No. 3, pp. 206–211, August 1979.

[14] R. Hirota, "Nonlinear partial difference equations. V. Nonlinear equations reducible to linear equations," *Journal of the Physical Society of Japan*, Vol. 46, No. 1, pp. 312–319, January 1979.

[15] C.Y. Huang, M.R. Lyu, and S.Y. Kuo, "A unified scheme of some nonhomogeneous Poisson process models for software reliability estimation," *IEEE Transactions on Software Engineering*, Vol. 29, No. 3, pp. 261–269, 2003.

[16] W.S. Humphrey, *Managing the Software Process.* Addison Wesley, New York, 1989.

[17] S.H. Kan, *Metrics and Models in Software Quality Engineering.* 2nd ed., Addison-Wesley, Boston, 2003.

[18] K. Kanoun, M. Kaaniche, and J.C. Laprie, "Qualitative and quantitative reliability assessment," *IEEE Software*, Vo. 14, No. 2, pp. 77–87, 1987.

[19] P.K. Kapur and S. Younes, "Modelling an imperfect debugging phenomenon in software reliability," *Microelectronics and Reliability*, Vol. 36, No. 5, pp. 645–650, 1996.

[20] J. Keyes, *Software Engineering Handbook.* Auerbach Publications, New York, 2002.

[21] M. Kijima, *Stochastic Processes with Applications to Finance.* Chapman & Hall/CRC, Boca Raton, 2003.

[22] M. Kimura, T. Toyota, and S. Yamada, "Economic analysis of software release problems with warranty cost and reliablity requirement," *Reliability Engineering and System Safety*, Vol. 66, No. 1, pp. 49–55, 1999.

[23] M. Kimura, S. Yamada, H. Tanaka, and S. Osaki, "Software reliability measurement with prior-information on initial fault content," *Transactions of Information Processing Society of Japan*, Vol. 34, No. 7, pp. 1601–1609, 1993.

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

[25] B. Littlewood and J.L. Verrall, "A Bayesian reliability growth model for computer software," *Journal Royal of Statistical Society-Series C*, Vol. 22, No. 3, pp. 332–346, 1973.

[26] D.G. Luenberger, *Investment Science.* Oxford University Press, New York, 1998.

[27] Y.K. Malaiya, M.N. Li, J.M. Bieman, and R. Karcich, "Software reliability growth with test coverage," *IEEE Transactions on Reliability*, Vol. 51, No. 4, pp. 420–426, 2002.

[28] F. Morishita, "The fitting of the logistic equation to the rate of increase of population density," *Research on Population Ecology*, Vol. VII, pp. 52–55, 1965.

[29] J.D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application.* McGraw-Hill, New York, 1987.

[30] T. Nakagawa and S. Osaki, "The discrete Weibull distribution," *IEEE Transactions on Reliability*, Vol. R-24, No. 5, pp. 300–301, 1975.

[31] Y. Nakamura (ed.), *Applied Integrable Systems* (in Japanese). Shokabo, Tokyo, 2000.

[32] M. Ohba, "Software reliability analysis models," *IBM Journal of Research and Development*, Vol. 28, No. 4, pp. 428–443, July 1984.

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

[34] H. Okamura, A. Murayama, and T. Dohi, "EM algorithm for discrete software reliability models: A unified parameter estimation method," *Proceedings of the 8th IEEE International Symposium on High Assurance Systems Engineering*, 2004, pp. 219–228.

[35] K. Okumoto and A.L. Goel, "Optimum release time for software system based on reliability and cost criteria," *Journal of Systems and Software*, Vol. 1, No. 4, pp. 315–318, 1980.

[36] S. Osaki, *Applied Stochastic System Modeling.* Springer-Verlag, Berlin, Heidelberg, 1992.

[37] B. Øksendal, *Stochastic Differential Equations : An Introduction with Applications.* Springer-Verlag, Berlin, 1985.

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

[39] H. Pham and X. Zhang, "NHPP software reliability and cost models with testing coverage," *European Journal of Operational Research*, Vol. 145, No. 2, pp. 443–454, 2003.

[40] K. Rinsaka and H. Sandoh, "A study on software maintenance service contracts," (in Japanese) *Transactions of IEICE*, Vol. J82-A, No. 12, pp. 1819–1829, December 1999.

[41] S.M. Ross, *Applied Probability Models with Optimization Applications.* Dover Publications, New York, 1992.

[42] S.M. Ross, *Introduction to Probability Models.* Academic Press, San Diego, California, 1993.

[43] N. Sakamoto, M. Ishiguro, and G. Kitagawa, *Information Statistics* (in Japanese). Kyoritsu-Shuppan, Tokyo, 1983.

[44] D. Satoh, "A discrete Gompertz equation and a software reliability growth model," *IEICE Transactions on Information and Systems*, Vol. E83-D, No. 7, pp. 1508–1513, July 2000.

[45] D. Satoh, "A discrete Bass model and its parameter estimation," *Journal of the Operations Research Society of Japan*, Vol. 44, No. 1, pp. 1–18, March 2001.

[46] D. Satoh and S. Yamada, "Discrete equations and software reliability growth models," *Proceedings of the* 12*th International Symposium on Software Reliability Engineering* (ISSRE'01), Hong Kong, China, November 28–December 1, 2001, pp. 176–184.

[47] D. Satoh and S. Yamada, "Parameter estimation of discrete logistic curve models for software reliability assessment," *Japan Journal of Industrial and Applied Mathematics*, Vol. 19, No. 1, pp. 39–54, February 2002.

[48] J.G. Shanthikumar, "A general software reliability model for performance prediction," *Microelectronics and Reliability*, Vol. 21, No. 5, pp. 671–682, 1981.

[49] Y. Tamura, M. Kimura, and S. Yamada, "Software reliability growth model for a distributed development environment: Stochastic differential equation approach and its numerical estimation,"(in Japanese) *Transactions of the Japan Society for Industrial and Applied Mathematics*, Vol. 11, No. 3, pp. 121–132, 2001.

[50] K.S. Trivedi, *Probability and Statistics with Reliability, Queueing and Computer Science.* (Second Ed.), John Wiley & Sons, New York, 2002.

[51] S. Yamada, *Software Reliability Assessment Technology* (in Japanese). HBJ Japan, Tokyo, 1989.

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

[53] S. Yamada, "Software reliability growth models incorporating imperfect debugging with introduced faults," *Electronics and Communications in Japan* (*Part 3*), Vol. 84, No. 4, pp. 33–41, 1998.

[54] S. Yamada, "Software reliability models," in *Stochastic Models in Reliability and Maintenance*, S. Osaki, Ed., Springer-Verlag, Berlin, 2002, pp. 253–280.

[55] S. Yamada and T. Fujiwara, *Software Reliability: Models, Tool, Management* (in Japanese). The Society of Project Management, Chiba, 2004.

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

[57] 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.

[58] S. Yamada and S. Osaki, "Software reliability growth modeling : Models and applications," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 12, pp. 1431–1437, 1985.

[59] S. Yamada and S. Osaki, "Cost-reliability optimal release policies for software systems," *IEEE Transactions on Reliability*, Vol. R-34, No. 5, pp. 422–424, December 1985.

[60] S. Yamada and S. Osaki, "Discrete software reliability growth models," *Journal of Applied Stochastic Models and Data Analysis*, Vol. 1, No. 1, pp. 65–77, 1985.

[61] S. Yamada and K. Sera, "Imperfect debugging models with two kinds of software hazard rate and their Bayesian formulation," *Electronics and Communications in Japan (Part 3)*, Vol. 84, No. 3, pp. 12–20, 2001.

[62] S. Yamada, T. Hangai, and S. Osaki, "A generalized delayed S-shaped software reliability growth model and evaluation of its goodness-of-fit," (in Japanese) *Transactions of IEICE*, Vol. J76-D-I, No. 11, pp. 613–620, November 1993.

[63] S. Yamada, T. Ichimori, and N. Nishiwaki, "Optimal allocation policies for testing-resource based on a software reliability growth model," *Mathematical and Computer Modelling*, Vol. 22, No. 10-12, pp. 295–301, 1995.

[64] S. Yamada, M. Kimura, and M. Takahashi, *Statistical Quality Control for TQM* (in Japanese). Corona Publishing, Tokyo, 1998.

[65] S. Yamada, A. Nishigaki, and M. Kimura, "A stochastic differential equation model for software reliability assessment and its goodness-of-fit," *International Journal of Reliability and Applications*, Vol. 4, No. 1, pp. 1–11, 2003.

[66] S. Yamada, M. Ohba, and S. Osaki, "S-shaped reliability growth modeling for software error detection," *IEEE Transactions on Reliability*, Vol. R-32, No. 5, pp. 475–478,484, 1983.

[67] S. Yamada, H. Ohtera, and H. Narihisa, "Software reliability growth models with testing-effort," *IEEE Transactions on Reliability*, Vol. R-35, No. 1, pp. 19–23, 1986.

[68] S. Yamada, H. Ohtera, and M. Ohba, "Testing-domain dependent software reliability models," *Computers & Mathematics with Applications*, Vol. 24, No. 1/2, pp. 679–86, July 1992.

[69] S. Yamada, K. Tokuno, and S. Osaki, "Imperfect debugging models with fault introduction rate for software reliability assessment," *International Journal of Systems Science*, Vol. 23, No. 12, pp. 2241–2252, 1991.

[70] S. Yamada, T. Yamane, and S. Osaki, "Software reliability growth models with error debugging rate,"(in Japanese) *Transactions of IPS Japan*, Vol. 27, No. 1, pp. 64–71, 1986.

[71] S. Yamada, M. Kimura, H. Tanaka, and S. Osaki, "Software reliability measurement and assessment with stochastic differential equations," *IEICE Transactions on Fundamentals of Electronics, and Computer Sciences*, Vol. E77-A, No. 1, pp. 109–116, 1994.

[72] P. Zeephongsekul, "Reliability growth of a software model under imperfect debugging and generation of errors," *Microelectronics and Reliability*, Vol. 36, No. 10, pp. 1475–1482, 1996.

Publication List of the Author

(Refereed Papers)

1. S. Yamada, S. Inoue, and D. Satoh, "Statistical data analysis modeling based on difference equations for software reliability assessment," (in Japanese) *Transactions of the Japan Society for Industrial and Applied Mathematics*, Vol. 12, No. 2, pp. 155–168, June 2002.

2. S. Inoue and S. Yamada, "NHPP modeling based on difference equations for software reliability assessment," *Proceedings of the Second Euro-Japanese Workshop on Stochastic Risk Modelling for Finance, Insurance, Production and Reliability*, Chamonix, France, September 18–20, 2002, pp. 145–154.

3. S. Inoue and S. Yamada, "A software reliability growth modeling based on infinite server queueing theory," *Proceedings of the Ninth ISSAT International Conference on Reliability and Quality in Design*, Honolulu, Hawaii, U.S.A., August 7–9, 2003, pp. 305–309.

4. S. Inoue and S. Yamada, "NHPP modeling based on discrete statistical data analysis models for software reliability assessment," *Proceedings of the International Workshop on Reliability and Its Applications*, Seoul, Korea, December 3–5, 2003, pp. 138–143.

5. S. Inoue and S. Yamada, "Stochastic differential equation modeling for testing-effort dependent software reliability assessment," *Proceedings of the Tenth ISSAT International Conference on Reliability and Quality in Design*, Las Vegas, Nevada, U.S.A., August 5–7, 2004, pp. 256–260.

6. T. Yamamoto, S. Inoue, and S. Yamada, "A software reliability growth model with testing-coverage maturity process," *Proceedings of the Tenth ISSAT International Conference on Reliability and Quality in Design*, Las Vegas, Nevada, U.S.A., August 5–7, 2004, pp. 299–303.

7. S. Inoue and S. Yamada, "Continuous-state software reliability growth modeling with testing-effort and its goodness-of-fit," in *Advanced Reliability Modeling* (Proceedings of the 2004 Asian International Workshop on Advanced Reliability Modeling [AIWARM 2004]), T. Dohi and W.Y. Yun, Eds., pp. 189–196, World Scientific, Singapore, 2004.

8. S. Inoue and S. Yamada, "A summary of discretized software reliability growth models," *Proceedings of the Seventh China-Japan International Conference on Industrial Management (ICIM '2004)*, Okayama, Japan, November 15–17, 2004, pp. 628–637.

9. S. Inoue and S. Yamada, "Testing-coverage dependent software reliability growth modeling," *International Journal of Reliability, Quality and Safety Engineering*, Vol. 11, No. 4, pp. 303–312, December 2004.

10. S. Inoue and S. Yamada, "Testing-coverage dependent software reliability growth modeling and its applications," *Proceedings of the 2004 International Computer Symposium (ICS 2004)*, Taipei, Taiwan, R.O.C., December 15–17, 2004, pp. 938–943.

11. S. Inoue and S. Yamada, "On accuracy improvement of software reliability assessment based on discrete models," (in Japanese) *Proceedings of the Thirty-fifth Reliability and Maintainability Symposium*, Tokyo, Japan, June 8–9, 2005, pp. 191–196.

12. S. Inoue and S. Yamada, "Discrete software reliability models based on software failure-occurrence times distribution," *Proceedings of the Eleventh ISSAT International Conference on Reliability and Quality in Design*, St. Louis, Missouri, U.S.A., August 4–6, 2005, pp. 87–91.

13. T. Fujiwara, S. Inoue, and S. Yamada, "A software reliability growth model with module composition," *Proceedings of the Eleventh ISSAT International Conference on Reliability and Quality in Design*, St. Louis, Missouri, U.S.A., August 4–6, 2005, pp. 266–270.

14. S. Inoue and S. Yamada, "A generalized discrete software reliability model and its application to optimal release problems," *Proceedings of the 2005 International Workshop on Recent Advances in Stochastic Operations Research (2005 RASOR)*, Canmore, Alberta, Canada, August 25–26, 2005, pp. 88–95.

15. S. Inoue and S. Yamada, "An extended delayed S-shaped software reliability growth model based on infinite server queueing theory," in *Reliability Modeling, Analysis and Optimization*, H. Pham, Ed., to be published, World Scientific, Singapore, 2005.

16. S. Inoue and S. Yamada, "Discrete software reliability assessment with discretized NHPP models," *Computers & Mathematics with Applications: An International Journal*, to be published in 2006.

17. S. Inoue and S. Yamada, "Generalized discrete software reliability modeling with effect of program size," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, to be published in 2006.

(Technical Reports)

1. S. Inoue and S. Yamada, "Discretization of software reliability growth models with applications to optimal software release problems," (in Jananese) *The Institute of Statistical Mathematics Cooperative Research Report 161 — Optimization : Modeling and Algorithms 16* —, pp. 101–109, February 2003.

2. S. Inoue, S. Yamada, and T. Ichimori, "Discretized software reliability growth models and its application to optimal software release problems," (in Jananese) *The Institute of Statistical Mathematics Cooperative Research Report 168 — Optimization : Modeling and Algorithms 17* —, pp. 75–82, February 2004.

3. S. Inoue, A. Nishigaki, S. Yamada, and M. Kimura, "A stochastic differential equation modeling for software reliability assessment and its application to an optimal release problem," (in Jananese) *The Research Institute for Mathematical Sciences of Kyoto University Research Report 1373, — Mathematical Programming Concerning Decision Makings and Uncertainties* —, pp. 80–88, May 2004.

4. S. Inoue and S. Yamada, "A optimal software release problem under imperfect debugging environment," (in Jananese) *The Institute of Statistical Mathematics Cooperative Research Report 178 — Optimization : Modeling and Algorithms 18* —, pp. 376–381, March 2005.

5. S. Inoue and S. Yamada, "Discrete software reliability growth models based on order-statistics and its application to optimal release problems," (in Jananese) *The Institute of Statistical Mathematics Cooperative Research Report — Optimization : Modeling and Algorithms 19* —, to be pulished in 2006.

(National Conference and Symposium Presentations)

1. S. Inoue, S. Yamada, and D. Satoh, "Software reliability growth models based on difference quations," (in Jananese) *Proceedings of the 2001 Fall National Conference of ORSJ*, Okayama, Japan, September 2001, pp. 66–67.

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

3. S. Inoue, S. Yamada, and K. Tokuno, "A software reliability growth modeling considering with time distribution of fault isolation process," (in Jananese) *Proceedings of the RENTAI 2002*, Matsue, Japan, October 2002, p. 505.

4. S. Inoue and S. Yamada, "An infinite server queueing model considering time distribution of fault isolation process for software reliability assessment," (in Jananese) *IEICE Technical Report [Reliability]*, Vol. 102, No. 454, pp. 119–124, November 2002.

5. S. Inoue, Y. Tamura, and S. Yamada, "A discrete software reliability growth model for distributed development environment," (in Jananese) *Proceedings of the 4th IEEE Hiroshima Student Symposium*, Yamaguchi, Japan, December 2002, pp. 254–255.

6. S. Inoue and S. Yamada, "A software reliability growth modeling based on infinite server queueing theory," (in Jananese) *Proceedings of the 2003 Spring National Conference of ORSJ*, Yokohama, Japan, March 2003, pp. 30–31.

7. S. Inoue and S. Yamada, "Stochastic software reliability growth modeling based on discrete Statistical data analysis models," (in Jananese) *IEICE Technical Report [Reliability]*, Vol. 103, No. 77, pp. 19–24, May 2003.

8. S. Inoue, T. Yamamoto, and S. Yamada, "On relationship between testing-coverage and software reliability," (in Jananese) *Proceedings of the 2003 Fall National Conference of ORSJ*, Fukuoka, Japan, September 2003, pp. 54–55.

9. S. Inoue and S. Yamada, "Discretized stochastic reliability growth modeling for software reliability assessment, " *Proceedings of the 5th IEEE Hiroshima Student Symposium*, Hiroshima, Japan, December 2003, pp. 148–149.

10. T. Yamamoto, S. Inoue, and S. Yamada, "Software reliability growth modeling considering with testing-coverage," (in Jananese) *Proceedings of the 5th IEEE Hiroshima Student Symposium*, Hiroshima, Japan, December 2003, pp. 150–151.

11. S. Inoue and S. Yamada, "Software reliability growth modeling with a detectability of faults based on testing-coverage," (in Jananese) *Proceedings of the 2004 Spring National Conference of ORSJ*, Tokyo, Japan, March 2004, pp. 270–271.

12. S. Inoue and S. Yamada, "Testing-effort dependent software reliability growth modeling based on stochastic differential equations," (in Jananese) *IEICE Technical Report* [*Reliability*], Vol. 104, No. 72, pp. 41–46, May 2004.

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

14. S. Inoue and S. Yamada, "A optimal software release policy with a debugging rate," (in Jananese) *Proceedings of the 2004 Fall National Conference of ORSJ*, Sendai, Japan, September 2004, pp. 184–185.

15. S. Inoue, T. Yamamoto, and S. Yamada, "Testing-coverage dependent software reliability growth modeling with designing-skill of test-cases," (in Jananese) *Proceedings of the RENTAI 2004*, Ube, Japan, October 2004, p. 428.

16. S. Inoue and S. Yamada, "A software reliability growth model with testing-effort expenditures based on a lognormal process," *Proceedings of the 6th IEEE Hiroshima Student Symposium*, Matsue, Japan, December 2004, pp. 258–259.

17. S. Inoue and S. Yamada, "Consideration on discrete software reliability growth modeling based on order-statistics," (in Jananese) *Proceedings of the 2005 Spring National Conference of ORSJ*, Tokyo, Japan, March 2005, pp. 138–139.

18. S. Inoue and S. Yamada, "Software reliability modeling with 2-types of imperfect debugging activities," (in Jananese) *IEICE Technical Report* [*Reliability*], Vol. 105, No. 101, pp. 1–6, May 2005.

19. S. Inoue and S. Yamada, "A generalized discrete binomial process model for software reliability assessment and its application to optimal release problems," *IEICE Technical Report* [*Reliability*], Vol. 105, No. 187, pp. 35–40, July 2005.

20. S. Inoue and S. Yamada, "On software reliability growth modeling with imperfect debugging activities," (in Jananese) *Proceedings of the 2005 Fall National Conference of ORSJ*, Kobe, Japan, September 2005, pp. 86–87.

21. S. Inoue and S. Yamada, "On software reliability assessment with introduced faults," *Proceedings of the 7th IEEE Hiroshima Student Symposium*, Okayama, Japan, November 2005, pp. 307–308.

Received Awards List of the Author

1. Distinguished Service Award (January 30, 2002)

   The 3rd IEEE Hiroshima Section Student Symposium (HISS 2001), Hiroshima, Japan, December 14–15, 2001.

2. Contribution Award (December 27, 2002)

   The 4th IEEE Hiroshima Section Student Symposium (HISS 2002), Yamaguchi, Japan, December 5–6, 2002.

3. Excellent Research Award (December 6, 2002)

   The 4th IEEE Hiroshima Section Student Symposium (HISS 2002), Yamaguchi, Japan, December 5–6, 2002.

   —Title—

   S. Inoue, Y. Tamura, and S. Yamada, "A discrete software reliability growth model for distributed development environment," (in Japanese) *Proceedings of the 4th IEEE Hiroshima Section Student Symposium* (*HISS 2002*), Yamaguchi, Japan, December 5–6, 2002, pp. 254–255.

4. Best Paper Award (December 16, 2004)

   The 2004 International Computer Symposium (ICS 2004), Taipei, Taiwan, R.O.C., December 15–17, 2004.

   —Title—

   S. Inoue and S. Yamada, "Testing-coverage dependent software reliability growth modeling and its applications," *Proceedings of the 2004 International Computer*

Symposium (*ICS 2004*), Taipei, Taiwan, R.O.C., December 15–17, 2004, pp. 938–943.

5. Outstanding Paper Award (May 13, 2005)

   The Information Processing Society of Japan (IPSJ), Chugoku Branch

   —Title—
   S. Inoue, T. Yamamoto, and S. Yamada, "Testing-coverage dependent software reliability growth modeling with designing-skill of test-cases," (in Japanese) *Proceedings of the RENTAI 2004*, Ube, Japan, October 16, 2004, p. 428.

6. Outstanding Paper Award (May 27, 2005)

   The Institute of Electronics, Information and Communication Engineers (IE-ICE), Chugoku Branch

   —Title—
   S. Inoue, T. Yamamoto, and S. Yamada, "Testing-coverage dependent software reliability growth modeling with designing-skill of test-cases," (in Japanese) *Proceedings of the RENTAI 2004*, Ube, Japan, October 16, 2004, p. 428.

Research Grants List of the Author

1. The International Conference Travel Grant from the Telecommunications Advancement Foundation (TAF). (2003)
   —Conference Name—
   Ninth ISSAT International Conference on Reliability and Quality in Design, Honolulu, Hawaii, U.S.A., August 7–9, 2003.

2. The Sasakawa Scientific Research Grant from the Japan Science Society.
   (April 2004–February 2005) [Grant No. 16-064]
   —Subject—
   "Studies on Accuracy Improvement Techniques for Software Reliability Assessment and Its Applications"

3. The Excellent Student Research Grant from the Doctoral Program of Tottori University.
   (October 2005–March 2006)
   —Subject—
   "Studies on Analysis of Fault-Detection Events Occurrence Mechanisms and Their Modeling for Software Reliability Assessment "

END