A Study
on
Markovian Software Reliability Modeling
for
Availability and Safety Assessment

January 1999

Koichi Tokuno

A Study
on
Markovian Software Reliability Modeling
for
Availability and Safety Assessment

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

Koichi Tokuno
Doctoral Program of the Graduate School of Engineering,
Tottori University, Tottori, Japan

January 1999

ABSTRACT

This doctoral dissertation deals with software quality measurement and assessment in dynamic environment such as the testing phase of the software development process and the operational maintenance phase. In particular, we propose several stochastic models for software reliability, availability, and safety assessment. These models are developed by employing Markov processes. The dissertation consists of the three main parts.

In the first part software reliability modeling is discussed. In particular, imperfect debugging environment is the central argument. Chapter 2 gives a software reliability model with imperfect debugging considering the uncertainty of fault removal. Based on this model, optimal software release policies with cost factors and reliability requirement are also investigated as application to practical use. Chapter 3 proposes an imperfect debugging model considering the existence of regenerative faults as well and presents application examples of the model to an actual testing data set. These two models provide several stochastic quantities for software reliability measurement.

In the second part software availability is the main issue. Chapter 4 discusses software availability modeling by treating operational and restoration time-intervals of a system as random variables depending on the cumulative number of corrected faults. Estimation of unknown parameters is also provided in this chapter. Chapters 5 and 6 give customer-oriented software availability models. The model considering two types of software failures in user operation is discussed in Chapter 5 and two types of restoration scenarios in Chapter 6. Chapter 7 proposes a software availability model with degenerated performance. This model can provide a new performance measure considering reliability and computation simultaneously. Chapter 8 deals with availability modeling for a computer-based system, which consists of one hardware and one software subsystem.

The third part focuses on software safety measurement and assessment. Chapter 9 gives a safety assessment model describing the relationship between software failure-occurrences and software safety. Based on the software reliability/availability modeling discussed in the above two parts, Chapter 10 deals with software safety modeling which takes account of the situation where a system engenders unsafe states in operation.

In the final chapter we summarize the results obtained in the dissertation and future research works on software reliability modeling.

ACKNOWLEDGMENTS

The author would like to express his gratitude to Professor Shigeru Yamada, the supervisor of the author's study and the chairman of this dissertation reviewing committee, for his introduction to research in software reliability, valuable advice, continuous support, and warm guidance.

The author is indebted to Professor Hajime Kawai, Professor Satoru Ikehara, Professor Yutaka Fukui, and Professor Kazuhiro Sugata, the members of the dissertation reviewing committee, for reading the manuscript and making helpful comments.

The author wishes to particularly thank Professor Shunji Osaki of Hiroshima University for invariable encouragement and support.

The author has also received priceless cooperation and suggestions from many people for the achievement of this work. Special thanks are due to Professor Naoto Kaio of Hiroshima Shudo University, Associate Professor Satoshi Fukumoto of Aichi Institute of Technology, Associate Professor Mitsuhiro Kimura of Tottori University, and Associate Professor Tadashi Dohi of Hiroshima University.

The author would also like to acknowledge the kind hospitality and encouragement of the past and present members of Professor Yamada's laboratory and the staff of Department of Social Systems Engineering of Tottori University.

Finally, the author wants to thank his late parents Kaoru and Mitsuko, to whom this dissertation is dedicated.

Contents

List of Figures

Part I SOFTWARE RELIABILITY MODELING

Part II SOFTWARE AVAILABILITY MODELING

Part III SOFTWARE SAFETY MODELING

List of Tables

Part I SOFTWARE RELIABILITY MODELING

Part II SOFTWARE AVAILABILITY MODELING

Chapter 1

Introduction

1.1 Software Reliability Engineering

At present, enormous software systems have been developed as computer systems have been utilized in various fields. The ability or utility of existing computer systems are greatly influenced by performance/quality of their software systems. Therefore, once system failures due to defects or faults latent in software systems come to the surface, the systems are entirely useless and many people sustain great damage. Moreover, they may cause serious accidents affecting people's lives [24]. Under the background like this, the software production technologies for the purpose of producing quality software systems efficiently, systematically, and economically have been developed and researched energetically. Among these, software management technologies have recently risen in importance [14, 63].

Comprehensive use of technologies and methodologies in software engineering is needed for improving software quality/reliability. Figure 1.1 summarizes the representative software reliability technologies [57]. *Software Reliability Engineering* (SRE) is a subdiscipline of software engineering and the quantitative study of how well the operational behavior of software-based systems meets customer requirements [27, 32]. Figure 1.2 shows SRE activities in the respective software life-cycle phases [4, 57].

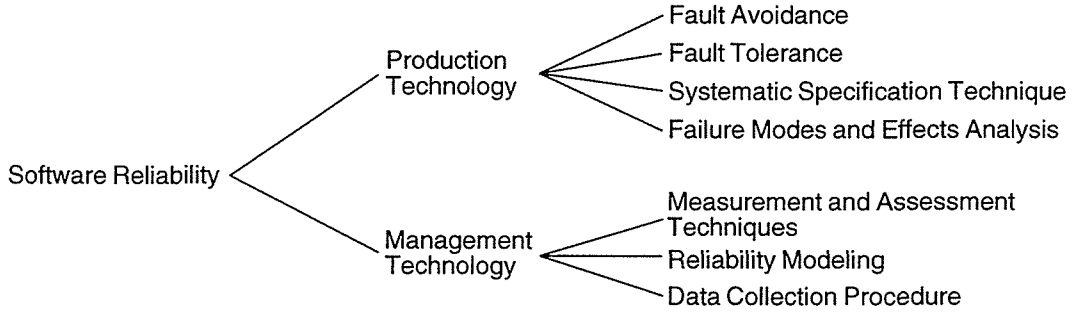

Fig. 1.1. Software reliability technologies.

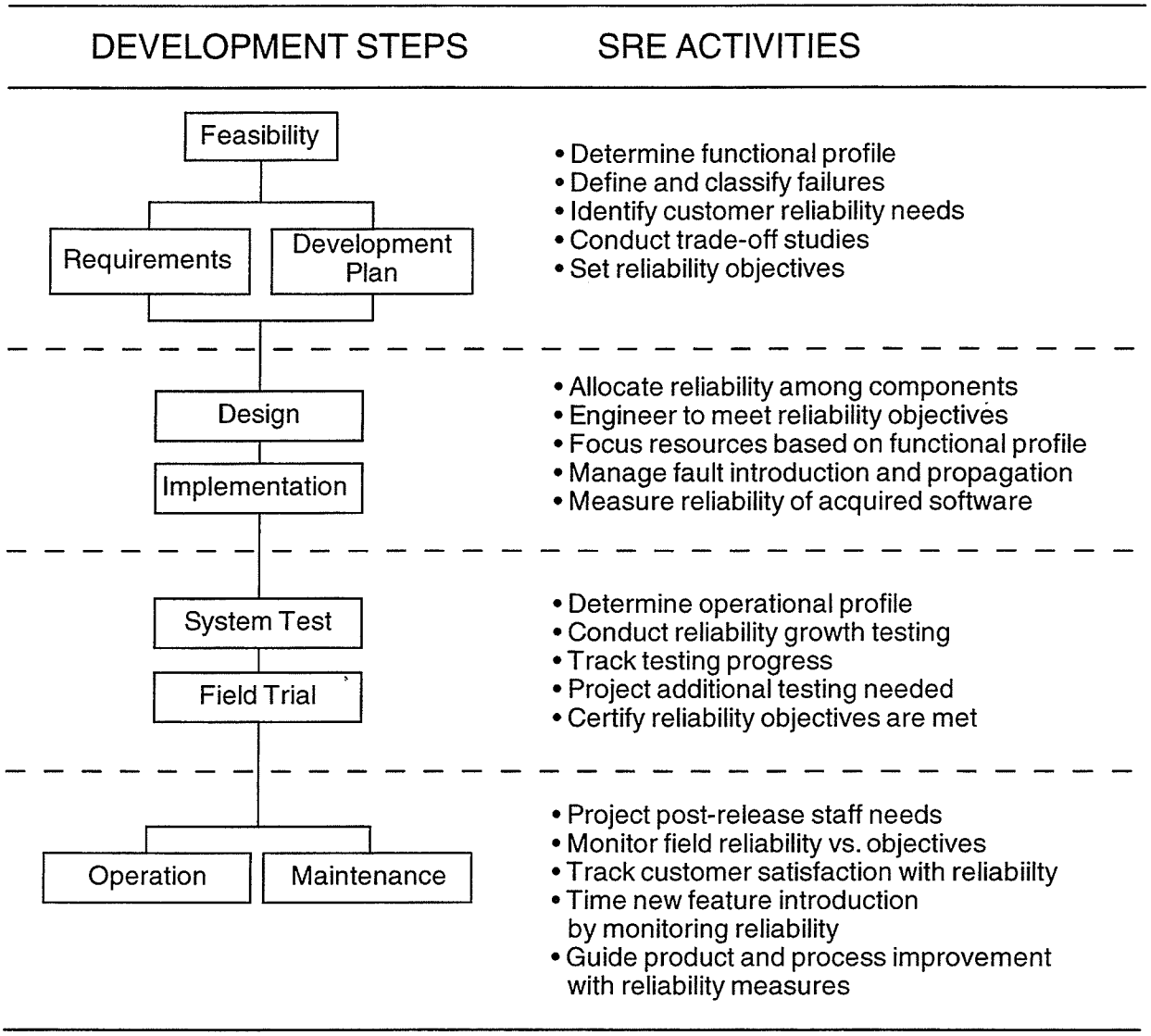

Fig. 1.2. SRE activities over the software product life-cycle phases.

There exist two fundamental ways to improve software quality/reliability: *fault avoidance* and *fault tolerance*. Fault avoidance is the more traditional approach and attempts to ensure a fault-free system by equipping methodologies for fault prevention and fault removal in the software development process. Fault tolerance techniques are based on the premise that a system will contain residual design faults, no matter how carefully designed and validated, and attempt to limit performance degradation and failure occurrences due to such faults to a minimum. The N-version programming [2] and the recovery block

scheme [45] are the representative software fault-tolerance approaches. The software-fault-tolerant architecture consists of plural *variants*, which are different systems produced from a common specification, and a *decider*, which monitors the results of variant execution [20].

We define the following technical terms in SRE [25, 35, 57]:

- *Software failure*

  A software failure occurs when a software system ceases to deliver the expected outputs and does not function in accordance with a software specification.

- *Software fault*

  A defective statement or a set of defective statements in a software program which may cause a software failure. A software fault is also referred as a *software bug*.

- *Software error*

  A human intellectual action that results in a software system containing a software fault. A software error introduces a software fault into a system.

In this dissertation the terms *software fault* and *software error* are used as the same meaning since a software fault is a manifestation of a software error. Hereafter a software fault is referred as a fault.

The definitions of dependability and the software attributes of dependability handled in this dissertation are given as follows:

Dependability is defined as the trustworthiness of a system and is a comprehensive term to describe the attributes such as reliability, availability, safety, confidentiality, integrity, and maintainability [25].

- **Software reliability** is the attribute that a software system will perform without causing a software failure over a given time period, under specified operational environment.

- **Software availability** is the attribute that a software system will perform and be available at a given time point, under specified operational environment.

- **Software safety** is the attribute that a software system will not induce any hazards whether or not the system is functioning in accordance with the specification, where

a *hazard* is defined as a state or a set of conditions of a software system that, together with other conditions in the environment of the system, will lead inevitably to an accident [23].

The main characteristics of dependability are summarized in Fig. 1.3 [25].

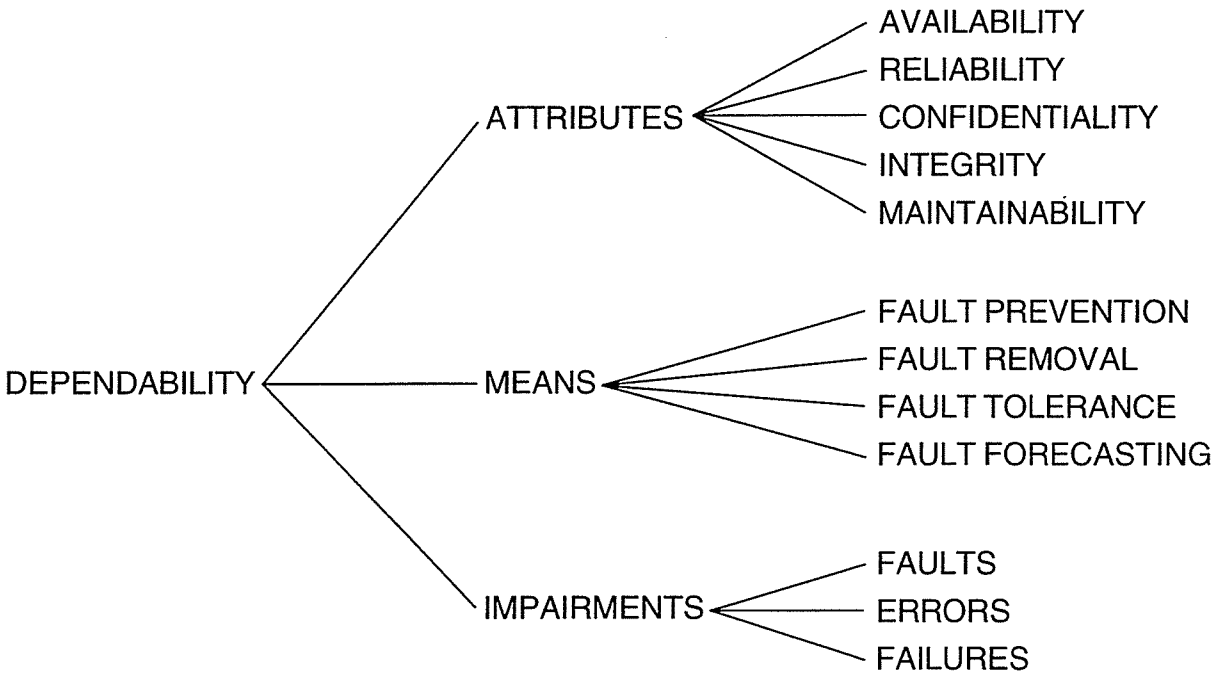

Fig. 1.3. The dependability tree.

One of the above software reliability technologies is a quantitative technique for software quality/reliability measurement, and many mathematical models for the purpose of measuring and assessing software reliability reasonably, which is a so-call *software reliability model*, have been proposed and applied to practical use because it is considered that software reliability is the main quality characteristic in a software product. A mathematical software reliability model is often called a *software reliability growth model* which describes a software fault-detection or a software failure-occurrence phenomenon during the testing phase of the software development process and the operation phase [1, 26, 35, 44, 56, 57].

The existing software reliability models are classified from various view points by several researchers such as Mellor [28], Musa et al. [35], Romamoorthy & Bastani [44], Shanthikumar [49], Xie [54], and Yamada [56]. Yamada [56, 57] classifies the software reliability growth models into the following three categories:

(1) *the failure-occurrence time model*: a stochastic model taking notice of the time-interval between software failure-occurrences.

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

(3) *the availability model*: a stochastic model describing the time-dependent behavior of a software system which alternates between up and down state.

The above models are utilized for measuring and assessing the degree of achievement of software reliability, deciding the time to software release for operational use, and estimating the maintenance cost for faults undetected during the testing phase.

1.2 Organization of Dissertation

This dissertation studies software reliability modeling for availability and safety assessment by employing Markov processes. The dissertation is divided into the three main parts: Part I on software reliability modeling, Part II on software availability modeling, and Part III on software safety modeling. The three parts are complemented by this introduction and a conclusion in Chapter 11.

- *Part I: Software Reliability Modeling*

  In the first part software reliability modeling is discussed. In particular, imperfect debugging environment is the central argument. Chapter 2 gives a software reliability model with imperfect debugging considering the uncertainty of fault removal. Based on this model, optimal software release policies with cost factors and reliability requirement are also investigated as application to practical use. Chapter 3 proposes an imperfect debugging model considering the existence of regenerative faults as well and presents application examples of the model to an actual testing data set.

- *Part II: Software Availability Modeling*

  Chapter 4 discusses software availability modeling by treating operational and restoration time-intervals of a system as random variables depending on the cumulative number of corrected faults. Estimation of unknown parameters is also provided in

this chapter. Chapters 5 and 6 give customer-oriented software availability models. The model considering two types of software failures in user operation is discussed in Chapter 5 and two types of restoration scenarios in Chapter 6. Chapter 7 proposes a software availability model with degenerated performance. This model can provide a new performance measure considering reliability and computation simultaneously. Chapter 8 deals with availability modeling for a computer-based system, which consists of one hardware and one software subsystem.

- *Part III: Software Safety Modeling*

  The third part focuses on software safety measurement and assessment. Chapter 9 gives a safety assessment model describing the relationship between software failure-occurrences and software safety. Based on the software reliability/availability modeling discussed in the above two parts, Chapter 10 deals with software safety modeling which takes account of the situation where a system engenders unsafe states in operation.

Chapter 11 summarizes the results obtained in the dissertation. We conclude the dissertation with prospects for the future research works on software reliability modeling.

Part I

SOFTWARE RELIABILITY MODELING

Chapter 2

Software Reliability Modeling with Imperfect Debugging

2.1 Introduction

As computer systems grow in size and complexity, quality software systems are more required for a higher degree of system reliability. Therefore, quality control approaches are indispensable to developing quality software systems efficiently. Software reliability, i.e. one of representative software quality characteristics, is called one of the taken-for-granted quality or must-be quality for a software system. Accordingly, it is great important for the software development managers to establish a reliability objective and its reliability assurance procedures precisely [14, 59, 63]. Generally, a mathematical model is very useful for evaluating and assessing the degree of achievement of software reliability. A mathematical software reliability model is called a software reliability growth model which describes a software fault-detection or a software failure-occurrence phenomenon during the testing phase in the software development process and the operation phase [55, 56]. A software failure is defined as an unacceptable departure from program operation caused by a fault remaining in the software system. Using the models, we can estimate several quantitative measures such as the initial fault content, the software reliability, and the mean time between software failures.

Most software reliability growth models proposed so far are based on the assumption of perfect debugging that all faults detected during the testing and the operation phase are corrected and removed perfectly. However, debugging actions in actual testing and operation environments are not always performed perfectly. For example, typographical errors invalidate fault-correction activities or fault removal is not carried out precisely due

to wrong analysis for the obtained testing results [50]. That is, they are in imperfect debugging environment. Therefore, the faults are not always corrected and removed perfectly when they are detected. Then, it is interesting to develop a software reliability growth model considering imperfect debugging environment (cf. [38, 48, 64]). Such an imperfect debugging model is expected to estimate reliability assessment measures more accurately.

In this chapter, we discuss a software reliability growth model with imperfect debugging that faults are not corrected/removed certainly when they are detected. Defining a random variable representing the number of faults corrected by a given time point, this model is formulated by a Markov process [10, 42]. We derive various interesting quantities for software reliability measurement. Furthermore, based on this model, we discuss optimal software release problems by introducing software cost and reliability criteria [7, 17, 40]. Finally, we show numerical illustrations of software reliability measurement and optimal software release problems.

2.2 Model Description

For imperfect debugging environment, the software reliability model developed in this chapter is based on the following assumptions:

A1. Each fault which causes a software failure, when detected, is corrected perfectly with probability p $(0 < p \leq 1)$, while it is not corrected with probability $q(= 1 - p)$ [48]. We call p the perfect debugging rate.

A2. The hazard rate is constant between software failures caused by a fault in the software system, and geometrically decreases whenever each detected fault is corrected [31].

A3. The probability that two or more software failures occur simultaneously is negligible.

A4. No new faults are introduced during the debugging. At most one fault is removed when it is corrected and the correction time is not considered.

We consider a stochastic process $(\boldsymbol{Y}, \boldsymbol{T})$ where fault count vector $\boldsymbol{Y} = \{Y(l);\ l = 1, 2, \ldots\}$ and time series vector $\boldsymbol{T} = \{T_l;\ l = 1, 2, \ldots\}$ [42, 47]. Let $i = 0, 1, 2, \ldots$ be the state space, where i represents the cumulative number of corrected faults. Then, the events

$\{Y(l) = i\}$ means that i faults have been corrected at the l-th software failure-occurrence and T_l represents the l-th software failure-occurrence time-point, where $Y(0) = 0$ and $T_0 = 0$. A sample function of $(\boldsymbol{Y}, \boldsymbol{T})$ is shown in Fig. 2.1. For example, Fig. 2.1 shows that a fault is detected at T_3 but the fault correction fails (i.e. the fault is imperfectly debugged). Furthermore, let $\{X(t),\ t \geq 0\}$ be a random variable representing the cumulative number of faults corrected up to the testing time t. Then, $\{X(t),\ t \geq 0\}$ forms a Markov process [10]. That is, from assumption A1, when i faults have been corrected by arbitrary testing time t, after the next software failure occurs,

$$X(t) = \begin{cases} i & \text{(with probability } q\text{)} \\ i+1 & \text{(with probability } p\text{),} \end{cases} \tag{2.1}$$

(see Fig. 2.2). Furthermore, from assumption A2, when i faults have been corrected, the hazard rate for the next software failure-occurrence is given by

$$z_i(t) = Dk^i \qquad (i = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{2.2}$$

where D and k are the initial hazard rate and the decreasing ratio, respectively. A sample function of $z_i(t)$ is shown in Fig. 2.3. We see that the hazard rate $z_i(t)$ decreases when the correction of a detected fault succeeds, but when failing in the fault correction it does not decrease. Equation (2.2) describes the software failure-occurrence phenomenon where a software system has high frequency of software failure-occurrence during the early stage of the testing or the operation phase and it gradually decreases after then [35]. This model has high applicability in practical software reliability modeling. For example, Gaudoin et al. [8] and Musa & Okumoto [33, 34] have discussed software reliability modeling based on this model. Early software reliability models such as those of Goel & Okumoto [10] and Jelinski & Moranda [13] often assume that the hazard rate decreases by a constant amount with perfect debugging. Then, the distribution function for the next software failure-occurrence time is given by

$$F_i(t) = 1 - e^{-Dk^i t}. \tag{2.3}$$

Let $Q_{i,j}(\tau)$ denote the one-step transition probability that after making a transition into state i, the process $\{X(t),\ t \geq 0\}$ makes a transition into state j by time t. Then,

$Q_{i,j}(\tau)$ representing the probability that if i faults have been corrected at time zero, j faults are corrected by time τ after the next failure occurs is given by

$$Q_{i,j}(\tau) = P_{ij}(1 - e^{-Dk^i t}), \tag{2.4}$$

where P_{ij} are the transition probabilities from state i to state j and given by

$$P_{ij} = \begin{cases} q & (j = i) \\ p & (j = i+1) \\ 0 & (\text{otherwise}) \end{cases} \quad (i,\ j = 0,\ 1,\ 2,\ \ldots). \tag{2.5}$$

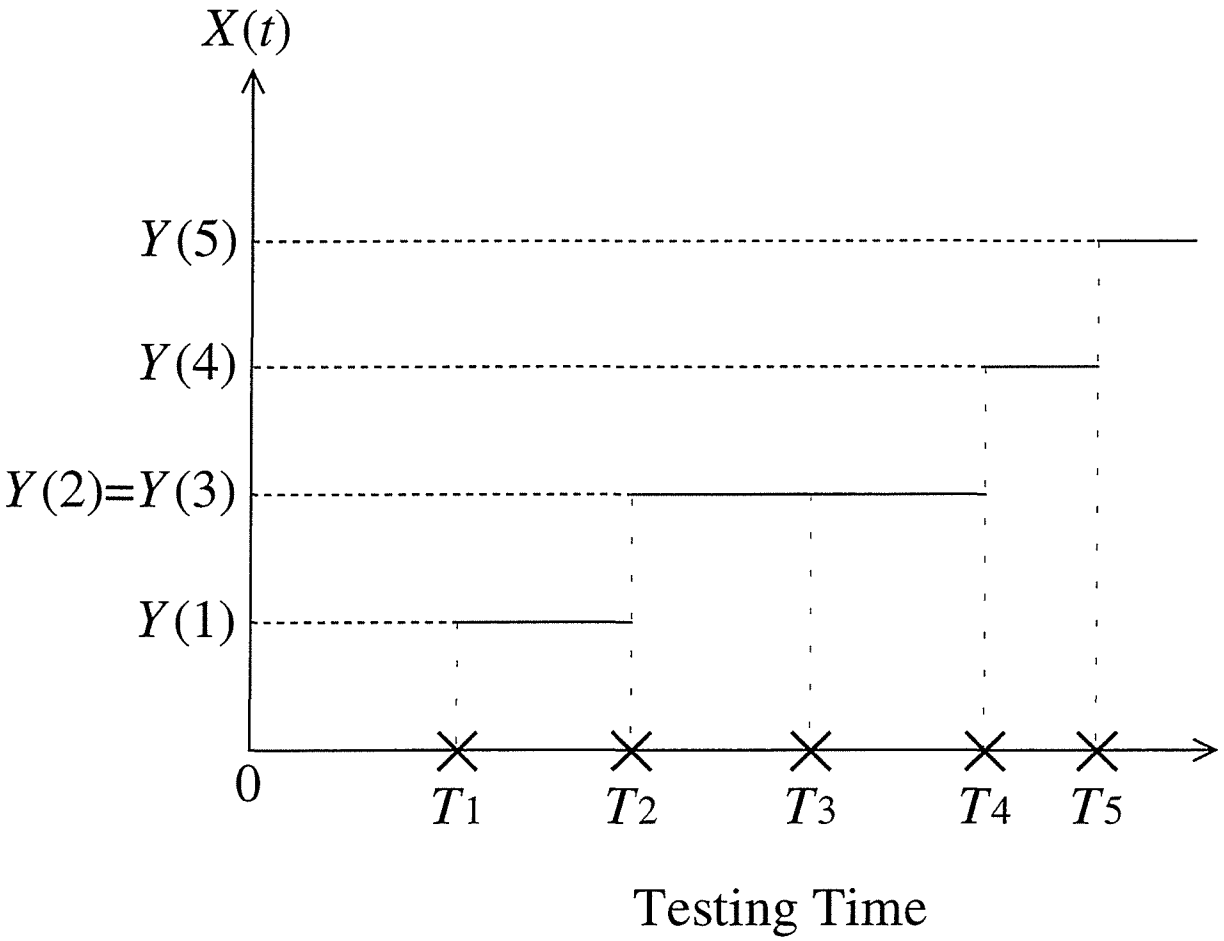

Fig. 2.1. A sample function of $(\boldsymbol{Y}, \boldsymbol{T})$.

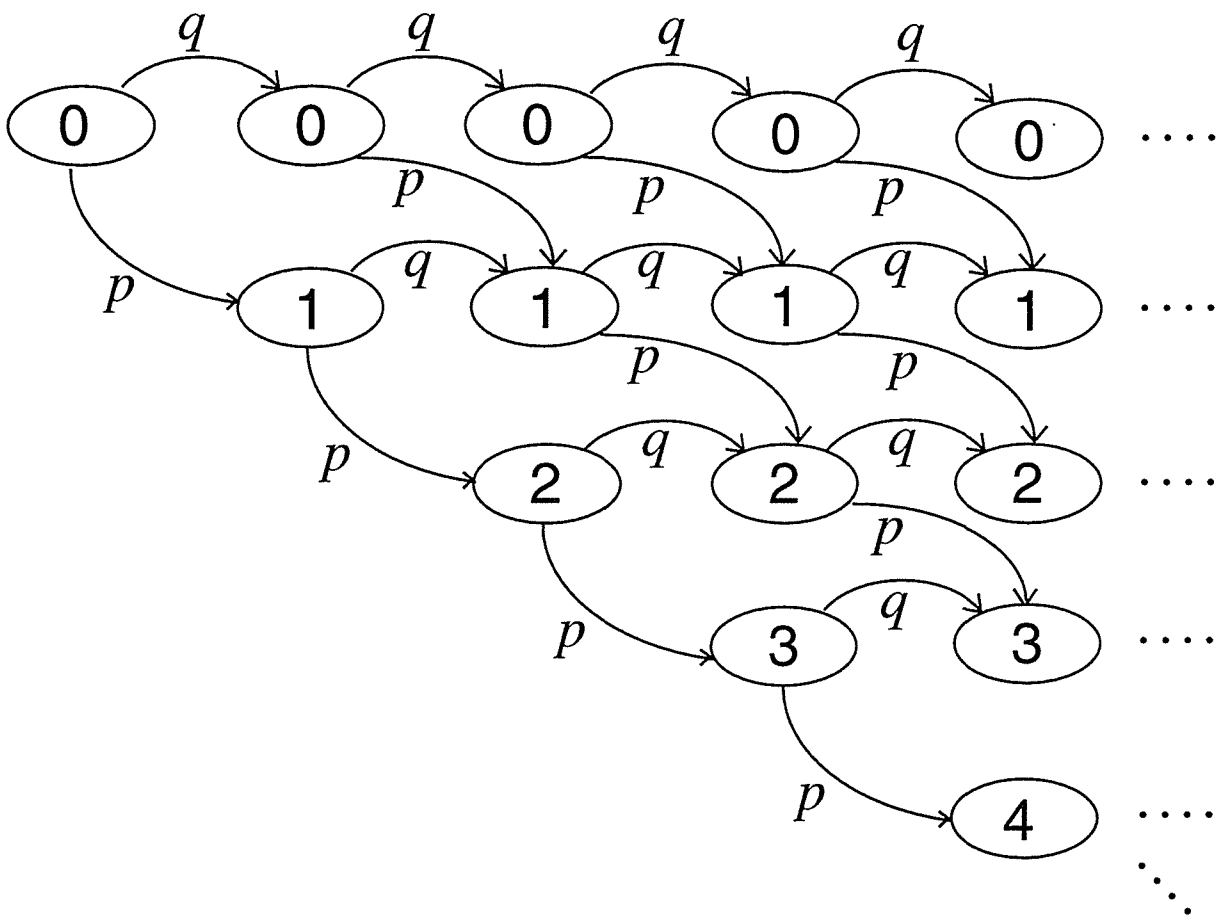

Fig. 2.2. A diagrammatic representation of transitions between states of $X(t)$ for software reliability modeling with imperfect debugging.

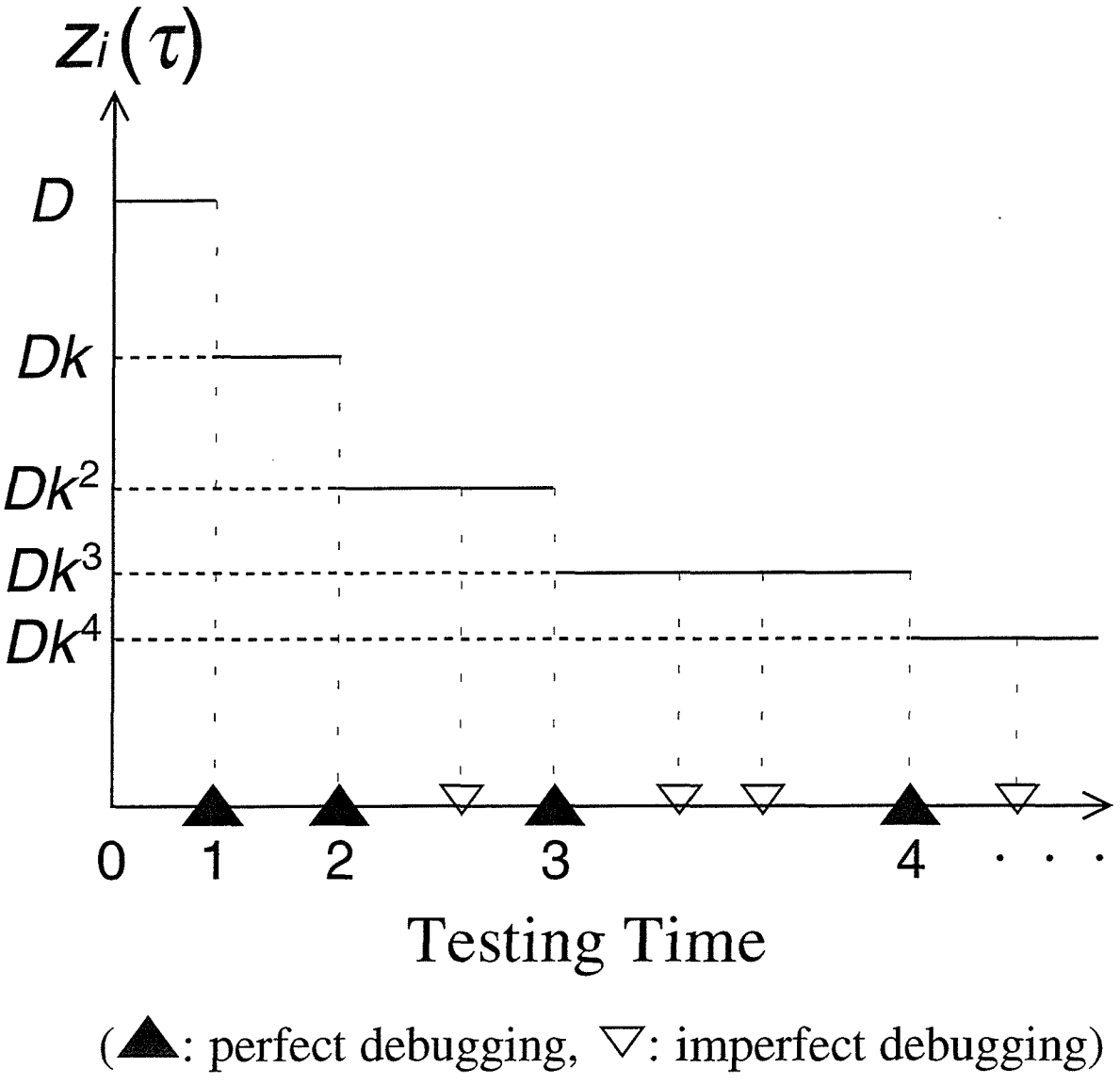

Fig. 2.3. A sample realization of hazard rate $z_i(t)$ for software reliability modeling with imperfect debugging.

2.3 Derivation of Reliability Measures

2.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Suppose that i faults have been corrected at some testing time. Let $G_{i,n}(t)$ denote a distribution function of the first passage time from state i to state n. In other words, $G_{i,n}(t)$ is the probability that n faults are corrected in the time interval $(0,\ t]$ on the condition that i faults have been already corrected at time zero. From (2.4) the probability of making a transition from state 0 to state 1 in the small time interval $(t,\ t+dt]$ is $dQ_{0,1}(t)$. Then, since the process $\{X(t),\ t \geq 0\}$ restarts on the condition that one fault has been corrected at time u (i.e. the fault has been perfectly debugged), the distribution function of the first passage time from state 0 to state n is given by

$$\int_0^t G_{1,n}(t-u)dQ_{0,1}(u) = Q_{0,1} * G_{1,n}(t), \tag{2.6}$$

where $*$ denotes a Stieltjes convolution. Similarly, for the case of imperfect debugging at the first software failure-occurrence, the distribution function of the first passage from state 0 to state n is given by

$$\int_0^t G_{0,n}(t-u)dQ_{0,0}(u) = Q_{0,0} * G_{0,n}(t). \tag{2.7}$$

Since the events described in (2.6) and (2.7) are mutually disjoint, we get the following renewal equation:

$$G_{0,n}(t) = Q_{0,1} * G_{1,n}(t) + Q_{0,0} * G_{0,n}(t). \tag{2.8}$$

In general, we have

$$G_{i,n}(t) = Q_{i,i+1} * G_{i+1,n}(t) + Q_{i,i} * G_{i,n}(t) \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{2.9}$$

where $G_{n,n}(t) = 1$ $(n = 1,\ 2,\ \ldots)$.

We use Laplace-Stieltjes (L-S) transforms [42] to solve (2.9), where the L-S transform of $G_{i,n}(t)$ is defined as

$$\tilde{G}_{i,n}(s) \equiv \int_0^\infty e^{-st} dG_{i,n}(t). \tag{2.10}$$

From (2.9) we get

$$\tilde{G}_{i,n}(s) = \tilde{Q}_{i,i+1}(s)\tilde{G}_{i+1,n}(s) + \tilde{Q}_{i,i}(s)\tilde{G}_{i,n}(s) \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1). \tag{2.11}$$

From (2.4) the L-S transforms of $Q_{i,i+1}(t)$ and $Q_{i,i}(t)$ are respectively given as

$$\tilde{Q}_{i,i+1}(s) = \frac{pDk^i}{s + Dk^i}, \tag{2.12}$$

$$\tilde{Q}_{i,i}(s) = \frac{qDk^i}{s + Dk^i}. \tag{2.13}$$

Substituting (2.12) and (2.13) into (2.11) yields

$$\tilde{G}_{i,n}(s) = \frac{pDk^i}{s + pDk^i}\tilde{G}_{i+1,n}(s) \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1). \tag{2.14}$$

Solving (2.14) recursively, we obtain the L-S transform of $G_{0,n}(t)$ as

$$\begin{aligned}\tilde{G}_{0,n}(s) &= \prod_{i=0}^{n-1} \frac{pDk^i}{s + pDk^i} \\ &= \sum_{i=0}^{n-1} A_i^n \frac{pDk^i}{s + pDk^i},\end{aligned} \tag{2.15}$$

where

$$\left.\begin{aligned} &A_0^1 \equiv 1 \\ &A_i^n = \frac{k^{\frac{1}{2}n(n-1)-i}}{\prod_{\substack{j=0 \\ j\neq i}}^{n-1}(k^j - k^i)} \quad (n = 2,\ 3,\ \ldots,\ i = 0,\ 1,\ 2,\ \ldots,\ n-1) \end{aligned}\right\}. \tag{2.16}$$

By inverting (2.15) and rewriting $G_{0,n}(t)$ as $G_n(t)$, we have the distribution function of the first passage time when n faults are corrected

$$G_n(t) = \sum_{i=0}^{n-1} A_i^n (1 - e^{-pDk^i t}), \tag{2.17}$$

where $G_0(t) \equiv 1$.

2.3.2 Distribution of the Number of Faults Corrected Up to a Specified Time

Let S_n $(n = 1,\ 2,\ \ldots)$ be random variables representing the n-th successful correction time of detected faults. Since $X(t)$ is a counting process [53] (see Fig. 2.1), we have the following equivalent relation:

$$\{S_n \leq t\} \Longleftrightarrow \{X(t) \geq n\}. \tag{2.18}$$

Therefore, we get

$$\Pr\{S_n \leq t\} = \Pr\{X(t) \geq n\}. \tag{2.19}$$

Let $P_n(t)$ denote the probability that n faults are corrected up to testing time t. From (2.17) and (2.19), we obtain the probability mass function $P_n(t)$ as

$$\begin{aligned} P_n(t) &= \Pr\{X(t) = n\} \\ &= \Pr\{X(t) \geq n\} - \Pr\{X(t) \geq n+1\} \\ &= \Pr\{S_n \leq t\} - \Pr\{S_{n+1} \leq t\} \\ &= G_n(t) - G_{n+1}(t). \end{aligned} \tag{2.20}$$

Suppose that the initial fault content in the system prior to the testing, N, is known. Then, we can derive the expected number of faults corrected up to testing time t using (2.20) as

$$\begin{aligned} \mathrm{E}[X(t)|N] &= \sum_{n=0}^{N} nP_n(t) \\ &= \sum_{n=0}^{N} n\{G_n(t) - G_{n+1}(t)\} \\ &= \sum_{n=1}^{N} G_n(t). \end{aligned} \tag{2.21}$$

It is noted that $P_N(t) = G_N(t)$ since $G_{N+1}(t) = 0$.

Furthermore, the second moment of $X(t)$ is given by

$$\begin{aligned} \mathrm{E}[X(t)^2|N] &= \sum_{n=0}^{N} n^2 P_n(t) \\ &= \sum_{n=1}^{N} (2n-1) G_n(t). \end{aligned} \tag{2.22}$$

Therefore, the variance of $X(t)$ is calculated by

$$\mathrm{Var}[X(t)|N] = \sum_{n=1}^{N} (2n-1) G_n(t) - \left\{ \sum_{n=1}^{N} G_n(t) \right\}^2. \tag{2.23}$$

2.3.3 Expected Number of Faults Detected Up to a Specified Time

We introduce a new random variable $Z(t)$ representing the cumulative number of faults detected up to testing time t. Let $M_i(t)$ be the expected number of faults detected up to

time t on the condition that i faults have been already corrected at time zero, i.e.

$$M_i(t) = \mathrm{E}[Z(t)|X(0) = i], \tag{2.24}$$

which is called a Markov renewal function [42]. Supposing that the initial fault content N is known, we obtain the following renewal equations:

$$M_i(t) = F_i(t) + Q_{i,i} * M_i(t) + Q_{i,i+1} * M_{i+1}(t) \qquad (i = 0,\ 1,\ 2,\ \ldots,\ N-1), \tag{2.25}$$

where $M_N(t) = 0$. Using the L-S transforms of $M_i(t)$ $(i = 0,\ 1,\ 2,\ \ldots,\ N-1)$, from (2.15) we get

$$\begin{aligned} \widetilde{M}_0(s) &= \frac{1}{p}\sum_{n=1}^{N}\prod_{i=0}^{n-1}\frac{pDk^i}{s+pDk^i} \\ &= \frac{1}{p}\sum_{n=1}^{N}\widetilde{G}_n(s). \end{aligned} \tag{2.26}$$

Inverting (2.26) and rewriting $M_0(t)$ as $M(t|N)$, we have the expected number of faults detected up to time t as

$$\begin{aligned} M(t|N) &= \frac{1}{p}\sum_{n=1}^{N}G_n(t) \\ &= \frac{1}{p}\mathrm{E}[X(t)|N]. \end{aligned} \tag{2.27}$$

We consider that all faults detected by the testing are divided into two types. One are the faults which are successfully corrected, the other are the faults detected again due to imperfect debugging [64]. Then, the expected number of faults debugged imperfectly is given by

$$\begin{aligned} D(t|N) &= M(t|N) - \mathrm{E}[X(t)|N] \\ &= \frac{q}{p}\mathrm{E}[X(t)|N]. \end{aligned} \tag{2.28}$$

2.3.4 Distribution of the Time between Software Failures

Let X_l $(l = 1,\ 2,\ \ldots)$ be a random variable representing the time interval between the $(l-1)$-st and the l-th software failure-occurrences and $\Phi_l(x)$ be a distribution function of X_l. It is noted that X_l depends on the number of the faults corrected up to the $(l-1)$-st software failure-occurrence, however, this is not explicitly known.

Here, let C_l be a random variable representing the number of faults corrected up to the $(l-1)$-st software failure-occurrence. Then, C_l follows a binomial distribution having the following probability mass function:

$$\Pr\{C_l = i\} = \binom{l-1}{i} p^i q^{l-1-i} \qquad (i = 0,\ 1,\ 2,\ \ldots,\ l-1), \tag{2.29}$$

where $\binom{l-1}{i} \equiv (l-1)!/[(l-1-i)!i!]$ denotes a binomial coefficient. From (2.29), at the $(l-1)$-st software failure-occurrence, the expected number of corrected faults is given by $p(l-1)$.

Furthermore, it is evident that

$$\Pr\{X_l \le x | C_l = i\} = F_i(x), \tag{2.30}$$

which is given by (2.3). Accordingly, we can get the distribution function for X_l as

$$\begin{aligned} \Phi_l(x) &\equiv \Pr\{X_l \le x\} \\ &= \sum_{i=0}^{l-1} \Pr\{X_l \le x | C_l = i\} \Pr\{C_l = i\} \\ &= \sum_{i=0}^{l-1} \binom{l-1}{i} p^i q^{l-1-i} (1 - e^{-Dk^i x}). \end{aligned} \tag{2.31}$$

Then, we have the reliability function for X_l as

$$\begin{aligned} R_l(x) &\equiv \Pr\{X_l > x\} \\ &= 1 - \Phi_l(x) \\ &= \sum_{i=0}^{l-1} \binom{l-1}{i} p^i q^{l-1-i} e^{-Dk^i x}. \end{aligned} \tag{2.32}$$

Furthermore, the hazard rate for X_l is given by

$$\begin{aligned} \lambda_l(x) &\equiv \frac{\frac{d}{dx}\Phi_l(x)}{R_l(x)} \\ &= \frac{D\sum_{i=0}^{l-1} \binom{l-1}{i} (kp)^i q^{l-1-i} e^{-Dk^i x}}{\sum_{i=0}^{l-1} \binom{l-1}{i} p^i q^{l-1-i} e^{-Dk^i x}}. \end{aligned} \tag{2.33}$$

The expectation of random variable X_l is defined by

$$\mathrm{E}[X_l] \equiv \int_0^\infty R_l(x) dx. \tag{2.34}$$

We call (2.34) the mean time between software failures (MTBSF). From (2.32), we can derive $\mathrm{E}[X_l]$ as

$$\mathrm{E}[X_l] = \frac{(p/k+q)^{l-1}}{D}. \tag{2.35}$$

Apparently, the following inequality holds for arbitrary natural number l:

$$\mathrm{E}[X_l] < \mathrm{E}[X_{l+1}] \qquad (l = 1,\ 2,\ \ldots). \tag{2.36}$$

That is, a software reliability growth occurs whenever a software failure is observed.

2.4 Optimal Software Release Problems

One of the most interesting problems for software development managers is deciding when to deliver a software system to customers. This decision problem is called an optimal software release one. Considering evaluation criteria such as achieved software reliability, cost, delivery time and so on, we need to estimate a testing termination time. Based on the software reliability model discussed above, we investigate optimal software release problems introducing total expected software cost and software reliability criteria [60, 62].

2.4.1 Reliability-Optimal Software Release Policies

We consider an optimal software release problem that decides the total testing time required to attain a software reliability objective. Suppose that we deliver a software system at the time point when m faults have been detected by the testing. Then, from (2.35) the mean time to software release is given by

$$\sum_{l=1}^{m} \mathrm{E}[X_l] = \frac{1-(p/k+q)^m}{pD(1-1/k)}. \tag{2.37}$$

And from (2.32) software reliability $R(x_0; m)$ for specified operational time x_0 is given by

$$R(x_0; m) = \sum_{j=0}^{m} \binom{m}{j} p^j q^{m-j} e^{-Dk^j x_0}. \tag{2.38}$$

Letting R_0 be a software reliability objective for the operational time x_0, the minimum integer m which satisfies $R(x_0; m) \geq R_0$ is the optimum number of detected faults, m^*, since $R(x_0; m)$ in (2.38) is a monotonically increasing function with respect to the number

of detected faults m. That is, if $R(x_0;0) < R_0$, then there exists a finite and unique $m = m_0$ $(1 \leq m_0 < \infty)$ which satisfies the following inequalities:

$$R(x_0;m) \geq R_0 \qquad \text{and} \qquad R(x_0;m-1) < R_0. \tag{2.39}$$

Thus, we have the following theorem for a reliability-optimal software release problem.

Theorem 1 *Suppose that* $x_0 \geq 0$ *and* $0 < R_0 < 1$.

(1) *If* $e^{-Dx_0} < R_0$, *then the optimum number of detected faults is* $m^* = m_0$, *and the optimum software release time is*

$$T^* = \frac{1-(p/k+q)^{m_0}}{pD(1-1/k)},$$

where m_0 *is an integer number which satisfies* (2.39).

(2) *If* $e^{-Dx_0} \geq R_0$, *then the optimum number of detected faults is* $m^* = 0$, *and the optimum software release time is* $T^* = 0$.

2.4.2 Cost-Optimal Software Release Policies

The following cost-parameters are defined:

c_1 : debugging cost per fault during the testing phase,

c_2 : debugging cost per fault during the operation phase $(c_2 > c_1 > 0)$,

c_3 : testing cost per unit time $(c_3 > 0)$.

Suppose that the expected number of faults detected eventually is $M = \lceil N/p \rceil$ since $M(\infty|N) = N/p$ from (2.27), where $\lceil x \rceil$ denotes the least integer that is not smaller than x. Then, the total expected software cost is given by

$$C(m) = c_1 m + c_2(M-m) + c_3 \frac{1-(p/k+q)^m}{pD(1-1/k)} \qquad (0 \leq m \leq M). \tag{2.40}$$

Therefore, the integer m minimizing $C(m)$ in (2.40) is the optimum number of detected faults, m^*.

For finding $m = m^*$ minimizing $C(m)$ in (2.40), we define the following equation:

$$\begin{aligned} Y(m) &= C(m+1) - C(m) \\ &= -(c_2 - c_1) + \frac{c_3}{D}(p/k+q)^m \qquad (0 \le m \le M-1). \end{aligned} \tag{2.41}$$

It is noted that $Y(m)$ is a monotonically increasing function with respect to the number of detected faults, m, since $p+q=1$ and $0 < k < 1$. If $Y(0) < 0$, then the minimum m which holds $Y(m) \ge 0$ satisfies inequalities $C(m+1) \ge C(m)$ and $C(m) < C(m-1)$. Accordingly, the optimum number of detected faults, $m^* = m_1$, is given by

$$m_1 = \left\lceil \frac{\ln\{D(c_2 - c_1)/c_3\}}{\ln(p/k+q)} \right\rceil. \tag{2.42}$$

Thus, we have the following theorem for a cost-optimal software release problem.

Theorem 2 *Suppose that $c_2 > c_1 > 0$ and $c_3 > 0$.*

(1) *If $D > \dfrac{c_3}{c_2 - c_1} \ge \dfrac{D}{(p/k+q)^{M-1}}$, then the optimum number of detected faults is $m^* = m_1$ $(1 \le m_1 \le M-1)$, and the optimum software release time is*

$$T^* = \frac{1 - (p/k+q)^{m_1}}{pD(1-1/k)},$$

where m_1 is given by (2.42).

(2) *If $\dfrac{D}{(p/k+q)^{M-1}} > \dfrac{c_3}{c_2 - c_1}$, then the optimum number of detected faults is $m^* = M$, and the optimum software release time is*

$$T^* = \frac{1 - (p/k+q)^{M}}{pD(1-1/k)}.$$

(3) *If $D \le \dfrac{c_3}{c_2 - c_1}$, then the optimum number of detected faults is $m^* = 0$, and the optimum software release time is $T^* = 0$.*

2.4.3 Cost-Reliability-Optimal Software Release Policies

We discuss an optimal software release problem which evaluates both software cost and reliability criteria simultaneously. Consider decision policy on the optimum number of faults to be detected by the release time which minimizes the total expected software cost

$C(m)$ in (2.40) subject to the condition that software reliability $R(x_0; m)$ in (2.38) satisfies reliability objective R_0. The optimal software release problem can be formulated as follows: For a specified operational time x_0 ($x_0 \geq 0$),

$$\left.\begin{array}{ll} \text{minimize} & C(m) \\ \text{subject to} & R(x_0; m) \geq R_0,\ 0 < R_0 < 1 \end{array}\right\}. \tag{2.43}$$

This problem is called as a cost-reliability-optimal software release problem [60]. From Sections 2.4.1 and 2.4.2, we have the following theorem for a cost-reliability-optimal software release problem.

Theorem 3 *Suppose that* $c_2 > c_1 > 0$, $c_3 > 0$, $x_0 \geq 0$, *and* $0 < R_0 < 1$.

(1) *If* $D > \dfrac{c_3}{c_2 - c_1} \geq \dfrac{D}{(p/k+q)^{M-1}}$ *and* $e^{-Dx_0} < R_0$, *then the optimum number of detected faults is* $m^* = \max\{m_0,\ m_1\}$, *and the optimum software release time is*

$$T^* = \frac{1 - (p/k+q)^{\max\{m_0,\ m_1\}}}{pD(1-1/k)}.$$

(2) *If* $\dfrac{D}{(p/k+q)^{M-1}} > \dfrac{c_3}{c_2 - c_1}$ *and* $e^{-Dx_0} < R_0$, *then the optimum number of detected faults is* $m^* = \max\{m_0,\ M\}$, *and the optimum software release time is*

$$T^* = \frac{1 - (p/k+q)^{\max\{m_0,\ M\}}}{pD(1-1/k)}.$$

(3) *If* $D > \dfrac{c_3}{c_2 - c_1} \geq \dfrac{D}{(p/k+q)^{M-1}}$ *and* $e^{-Dx_0} \geq R_0$, *then the optimum number of detected faults is* $m^* = m_1$, *and the optimum software release time is*

$$T^* = \frac{1 - (p/k+q)^{m_1}}{pD(1-1/k)}.$$

(4) *If* $\dfrac{D}{(p/k+q)^{M-1}} > \dfrac{c_3}{c_2 - c_1}$ *and* $e^{-Dx_0} \geq R_0$, *then the optimum number of detected faults is* $m^* = M$, *and the optimum software release time is*

$$T^* = \frac{1 - (p/k+q)^{M}}{pD(1-1/k)}.$$

(5) *If* $D \leq \dfrac{c_3}{c_2 - c_1}$ *and* $e^{-Dx_0} < R_0$, *then the optimum number of detected faults is* $m^* = m_0$, *and the optimum software release time is*

$$T^* = \frac{1 - (p/k+q)^{m_0}}{pD(1-1/k)}.$$

(6) *If* $D \leq \dfrac{c_3}{c_2 - c_1}$ *and* $e^{-Dx_0} \geq R_0$, *then the optimum number of detected faults is* $m^* = 0$, *and the optimum software release time is* $T^* = 0$.

2.5 Numerical Examples

Using the software reliability model discussed above, we show some numerical illustrations for software reliability measurement and its application.

The distribution functions of the first passage time to the specified number of corrected faults, $G_n(t)$, in (2.17) are shown in Fig. 2.4 for various perfect debugging rates, p's, where $n = 10$, $D = 0.2$, and $k = 0.9$. We can calculate the testing time t_r to reach an objective probability r $(0 \leq r \leq 1)$ such that $G_n(t) = r$. In case of $r = 0.9$, the values of t_r for various perfect debugging rates p's are shown in Table 2.1. We can see that the smaller perfect debugging rate p becomes, the more difficult it is to correct faults.

Table 2.1. Testing time t_r to reach the objective probability r $(r = 0.9,\ n = 10,\ D = 0.2,\ k = 0.9)$.

p	t_r
1.0	120.9
0.95	127.3
0.9	134.4
0.85	142.3
0.8	151.2

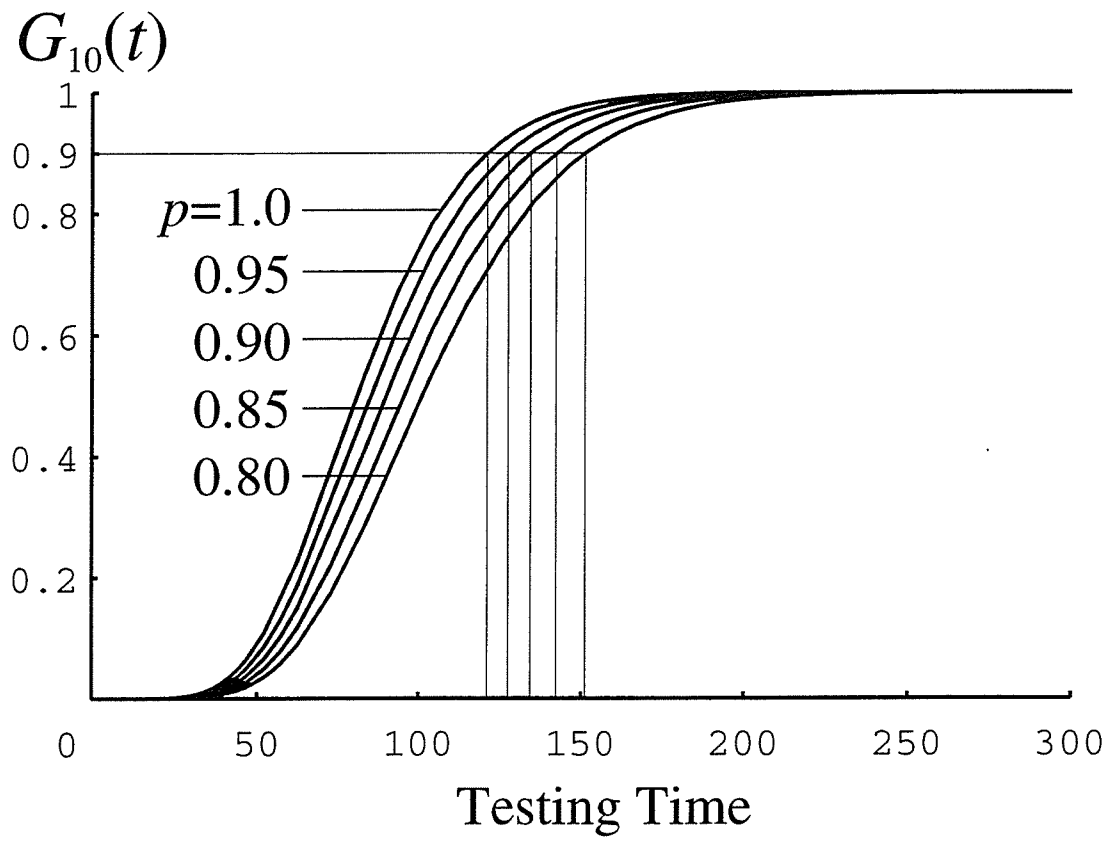

Fig. 2.4. Dependence of perfect debugging rate p on $G_{10}(t)$ ($D = 0.2,\ k = 0.9$).

The expected numbers of faults corrected up to testing time t, $\mathrm{E}[X(t)|N]$, in (2.21) for various p's are shown in Fig. 2.5 where $N = 20$, $D = 0.2$, and $k = 0.9$.

The variances of the number of faults corrected up to testing time t, $\mathrm{Var}[X(t)|N]$, in (2.23) for various p's are shown Fig. 2.6 where $N = 20$, $D = 0.2$, and $k = 0.9$. As shown in Fig. 2.6, $\mathrm{Var}[X(t)|N]$ is a convex function with respect to testing time t with

$$\mathrm{Var}[X(0)|N] = \mathrm{Var}[X(\infty)|N] = 0. \tag{2.44}$$

This means that the correctability of faults in debugging is unstable during the early stage of the testing, and as the testing is in progress, it becomes stable. Then, we can calculate testing time t_s maximizing $\mathrm{Var}[X(t)|N]$, which is regarded as a minimum testing time required to stabilize the fault-correctability. The values of t_s for various p's are shown in Table 2.2. As shown in Fig. 2.6 and Table 2.2, we can see that the smaller the perfect-debugging rate p becomes, the more difficult it is to stabilize the fault-correctability.

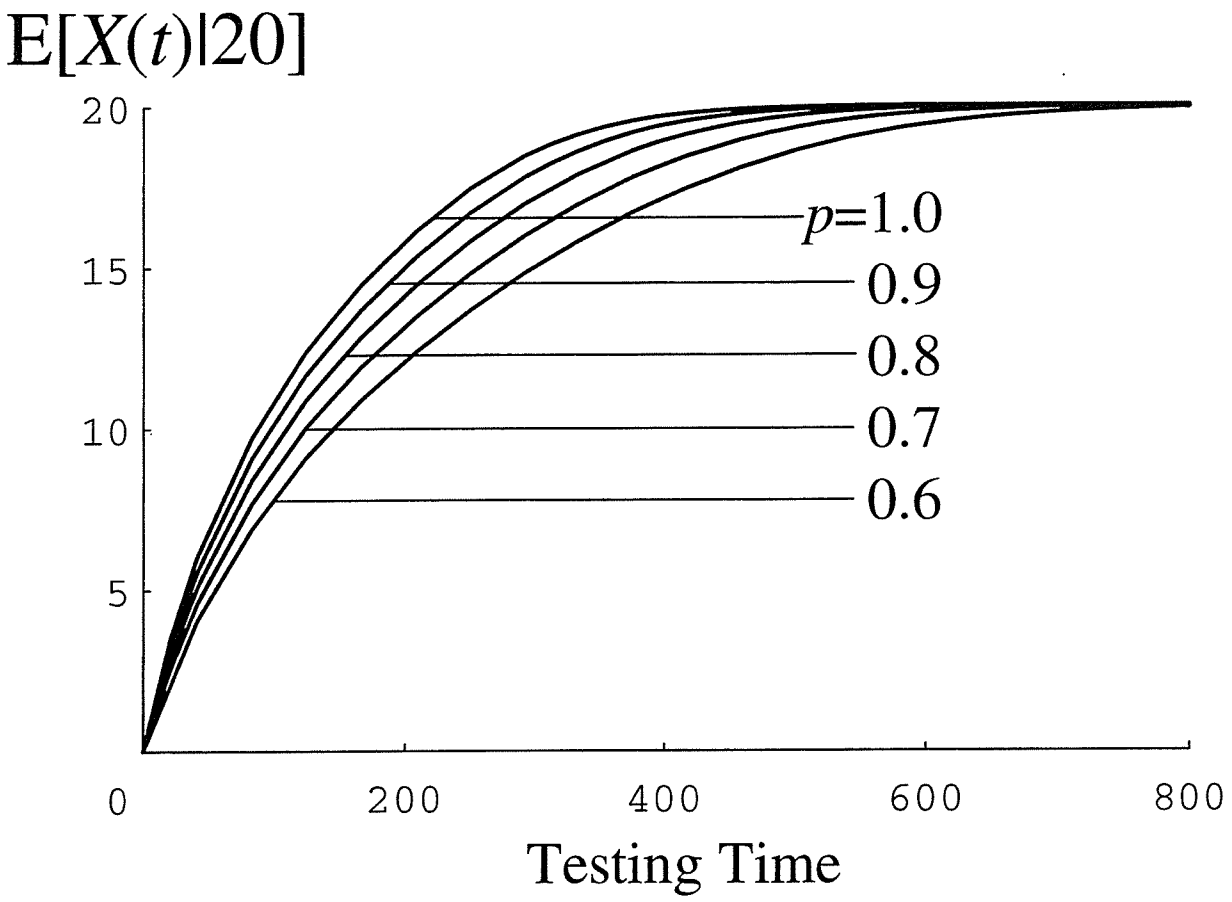

Fig. 2.5. Dependence of perfect debugging rate p on $\mathrm{E}[X(t)|20]$ ($D = 0.2,\ k = 0.9$).

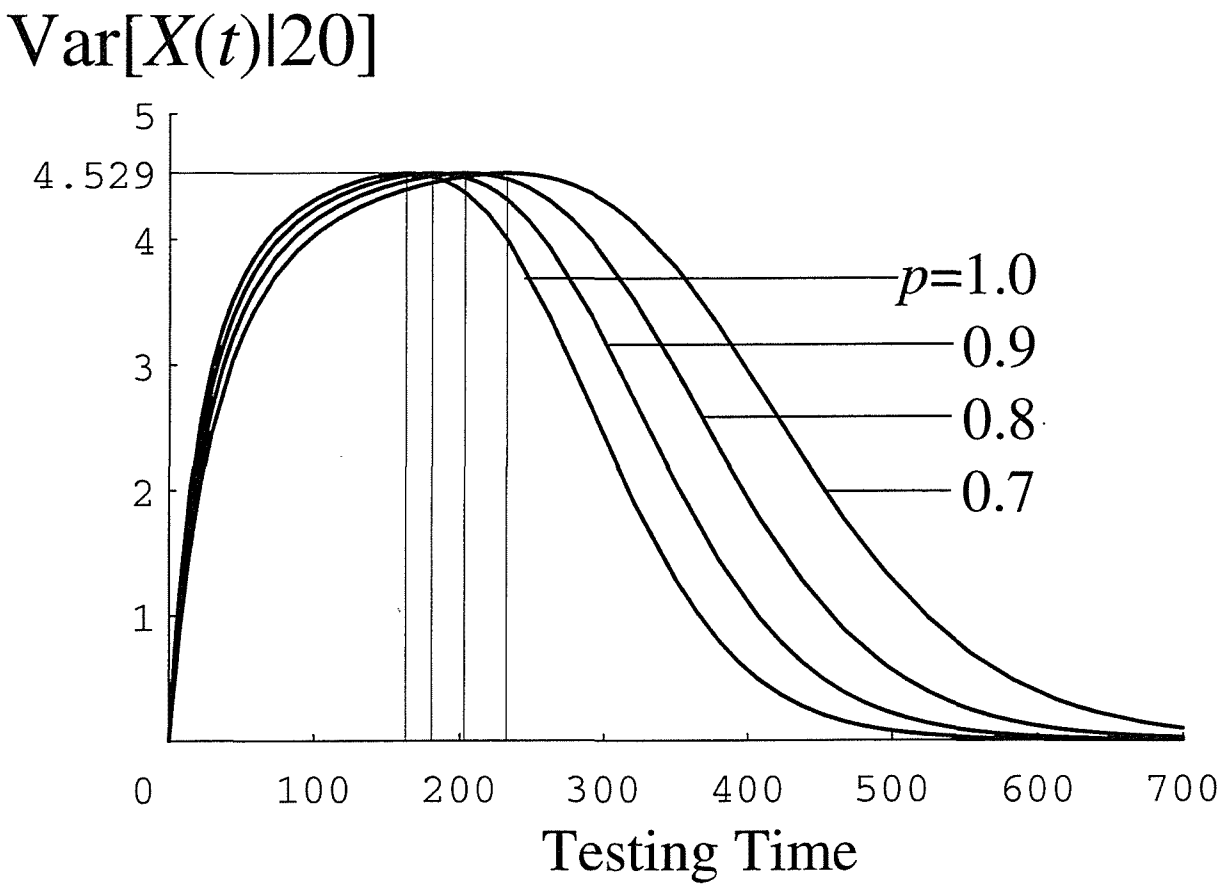

Fig. 2.6. Dependence of perfect debugging rate p on $\mathrm{Var}[X(t)|20]$ $(D = 0.2,\ k = 0.9)$.

Table 2.2. Testing time t_s maximizing $\mathrm{Var}[X(t)|N]$ $(D = 0.2,\ k = 0.9)$.

N \ p	1.0	0.9	0.8	0.7	$\mathrm{E}[X(t_s)\|N]$	$\mathrm{Var}[X(t_s)\|N]$
10	45.193	50.217	56.491	64.561	6.385	3.346
20	162.55	180.62	203.19	232.22	14.26	4.529
30	462.23	513.59	577.79	660.33	22.74	4.744

Furthermore, the coefficients of variation (cv) of $X(t)$, $\mathrm{cv}[X(t)|N]$, defined as

$$\mathrm{cv}[X(t)|N] \equiv \frac{\sqrt{\mathrm{Var}[X(t)|N]}}{\mathrm{E}[X(t)|N]}, \tag{2.45}$$

for various p's are shown in Fig. 2.7 where $N = 20$, $D = 0.2$, and $k = 0.9$. As shown in Fig. 2.7, $\mathrm{cv}[X(t)|N]$ is a monotonically decreasing function with respect to testing time t. We can calculate the testing time t_c to reach an objective cv, c. In case of $c = 0.1$, the values of t_c for various p's are shown in Table 2.3 along with $\mathrm{E}[X(t_c)|N]$ and $\mathrm{Var}[X(t_c)|N]$.

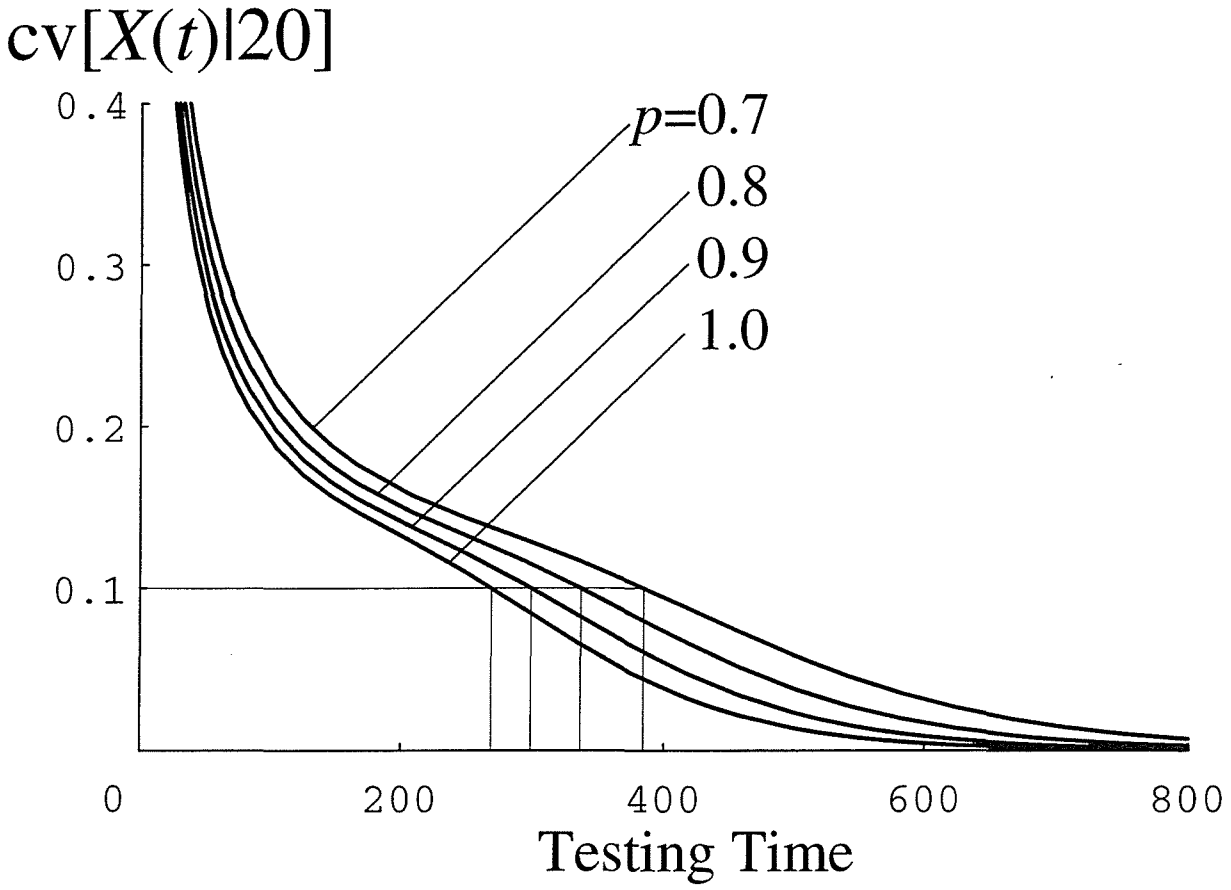

Fig. 2.7. Dependence of perfect debugging rate p on $\mathrm{cv}[X(t)|20]$ ($D = 0.2$, $k = 0.9$).

Table 2.3. Testing time t_c to reach the objective cv ($c = 0.1,\ D = 0.2,\ k = 0.9$).

N \ p	1.0	0.9	0.8	0.7	$\mathrm{E}[X(t_c)\|N]$	$\mathrm{Var}[X(t_c)\|N]$
10	97.342	108.16	121.68	139.06	9.500	0.9019
20	269.03	298.92	336.28	384.32	17.95	3.222
30	413.04	458.93	516.30	590.05	21.77	4.740

$\mathrm{Var}[X(t)|N]$'s for various k's representing the decreasing ratio of the hazard rate are shown in Fig. 2.8 where $N = 20$, $D = 0.2$, and $p = 0.9$. We can see that the smaller k becomes, the smaller the maximum of $\mathrm{Var}[X(t)|N]$ becomes, while the longer time $\mathrm{Var}[X(t)|N]$ takes to converge.

The expected number of faults detected up to testing time t, $M(t|N)$, in (2.27) is shown in Fig. 2.9 along with the expected number of imperfect debugging faults, $D(t|N)$, in (2.28) where $N = 30$, $D = 0.2$, $k = 0.9$, and $p = 0.9$. In this case,

$$M(\infty|30) = 33.3,\ D(\infty|30) = \frac{q}{p} \cdot 30 = 33.3q. \tag{2.46}$$

Then, $(100q)\%$ of the cumulative number of faults detected eventually is imperfectly debugged.

The reliability function, $R_l(x)$, in (2.32) and the hazard rate, $\lambda_l(x)$, in (2.33) for various l's are shown in Figs. 2.10 and 2.11 where $D = 0.2$, $k = 0.9$, and $p = 0.9$, respectively, and the values of MTBSF for various l's are shown in Table 2.4. Figs. 2.10, 2.11, and Table 2.4 show that a software reliability growth during the testing occurs whenever a software failure occurs.

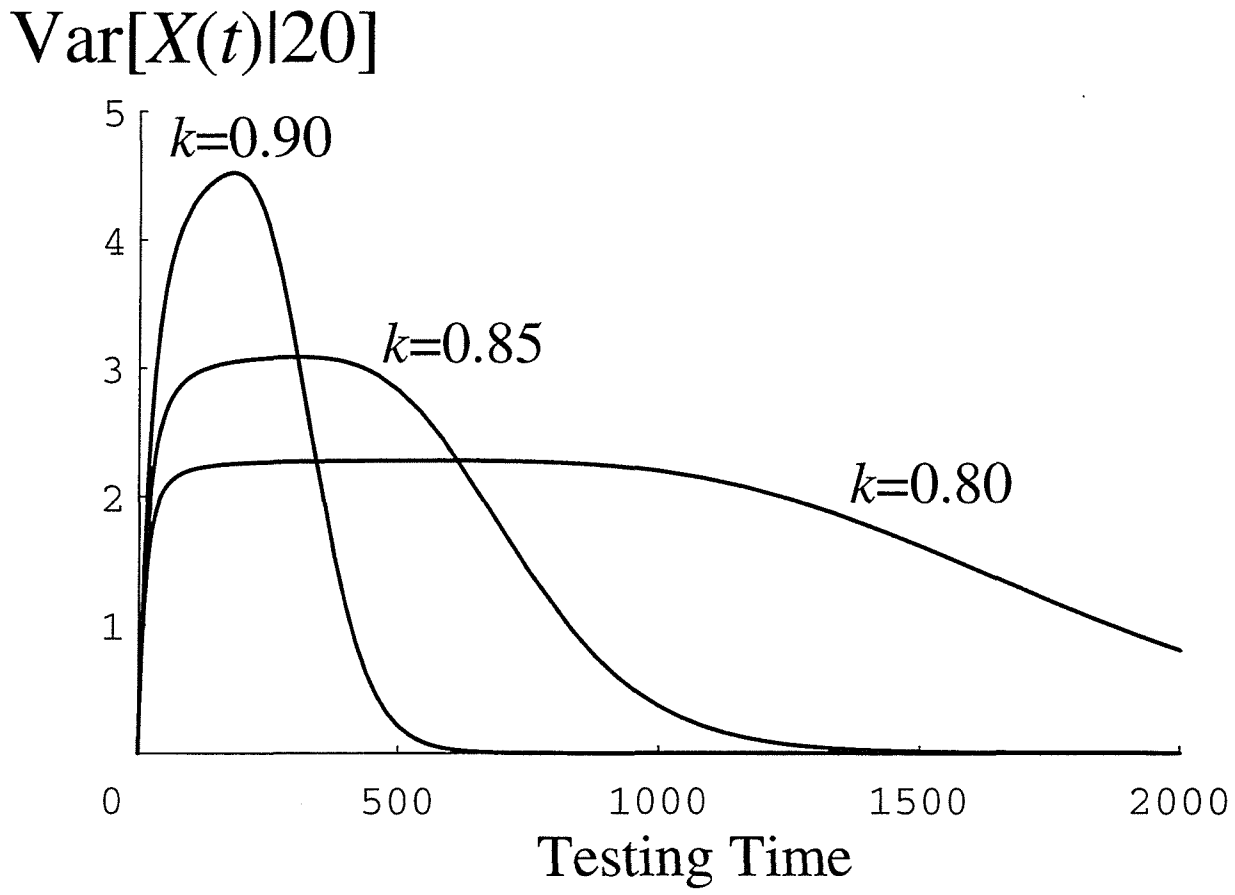

Fig. 2.8. Dependence of decreasing ratio of the hazard rate, k on $\mathrm{Var}[X(t)|20]$ ($D = 0.2$, $p = 0.9$).

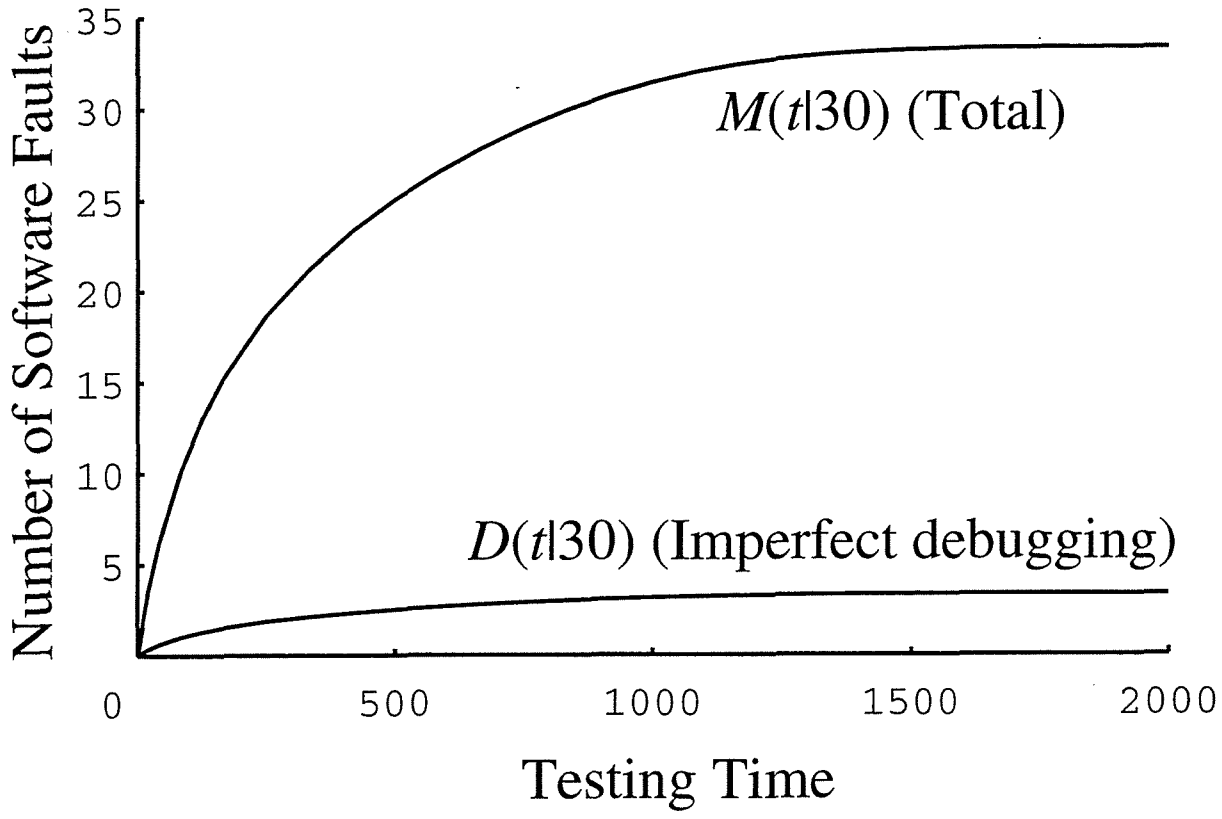

Fig. 2.9. $M(t|30)$ and $D(t|30)$ ($D = 0.2,\ k = 0.9,\ p = 0.9$).

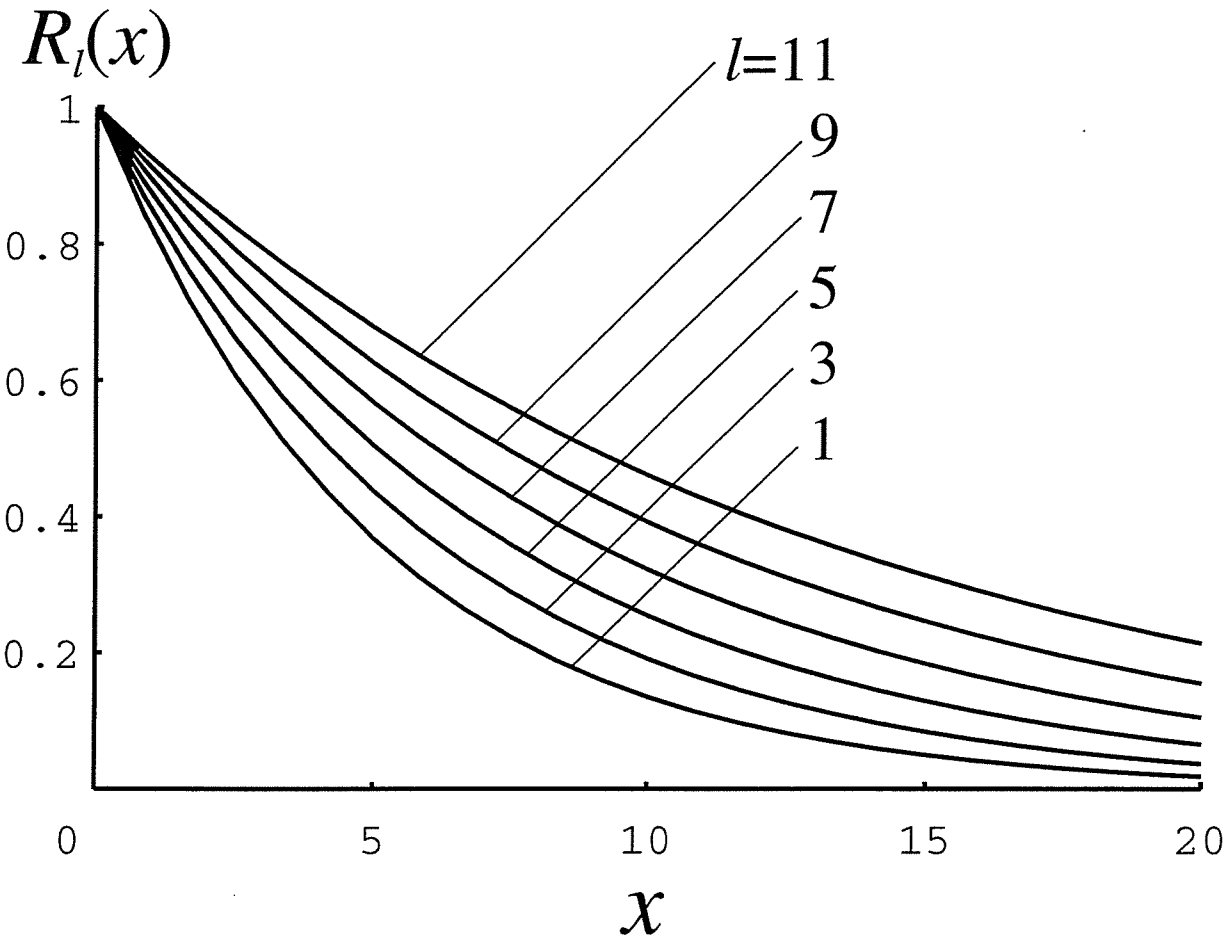

Fig. 2.10. Dependence of number of failures l on software reliability $R_l(x)$ ($D = 0.2$, $k = 0.9$, $p = 0.9$).

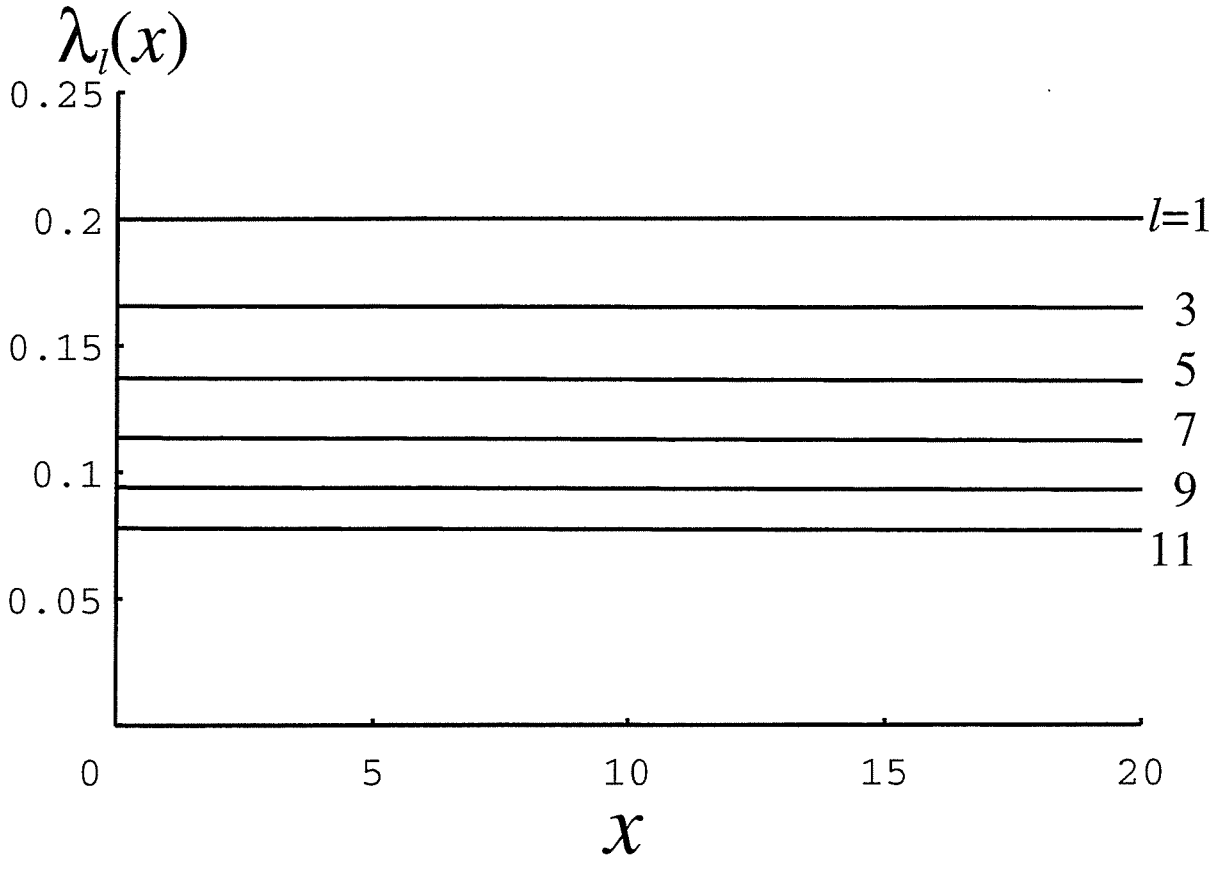

Fig. 2.11. Dependence of number of failures l on hazard rate $\lambda_l(x)$ ($D = 0.2,\ k = 0.9,\ p = 0.9$).

Table 2.4. MTBSF $\mathrm{E}[X_l]$ ($p = 0.9,\ D = 0.2,\ k = 0.9$).

l	$\mathrm{E}[X_l]$
1	5.000
2	5.500
3	6.050
4	6.655
5	7.321
6	8.053
7	8.858
8	9.744
9	10.72
10	11.79

Next, we show numerical examples of the optimal software release problems. Tables 2.5 and 2.6 show relationships between the perfect debugging rate p and the optimum software release time T^* for Theorem 1 and Theorem 2 where $x_0 = 2.5,\ R_0 = 0.95,\ c_1 = 1.0,\ c_2 = 10.0,\ c_3 = 0.2,\ D = 0.2,\ k = 0.9$, and $N = 30$, respectively. We can see that improving the perfect debugging rate is more efficient to quicken the optimum software release time when the perfect debugging rate is low.

Furthermore, we show a numerical example on the cost-reliability-optimal software release problem. Consider the optimal software release problem formulated as follows:

$$\left.\begin{array}{ll} \text{minimize} & C(m) \\ \text{subject to} & R(2.5; m) \geq 0.95 \end{array}\right\}. \tag{2.47}$$

The minimum satisfying $R(2.5; m) \geq 0.95$ is $m_0 = 25$ and the optimum number of detected faults, m minimizing $C(m)$, is $m_1 = 24$. From Theorem 3 (1), the optimum number of detected faults is $m^* = \max\{25,\ 24\} = 25$, i.e. though the total expected software cost $C(m)$ is minimized when $m = 24$, the software reliability objective is not satisfied. Accordingly, the optimum software release time is given by

$$T^* = \frac{1 - (p/k + q)^{25}}{pD(1 - 1/k)} = 491.74.$$

Table 2.5. Optimum release time T^* for Theorem 1 ($x_0 = 2.5,\ R_0 = 0.95,\ D = 0.2,\ k = 0.9$).

p	m^*	T^*
1.0	22	411.96
0.9	25	491.74
0.8	28	554.22
0.7	32	642.11
0.6	37	741.82
0.5	45	935.40
0.4	56	1171.99
0.3	75	1604.38
0.2	113	2471.52
0.1	226	5017.14

Table 2.6. Optimum release time T^* for Theorem 2 ($c_1 = 1.0,\ c_2 = 10.0,\ c_3 = 0.2,\ D = 0.2,\ k = 0.9,\ N = 30$).

p	m^*	T^*	$C(m^*)$
1.0	21	366.26	184.25
0.9	24	442.49	212.50
0.8	26	458.62	237.72
0.7	30	543.84	268.77
0.6	35	642.90	313.58
0.5	41	735.98	378.20
0.4	51	920.99	475.20
0.3	68	1245.57	636.92
0.2	100	1801.35	960.27
0.1	199	3607.86	1930.37

2.6 Concluding Remarks

In this chapter, we have developed an software reliability model considering the imperfect debugging environment where software faults detected by testing are not always corrected/

removed. This model has been described by a Markov process. Various interesting stochastic quantities for software reliability measurement and assessment have been derived from the model and their numerical examples are illustrated. Application of this model to several optimal software release problems has been also discussed.

Chapter 3

Software Reliability Modeling with Two Types of Failures

3.1 Introduction

Many software reliability models have been developed for the purpose of estimating software reliability quantitatively. Among these, it is said that software reliability growth models have high validity and usefulness. These models can describe a software fault-detection or a software failure-occurrence phenomenon during the testing phase in the software development process and/or the operation phase. A software failure is defined as an unacceptable departure from program operation caused by a fault remaining in the system.

Most of representative software reliability growth models are based on the following assumptions:

- The debugging is perfect.
- The hazard rate for software failures is proportional to the residual current fault content.

The above assumptions imply that all of faults latent in the system are corrected and a hazard rate converges to zero. However, in general, it is impossible to remove all faults from the system and deliver a fault-free software system to the customers. Proficient programmers recognize the existence of the regenerative faults [50]. Accordingly, the software system may show the same failure-occurrence phenomenon as a hardware system during the last stage of the testing phase. Furthermore, debugging activities contain human-error factors such as typographical errors or misunderstandings about the test results. That

is, the actual debugging environment is the imperfect one. Several imperfect debugging models have been proposed [18, 38, 48, 52].

In this chapter, modifying and extending the model of Chapter 2 (hereafter referred to as the basic model), we discuss a software reliability model with the existence of the two types of software failures. One is due to the original faults remaining in the system prior to testing. The other is due to faults randomly introduced or regenerated during the testing phase. The former and the latter types of software failure-occurrence phenomena are described by a geometrically decreasing and a constant hazard rate, respectively. Defining a random variable representing the cumulative number of faults successfully corrected up to a specified time point, we use a Markov process to formulate this model [42]. From this model, several interesting quantities for software reliability assessment are derived. Furthermore, the method of maximum-likelihood estimation of model parameters is presented. Finally, numerical examples of software reliability measurement and assessment based on the actual testing data are illustrated.

3.2 Model Description

In this chapter, we assume that the following two types of software failures exist:

F1: software failures due to faults originally latent in the system prior to the testing.

F2: software failures due to faults randomly introduced or regenerated during the testing phase.

The extended software reliability growth model constructed here is based on the following assumptions:

A1. The debugging activity is performed as soon as the software failure occurs.

A2. The debugging activity for the fault which has caused the corresponding software failure succeeds with probability p $(0 < p \leq 1)$, and fails with probability $q(= 1 - p)$. We call p the perfect debugging rate.

A3. The hazard rate for F1 is constant between software failures and decreases geometrically as each fault is corrected. The hazard rate for F2 is constant throughout the testing phase [31].

A4. The debugging activity is performed without distinguishing between F1 and F2.

A5. The probability that two or more software failures occur simultaneously is negligible.

A6. At most one fault is corrected when the debugging activity is performed, and the fault-correction time is not considered.

From assumptions A3 and A4, when i faults have been corrected, the hazard rate for the next software failure-occurrence is given by

$$z_i(\tau) = Dk^i + \theta$$
$$(i = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1,\ \theta \geq 0,\ \tau \geq 0), \tag{3.1}$$

where D is the initial hazard rate for F1, k is the decreasing ratio of the hazard rate, and θ is the hazard rate for F2. In the actual testing environment, it is difficult to specify immediately when the faults are introduced or regenerated and whether the software failure is due to the regenerative fault or not. Therefore, we assume that the faults are introduced randomly in the system and that the software failures caused by the introduced faults occur randomly during the testing phase. Accordingly, the hazard rate for F2 is on average constant through the testing phase. Furthermore, we consider that increase in the hazard rate by introduction of other new faults during debugging can be negligible [9]. A sample function of $z_i(\tau)$ is shown in Fig. 3.1. We can see that the hazard rate $z_i(\tau)$ decreases when the debugging succeeds, but remains constant when debugging fails.

The expression of (3.1) comes from the point of view that software reliability depends on the debugging efforts, not the residual fault content. We do not note how many faults remain in the software system. Equation (3.1) describes a software failure-occurrence phenomenon where a software system has high frequency of software failure-occurrence and perfect debugging contributes largely to the improvement of software reliability during the early stage of the testing phase; later in the testing phase, the decrease in the hazard rate is slower even if debugging is perfect [35, 56, 57].

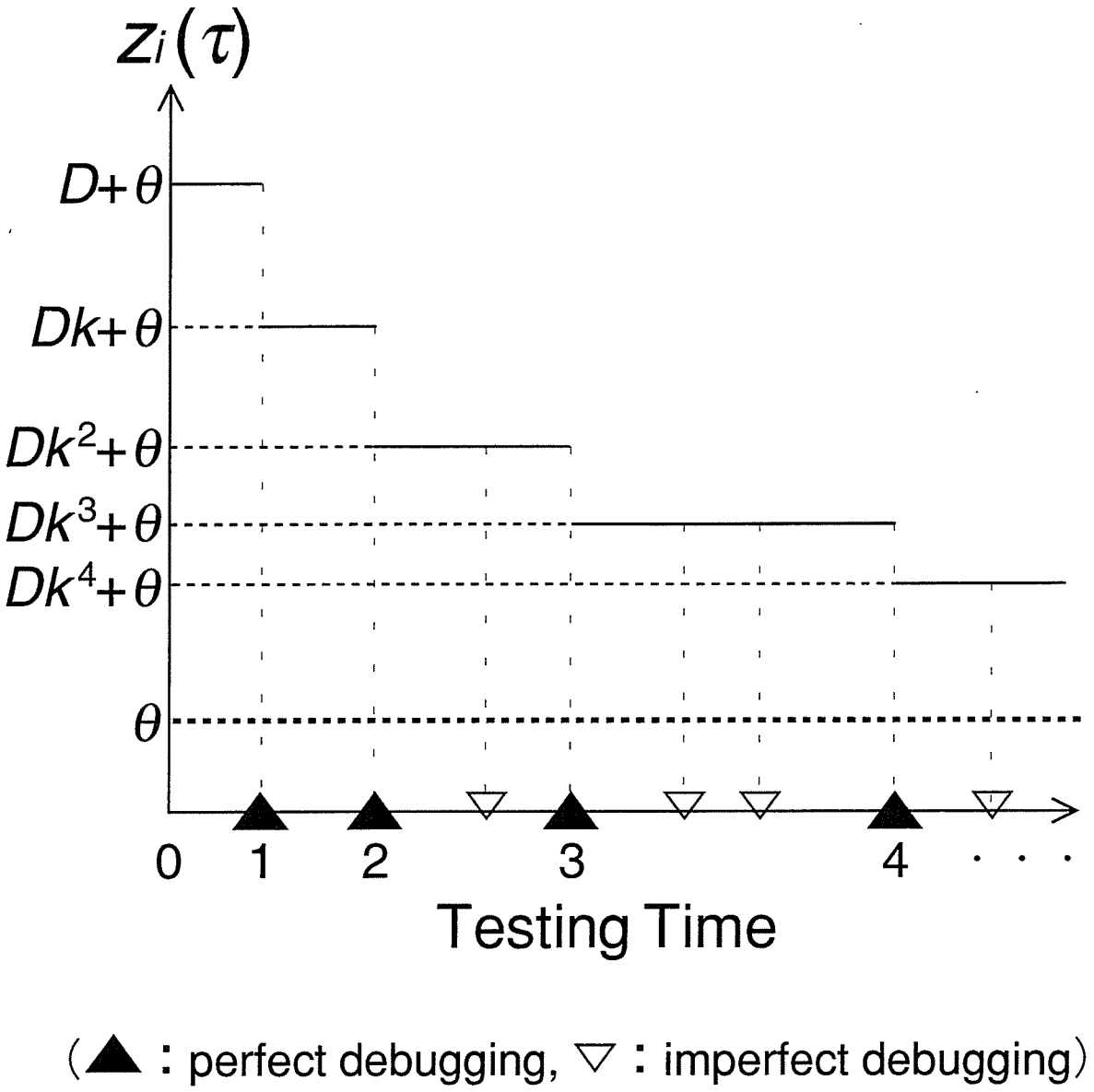

Fig. 3.1. A sample realization of hazard rate $z_i(\tau)$ for software reliability modeling with two types of failures.

From (3.1), the distribution function for the next software failure is given by

$$F_i(\tau) = 1 - e^{-(Dk^i+\theta)\tau}. \tag{3.2}$$

Furthermore, let $Q_{i,j}(\tau)$ denote the one step transition probability that after making a transition into state i, the process $\{X(t), t \geq 0\}$ makes a transition into state j by time τ.

Then, $Q_{i,j}(\tau)$, which represents the probability that if i faults have been corrected at time zero, j faults are corrected by time τ after the next failure occurs, is given by

$$Q_{i,j}(\tau) = P_{ij}\left[1 - e^{-(Dk^i+\theta)\tau}\right], \tag{3.3}$$

where P_{ij} are the transition probabilities from state i to state j and given by

$$P_{ij} = \begin{cases} q & (j = i) \\ p & (j = i+1) \\ 0 & (\text{otherwise}) \end{cases} \qquad (i,\ j = 0,\ 1,\ 2,\ \ldots). \tag{3.4}$$

3.3 Derivation of Software Reliability Measures

Using the similar procedures to the basic model, we can derive several quantitative measures for software reliability measurement.

3.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Recall that S_n denotes the random variable representing the time spent in removing n faults, which is defined in Chapter 2. Then, the distribution function of S_n is given by

$$\begin{aligned} G_n(t) &\equiv \Pr\{S_n \le t\} \\ &= \sum_{i=0}^{n-1} A_i^n \left[1 - e^{-p(Dk^i+\theta)t}\right] \quad (t \ge 0;\ n = 1,\ 2,\ \ldots), \end{aligned} \tag{3.5}$$

where

$$\left.\begin{aligned} A_0^1 &\equiv 1 \\ A_i^n &= \prod_{\substack{j=0 \\ j\neq i}}^{n-1} \frac{k^j + \theta/D}{k^j - k^i} \quad (n = 2,\ 3,\ \ldots;\ i = 0,\ 1,\ 2,\ \ldots,\ n-1) \end{aligned}\right\}. \tag{3.6}$$

Furthermore, the mean and the variance of S_n are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1} \frac{1}{p(Dk^i + \theta)}, \tag{3.7}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1} \frac{1}{p^2(Dk^i + \theta)^2}, \tag{3.8}$$

respectively.

3.3.2 Distribution of the Number of Faults Corrected Up to a Specified Time

Recall that $X(t)$ denotes a counting process representing the cumulative number of faults corrected up to testing time t, which is defined in Chapter 2. Using (3.5), we can obtain the expectation and the variance of $X(t)$ as

$$\mathrm{E}[X(t)] = \sum_{n=1}^{\infty} G_n(t), \tag{3.9}$$

$$\mathrm{Var}[X(t)] = \sum_{n=1}^{\infty}(2n-1)G_n(t) - \left[\sum_{n=1}^{\infty} G_n(t)\right]^2, \tag{3.10}$$

respectively.

3.3.3 Expected Number of Software Failures

The expectation of the number of software failures up to testing time t are given by

$$\begin{aligned} M(t) &= \frac{1}{p}\sum_{n=1}^{\infty} G_n(t) \\ &= \frac{1}{p}\mathrm{E}[X(t)], \end{aligned} \tag{3.11}$$

where (3.9) is used as $\mathrm{E}[X(t)]$.

3.3.4 Distribution of the Time between Software Failures

Recall that X_l ($l = 1,\ 2,\ \ldots$) denotes the random variable representing the time interval between the $(l-1)$-st and the l-th software failure occurrences. Then, the reliability function and the hazard rate of X_l are given by

$$\begin{aligned} R_l(x) &\equiv \Pr\{X_l > x\} \\ &= \sum_{i=0}^{l-1} \binom{l-1}{i} p^i q^{l-1-i} e^{-(Dk^i+\theta)x}, \end{aligned} \tag{3.12}$$

$$\lambda_l(x) \equiv \frac{-\dfrac{d}{dx}R_l(x)}{R_l(x)}$$

$$= \frac{\sum_{i=0}^{l-1} \binom{l-1}{i} p^i q^{l-1-i} (Dk^i + \theta) e^{-(Dk^i+\theta)x}}{\sum_{i=0}^{l-1} \binom{l-1}{i} p^i q^{l-1-i} e^{-(Dk^i+\theta)x}}, \tag{3.13}$$

respectively. Furthermore, the expectation of X_l, i.e. MTBSF, is obtained as

$$\mathrm{E}[X_l] = \sum_{i=0}^{l-1} \binom{l-1}{i} \frac{p^i q^{l-1-i}}{Dk^i + \theta}. \tag{3.14}$$

3.4 Parameter Estimation

In this section, we discuss the estimation method of the unknown parameters D and k.

The hazard rate for X_l can be regarded as a random variable depending on the cumulative number of the corrected faults. Letting H_l ($l = 1, 2, \ldots$) be the random variable representing the hazard rate for X_l, we have

$$\Pr\{C_l = i\} = \Pr\{H_l = Dk^i + \theta\} = \binom{l-1}{i} p^i q^{l-1-i}$$
$$(i = 0, 1, 2, \ldots, l-1). \tag{3.15}$$

Then, the expectation of H_l is given by

$$\mathrm{E}[H_l] = \sum_{i=0}^{l-1} (Dk^i + \theta) \cdot \binom{l-1}{i} p^i q^{l-1-i}$$
$$= D(pk+q)^{l-1} + \theta. \tag{3.16}$$

Accordingly, using (3.16), we approximate the hazard rate for X_l expressed by (3.13). Then, the probability density function for X_l is approximated by

$$\phi_l(x) = [D(pk+q)^{l-1} + \theta] e^{-[D(pk+q)^{l-1}+\theta]x}. \tag{3.17}$$

Suppose that the data set on m software failure-occurrence time-intervals x_l, i.e. the realization of X_l ($l = 1, 2, \ldots, m$) is observed during the testing phase. The simultaneous probability density function, i.e. the likelihood function, is given by

$$L = \prod_{l=1}^{m} \phi_l(x_l)$$
$$= \prod_{l=1}^{m} [D(pk+q)^{l-1} + \theta] \cdot \exp\left[-\sum_{l=1}^{m} [D(pk+q)^{l-1} + \theta] x_l \right]. \tag{3.18}$$

Taking the natural logarithm of (3.18) yields

$$\ln L = \sum_{l=1}^{m} \{\ln[D(pk+q)^{l-1} + \theta] - [D(pk+q)^{l-1} + \theta] x_l\}. \tag{3.19}$$

The maximum-likelihood estimates $\widehat{D}$ and $\widehat{k}$ for the unknown parameters D and k can be obtained by solving the simultaneous likelihood equations $\partial \ln L/\partial D = \partial \ln L/\partial k = 0$, i.e.

$$\sum_{l=1}^{m} \left[\frac{(pk+q)^{l-1}}{D(pk+q)^{l-1} + \theta} - (pk+q)^{l-1} x_l \right] = 0, \tag{3.20}$$

$$\sum_{l=2}^{m} \left[\frac{(l-1)(pk+q)^{l-2}}{D(pk+q)^{l-1} + \theta} - (l-1)(pk+q)^{l-2} x_l \right] = 0, \tag{3.21}$$

which can be solved numerically.

3.5 Numerical Examples

Applying the extended software reliability model discussed above to the actual testing-data, we show several numerical illustrations of software reliability measurement and assessment.

The data set consists of 26 software failure-occurrence time-interval data x_l (days; $l = 1, 2, \ldots, 26$) cited by Goel and Okumoto [11]. The testing termination time is 250 days, i.e. $\sum_{l=1}^{26} x_l = 250$.

For these data, we assume that model parameters p and θ can be prespecified. Let $d = \theta T$ be the expected number of F2 in the testing time-interval $(0, T]$. Assuming that 10% of the cumulative number of software failures at the testing termination time $T = 250$ are F2, we can determine that $\theta = 0.0104$. In case of $p = 0.9$, the maximum-likelihood estimates of unknown parameters D and k are estimated as

$$\widehat{D} = 0.189, \quad \widehat{k} = 0.947 \qquad (\theta = 0.0104,\ p = 0.9),$$

respectively. The estimated expected numbers of software failures $\widehat{M(t)}$ in (3.11) and corrected faults $\mathrm{E}[\widehat{X(t)}]$ in (3.9) are shown in Fig. 3.2.

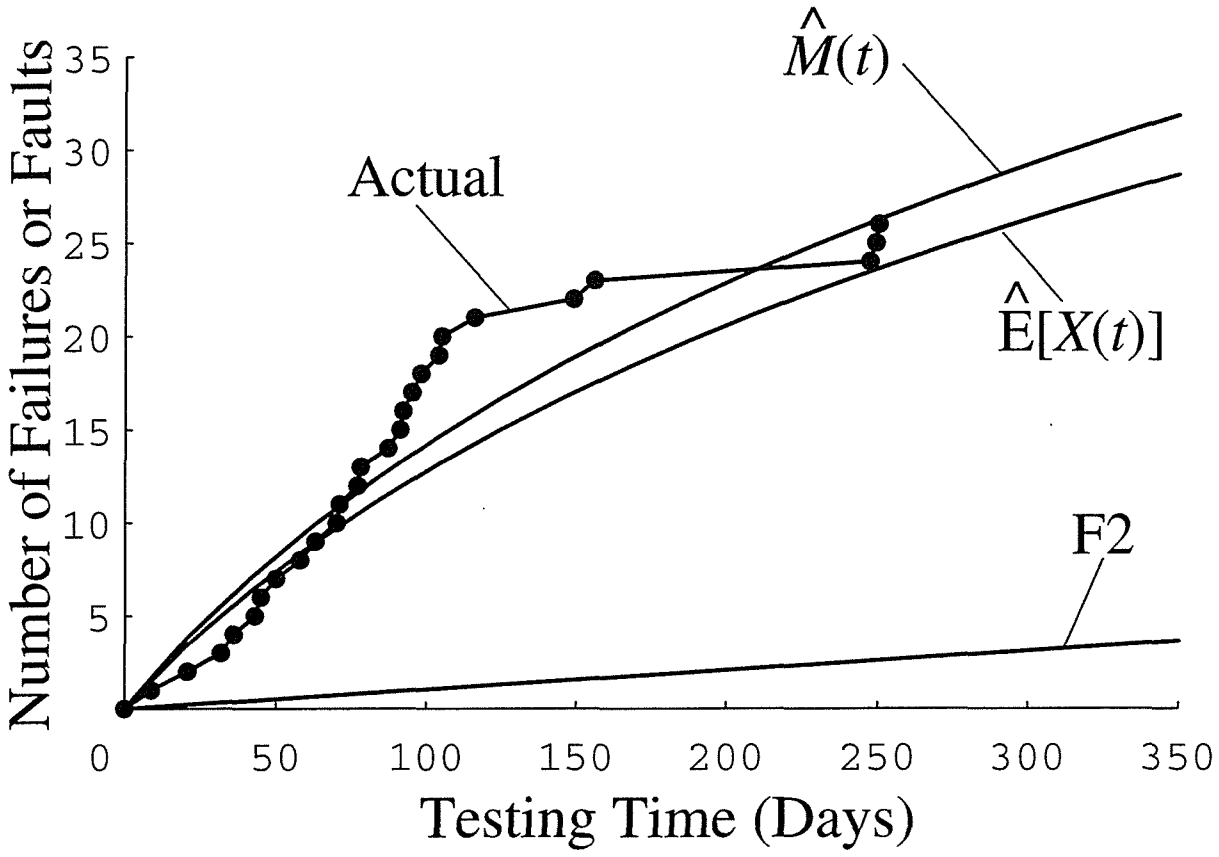

Fig. 3.2. Estimated $\widehat{M(t)}$ and $\mathrm{E}[\widehat{X(t)}]$.

The estimated $\widehat{G_{26}(t)}$ in (3.5), which represents the distribution function of the time spent in correcting 26 faults, are shown in Fig. 3.3. Though 26 software failures are observed at the testing termination time, the probability that 26 faults are corrected is estimated as 0.266. Letting the objective of this probability be 0.9, we can estimate that about 120-day testing time must be added.

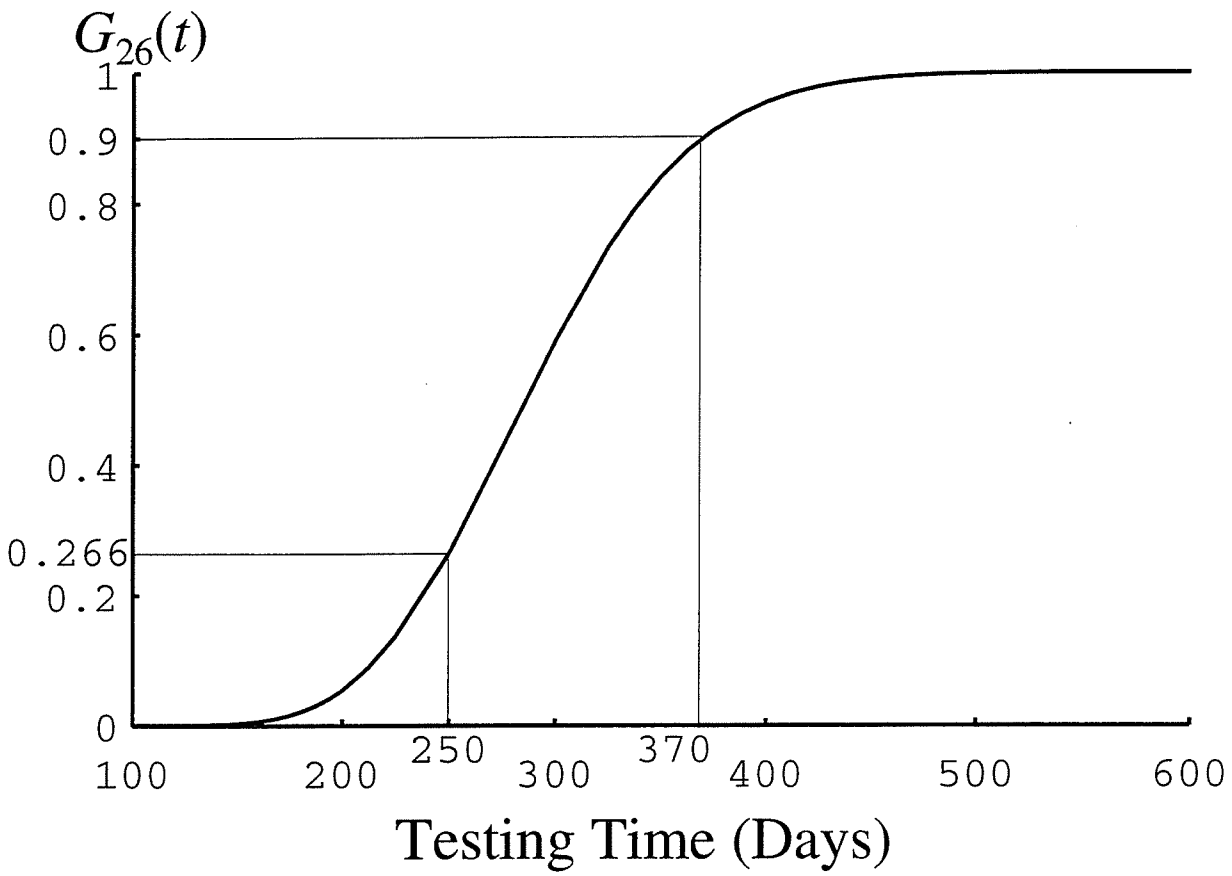

Fig. 3.3. Estimated $\widehat{G_{26}(t)}$.

The coefficient of variation (cv) of $X(t)$, cv$[X(t)]$, is defined as

$$\text{cv}[X(t)] \equiv \frac{\sqrt{\text{Var}[X(t)]}}{\text{E}[X(t)]}. \tag{3.22}$$

Figure 3.4 shows the estimated $\widehat{\text{cv}[X(t)]}$. As shown in Fig. 3.4, cv$[X(t)]$ is a monotone decreasing function with respect to testing time t. cv$[X(t)]$ is can be regarded as a measure of stability of the fault correction. The cv at the testing termination time is estimated as 0.131. Letting the objective cv be 0.1, we can estimate that about 200-day testing time must be added.

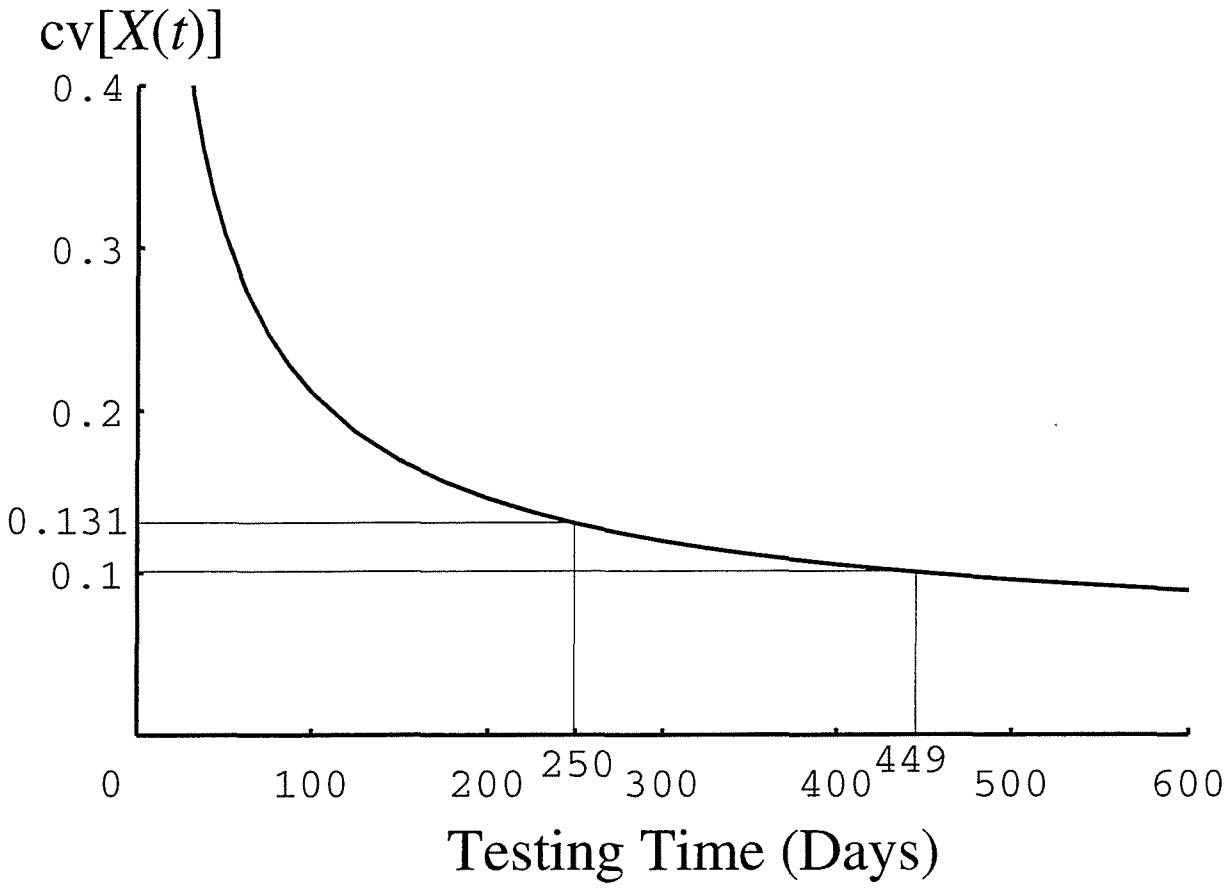

Fig. 3.4. Estimated $\widehat{\mathrm{cv}[X(t)]}$.

The estimated reliability function $\widehat{R_{27}(x)}$ in (3.12) for the next software failure-occurrence time-interval X_{27} is shown in Fig. 3.5 along with that in the case where F2 is not considered, i.e. $\theta = 0$. The value of MTBSF for various l's are shown in Table 3.1. These figure and table indicate that the consideration of F2 lowers software reliability.

The estimated hazard rates $\widehat{\lambda_l(x)}$ expressed by (3.13) for various l's are shown in Fig. 3.6. From (3.13) and (3.16), it is noted that $\lambda_l(0) = \mathrm{E}[H_l]$ for arbitrary natural number l. Since this figure indicates that (3.13) is nearly constant with respect to x, the approximation of the hazard rate $\lambda_l(x)$ by using (3.16) is valid.

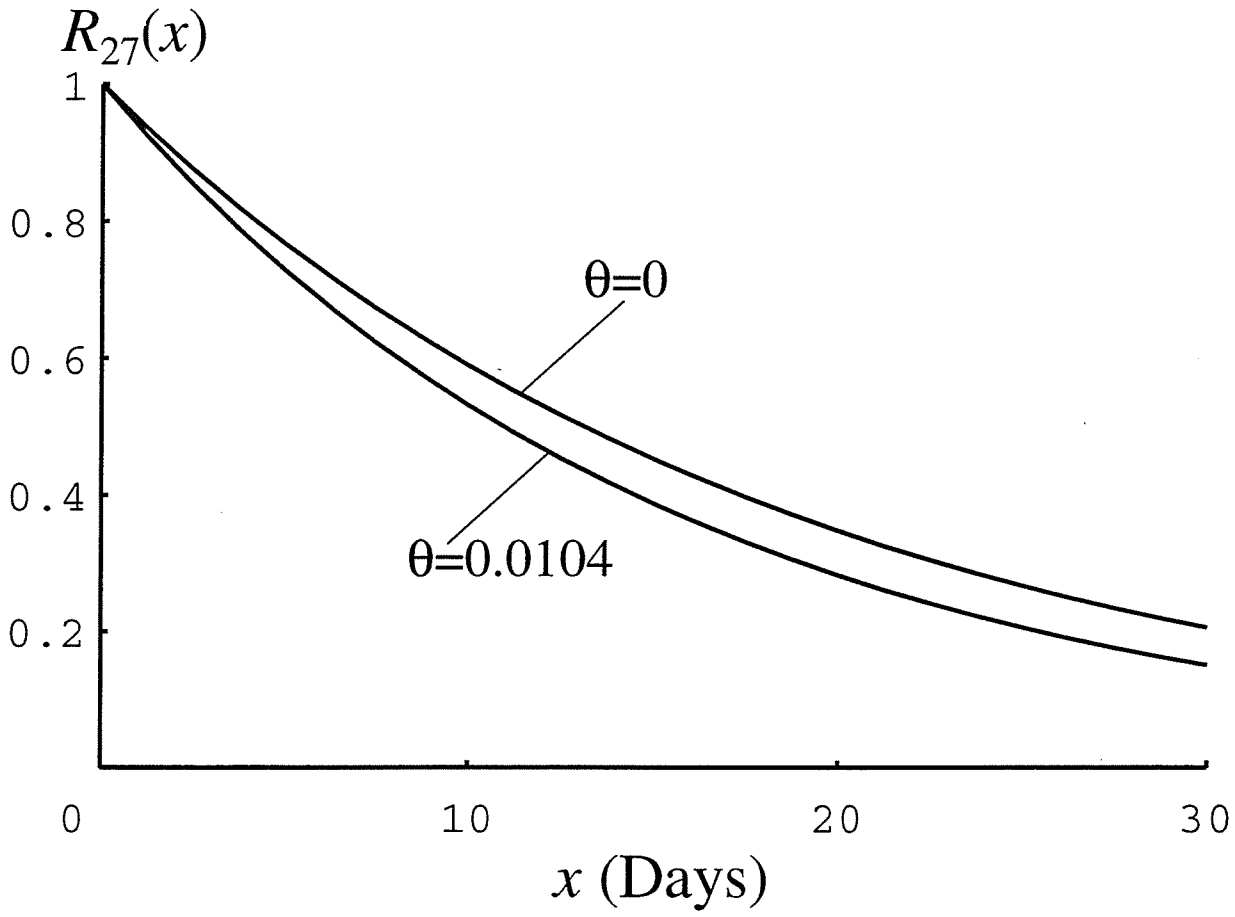

Fig. 3.5. Estimated $\widehat{R_{27}(x)}$.

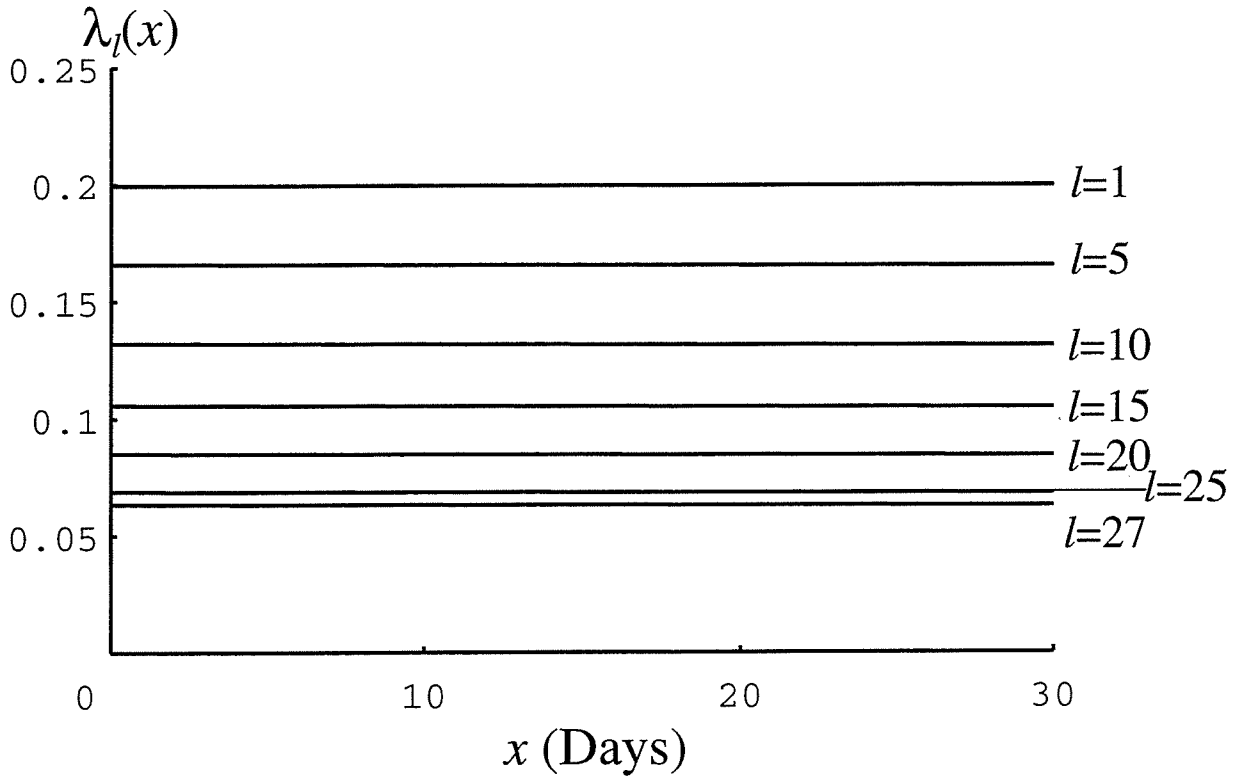

Fig. 3.6. Estimated $\widehat{\lambda_l(x)}$.

Table 3.1. MTBSF $\mathrm{E}[X_l]$.

l	θ	
	0	0.0104
27	18.99	15.84
28	19.94	16.50
29	20.95	17.18
30	22.00	17.88

3.6 Concluding Remarks

In this chapter, we have developed a software reliability growth model considering the software failure-occurrence due to the faults introduced during debugging of the testing phase. This model can consider the imperfect debugging environment in which both fault removal is uncertain and debugging may introduce other new faults. Accordingly, this imperfect debugging model is superior to the earlier one discussed in Chapter 2. Several quantities for software reliability measurement have been derived from this model and the numerical examples of software reliability analysis based on the actual testing-data have been illustrated.

Part II

SOFTWARE AVAILABILITY MODELING

Chapter 4

Software Availability Modeling

4.1 Introduction

It is often reported that unexpected system accidents have occurred due to defects or faults remaining in software systems. Since today's social life depends on computer systems so much, their breakdowns do great damage. Furthermore, they may give rise to serious accidents menacing human life [43, 57]. Under the background like this, the software production technologies for the purpose of producing quality software systems efficiently, systematically, and economically have been developed and researched energetically.

The software failure-occurrence process is primarily systematic since software systems are the sets of logic. But it is impossible to recognize all of input domain in advance. Furthermore, even if we were to recognize input domain, we could not specify which data are inputted in execution [24]. In addition, software faults, which are the causes of software failures, are more latent and more difficult to be isolated than hardware defects. Accordingly, the description of the failure-occurrence phenomenon for software systems needs the stochastic approach as well as for hardware systems. Many mathematical models for the purpose of reasonable software reliability measurement and assessment have been proposed and applied to practical use since it is considered that software reliability is the main quality characteristic in software products. A mathematical software reliability model is often called a software reliability growth model which describes a software fault-detection or a software failure-occurrence phenomenon during the testing phase of the software development process and the operation phase [35, 56]. A software failure is defined as an unacceptable departure from program operation caused by a fault remaining in the software system. This model is utilized for measuring and assessing the degree of achievement of software reliability, deciding the time to software release for operational

use, and estimating the maintenance cost for faults undetected during the testing phase.

Software development managers and customers have taken a growing interest in the software performance measure such as the possible utilization factor as well as in the hardware product. In particular, for telecommunication software systems, it is one of the software quality characteristics to be considered. Accordingly, it is very important to measure and assess software availability, which is defined as the probability that the software system is performing successfully, according to the specifications, at a specified time point [50]. However, few quantitative models for measuring software availability are proposed.

It is often assumed that the hardware systems are renewed by repair or replacement of failed parts. Then, the hardware failure-characteristic is described without concerning with the number of failures. On the other hand, when software failures occur, debugging activities to remove faults are performed and software reliability improves. Accordingly, it is necessary to consider the software reliability-growth process in software availability modeling.

In this chapter, we discuss software availability modeling [16, 19, 21, 39]. It is assumed that the mean times to the software failure-occurrence and the restoration times become longer geometrically with the progress in fault correction. This assumption reflects that the fault complexity becomes higher as the debugging activity proceeds [35, 36]. Furthermore, we consider the assumption of imperfect debugging. That is, it is assumed that all detected faults are not corrected and removed certainly. Aiming at the cumulative number of faults corrected perfectly, we describe the time-dependent behavior of the system, which alternates between the operational state (up state) that a system is operating regularly and the restoration state (down state) that a system is inoperable, with a Markov process [47]. Several stochastic quantities for software availability measurement in dynamic environment are derived from this model. Estimation of unknown parameters in this model is also discussed. Finally, numerical illustrations for software availability measurement and assessment based on this model are presented.

4.2 Modeling and Assumptions

The following assumptions are made for software availability modeling:

A1. The software system is unavailable and starts to be restored as soon as a software failure occurs, and the system can not operate until the restoration action is complete.

A2. The restoration action implies the debugging activity, which is performed perfectly with probability a $(0 < a \leq 1)$ and imperfectly with probability $b(= 1 - a)$. We call a the perfect debugging rate. One fault is corrected and removed from the software system when the debugging activity is perfect.

A3. The hazard rate is constant between software failures caused by faults in the software system, and geometrically decreases whenever each fault is corrected perfectly [31].

A4. The next restoration time when n faults have been already corrected from the system, follows an exponential distribution with mean $1/\mu_n$.

A5. The probability that two or more software failures occur simultaneously is negligible.

Consider a stochastic process $\{X(t),\ t \geq 0\}$ with the state space $(\boldsymbol{W},\ \boldsymbol{R})$ where up state vector $\boldsymbol{W} = \{W_n;\ n = 0,\ 1,\ 2,\ \ldots\}$ and down state vector $\boldsymbol{R} = \{R_n;\ n = 0,\ 1,\ 2,\ \ldots\}$ [42, 47]. Then, the events $\{X(t) = W_n\}$ and $\{X(t) = R_n\}$ mean that the system is operating and inoperable due to the restoration action at time point t, when n faults have been already corrected, respectively.

From assumption A2, when the restoration action has been complete in $\{X(t) = R_n\}$,

$$X(t) = \begin{cases} W_n & \text{(with probability } b\text{)} \\ W_{n+1} & \text{(with probability } a\text{).} \end{cases} \tag{4.1}$$

Furthermore, from assumption A3, when n faults have been corrected, the hazard rate for the next software failure-occurrence is given by

$$\lambda_n = Dk^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{4.2}$$

where D and k are the initial hazard rate and the decreasing ratio of the hazard rate, respectively. The expression of (4.2) comes from the point of view that software reliability

depends on the debugging efforts, not the residual fault content. We do not note how many faults remain in the software system. Equation (4.2) describes a software failure-occurrence phenomenon where a software system has high frequency of software failure-occurrence during the early stage of the testing or the operation phase and it gradually decreases after then [35]. Early software availability models such as those of Kim et al. [16] and Okumoto & Goel [39] often assume that the hazard rate is proportional to the residual fault content and decreases by a constant amount with the perfect debugging.

Next, we describe the time-dependent behavior of the restoration action. The restoration action for software systems includes not only the data recovery and the program reload but also the debugging activities for manifested faults. From the viewpoint of the fault complexity, there are cases where the faults detected during the early stage of the testing or the operation phase have low complexity and are easy to correct/remove, and as the testing is in progress, detected faults have higher complexity and are more difficult to correct/remove [35, 36]. In the above case, it is appropriate that the mean restoration-time becomes longer with increasing the cumulative number of corrected faults. Accordingly, we express μ_n as follows:

$$\mu_n = Er^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ E > 0,\ 0 < r \leq 1), \tag{4.3}$$

where E and r are the initial restoration rate and the decreasing ratio of the restoration rate, respectively. In case of $r = 1$, i.e. $\mu_n = E$ means that the complexity of each fault is random.

Let T_n and U_n $(n = 0,\ 1,\ 2,\ \ldots)$ be the random variable representing the next software failure-occurrence and the next restoration time-interval when n faults have been corrected, respectively. Furthermore, let $Y(t)$ be the random variable representing the cumulative number of corrected faults up to time t. The sample behavior of $Y(t)$ is illustrated in Fig. 4.1. It is noted that the cumulative number of corrected faults is not always coincident with that of software failures or restoration actions.

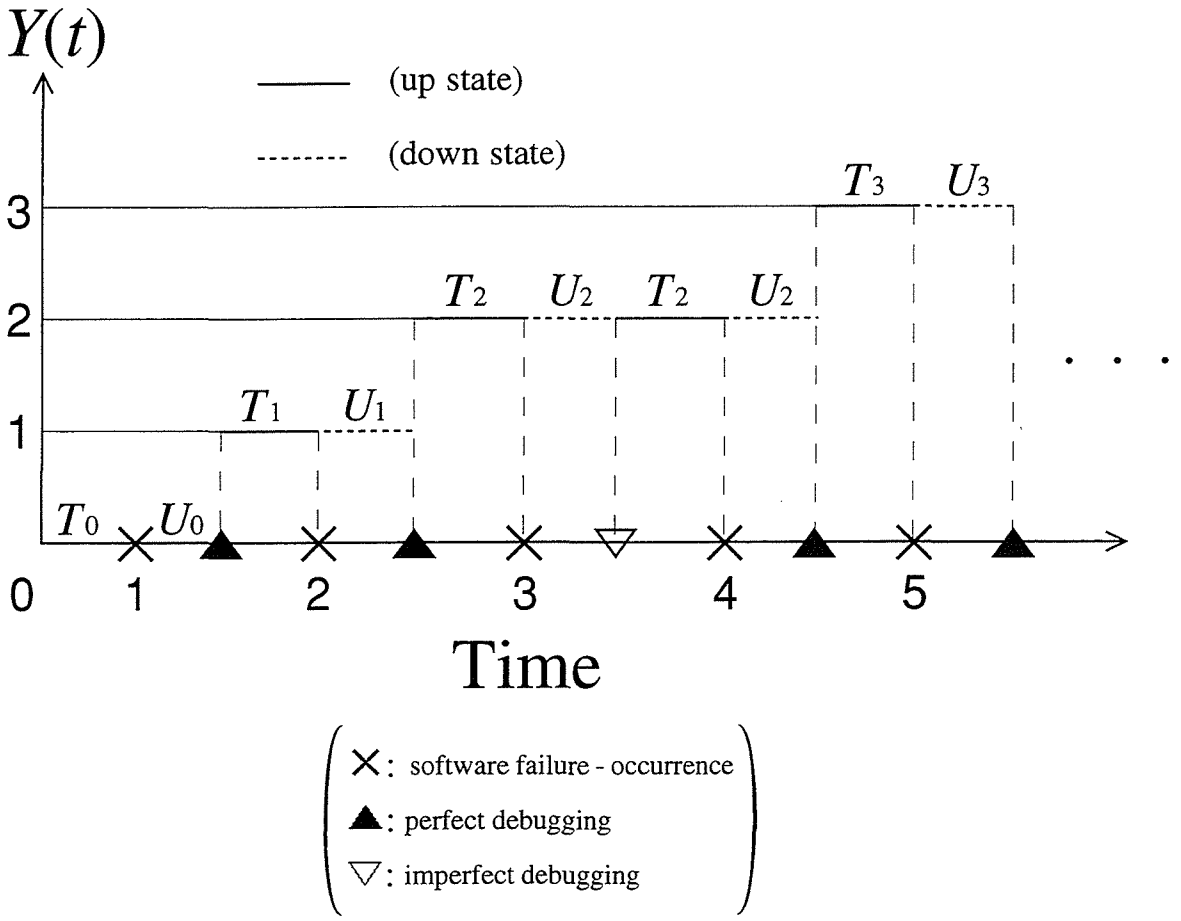

Fig. 4.1. A sample realization of $Y(t)$.

Let $Q_{A,B}(\tau)$ ($A,\ B \in (\boldsymbol{W},\ \boldsymbol{R})$) denote the one-step transition probability that after making a transition into state A, the process $\{X(t),\ t \geq 0\}$ makes a transition into state B by time τ. The expressions for $Q_{A,B}(\tau)$'s are given as follows:

$$Q_{W_n,R_n}(\tau) = 1 - e^{-\lambda_n \tau}, \tag{4.4}$$

$$Q_{R_n,W_{n+1}}(\tau) = a(1 - e^{-\mu_n \tau}), \tag{4.5}$$

$$Q_{R_n,W_n}(\tau) = b(1 - e^{-\mu_n \tau}). \tag{4.6}$$

The sample state transition diagram of $X(t)$ is illustrated in Fig. 4.2.

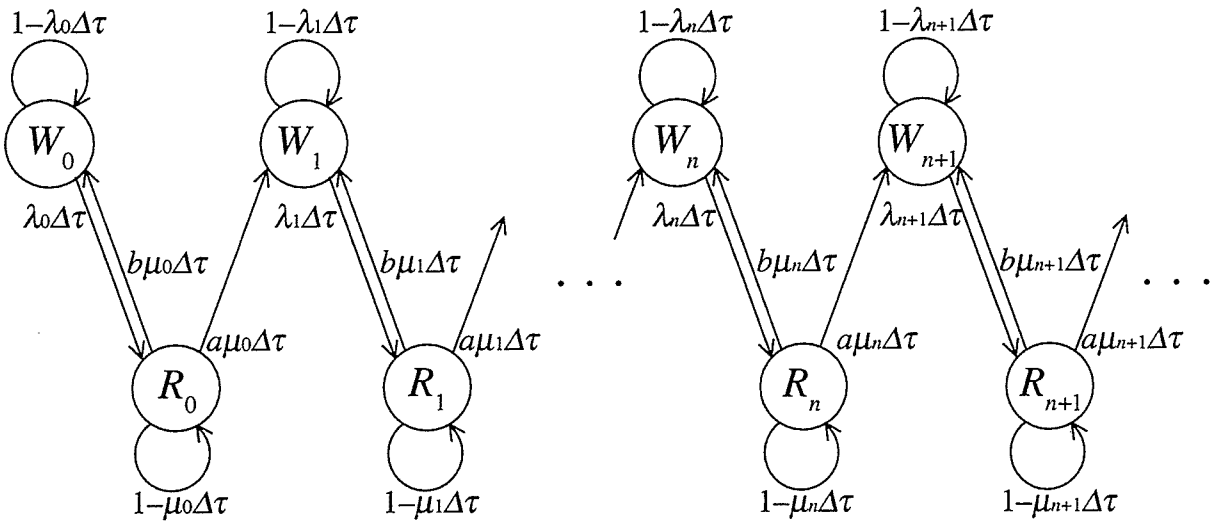

Fig. 4.2. A diagrammatic representation of state transitions between $X(t)$'s for software availability modeling.

4.3 Derivation of Software Performance Measures

4.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Let S_n $(n = 1, 2, \ldots; S_0 \equiv 0)$ be the random variable representing the time spent in correcting n faults and $G_n(t)$ be the distribution function of S_n,respectively. Furthermore, let $G_{i,n}(t)$ be the distribution function associated with the probability that n faults are corrected in the time interval $(0, t]$ on the condition that i faults have been already corrected at time zero. Then, we obtain the following renewal equation with respect to $G_{i,n}(t)$:

$$G_{i,n}(t) = Q_{W_i,R_i} * Q_{R_i,W_{i+1}} * G_{i+1,n}(t) + Q_{W_i,R_i} * Q_{R_i,W_i} * G_{i,n}(t)$$
$$(i = 0, 1, 2, \ldots, n-1), \tag{4.7}$$

where $*$ denotes a Stieltjes convolution and $G_{n,n}(t) = 1$ $(n = 1, 2, \ldots)$. Then, the Laplace-Stieltjes (L-S) transform of (4.7) is obtained as follows (see Section 2.3):

$$\tilde{G}_{i,n}(s) = \tilde{Q}_{W_i,R_i}(s)\tilde{Q}_{R_i,W_{i+1}}(s)\tilde{G}_{i+1,n}(s) + \tilde{Q}_{W_i,R_i}(s)\tilde{Q}_{R_i,W_i}(s)\tilde{G}_{i,n}(s)$$
$$(i = 0, 1, 2, \ldots, n-1). \tag{4.8}$$

From (4.4)–(4.6), the L-S transforms of $Q_{A,B}(t)$'s are respectively given as

$$\tilde{Q}_{W_i,R_i}(s) = \frac{\lambda_i}{s+\lambda_i}, \tag{4.9}$$

$$\tilde{Q}_{R_i,W_{i+1}}(s) = \frac{a\mu_i}{s+\mu_i}, \tag{4.10}$$

$$\tilde{Q}_{R_i,W_i}(s) = \frac{b\mu_i}{s+\mu_i}. \tag{4.11}$$

Substituting (4.9)–(4.11) into (4.8) yields

$$\begin{aligned}\tilde{G}_{i,n}(s) &= \frac{a\lambda_i\mu_i}{s^2+(\lambda_i+\mu_i)s+a\lambda_i\mu_i}\tilde{G}_{i+1,n}(s) \\ &= \frac{x_i y_i}{(s+x_i)(s+y_i)}\tilde{G}_{i+1,n}(s) \quad (i=0,\ 1,\ 2,\ \ldots,\ n-1),\end{aligned} \tag{4.12}$$

where

$$\left.\begin{matrix} x_i \\ y_i \end{matrix}\right\} = \frac{1}{2}\left[(\lambda_i+\mu_i) \pm \sqrt{(\lambda_i+\mu_i)^2 - 4a\lambda_i\mu_i}\right]$$

(double signs in same order). (4.13)

Solving (4.12) recursively, we obtain $\tilde{G}_n(s)$ as

$$\begin{aligned}\tilde{G}_n(s) &= \prod_{i=0}^{n-1}\frac{x_i y_i}{(s+x_i)(s+y_i)} \\ &= \sum_{i=0}^{n-1}\left(\frac{A^1_{n,i}x_i}{s+x_i} + \frac{A^2_{n,i}y_i}{s+y_i}\right),\end{aligned} \tag{4.14}$$

where constant coefficients $A^1_{n,i}$ and $A^2_{n,i}$ are given by

$$A^1_{n,i} = \frac{\prod_{j=0}^{n-1} x_j y_j}{x_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(x_j - x_i)\prod_{j=0}^{n-1}(y_j - x_i)} \quad (i=0,\ 1,\ 2,\ \ldots,\ n-1), \tag{4.15}$$

$$A^2_{n,i} = \frac{\prod_{j=0}^{n-1} x_j y_j}{y_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(y_j - y_i)\prod_{j=0}^{n-1}(x_j - y_i)} \quad (i=0,\ 1,\ 2,\ \ldots,\ n-1), \tag{4.16}$$

respectively. By inverting (4.14), we have the distribution function of the first passage time when n faults are corrected

$$\begin{aligned} G_n(t) &\equiv \Pr\{S_n \leq t\} \\ &= 1 - \sum_{i=0}^{n-1}(A_{n,i}^1 e^{-x_i t} + A_{n,i}^2 e^{-y_i t}) \quad (n = 1,\ 2,\ \ldots), \end{aligned} \tag{4.17}$$

where we postulate $\prod_{\substack{j=0 \\ j\neq 0}}^{0} \cdot = 1$. It is noted that

$$\sum_{i=0}^{n-1}(A_{n,i}^1 + A_{n,i}^2) = 1, \tag{4.18}$$

for arbitrary $n \geq 1$.

Furthermore, $\mathrm{E}[S_n]$ and $\mathrm{Var}[S_n]$ are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1}\left(\frac{1}{x_i} + \frac{1}{y_i}\right), \tag{4.19}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1}\left(\frac{1}{{x_i}^2} + \frac{1}{{y_i}^2}\right), \tag{4.20}$$

respectively.

4.3.2 State Occupancy Probability and Software Availability

Let $P_{A,B}(t)$ $(A,\ B \in (\boldsymbol{W},\ \boldsymbol{R}))$ be the conditional state occupancy probability that the system is in state B at time point t on the condition that the system was in state A at time point zero, i.e.

$$\begin{aligned} P_{A,B}(t) &\equiv \Pr\{X(t) = B | X(0) = A\} \\ &\qquad (A,\ B \in (\boldsymbol{W},\ \boldsymbol{R})), \end{aligned} \tag{4.21}$$

and $P_{W_n}(t) \equiv P_{W_0,W_n}(t)$ and $P_{R_n}(t) \equiv P_{W_0,R_n}(t)$ are called the operational and the restoration state occupancy probability, respectively.

We obtain the following renewal equations with respect to $P_{W_n}(t)$ and $P_{W_n,W_n}(t)$:

$$P_{W_n}(t) = G_n * P_{W_n,W_n}(t), \tag{4.22}$$

$$P_{W_n,W_n}(t) = e^{-\lambda_n t} + Q_{W_n,R_n} * Q_{R_n,W_n} * P_{W_n,W_n}(t). \tag{4.23}$$

From (4.23), the L-S transform of $P_{W_n,W_n}(t)$ is given by

$$\begin{aligned}\widetilde{P}_{W_n,W_n}(s) &= \frac{s(s+\mu_n)}{(s+x_n)(s+y_n)} \\ &= \left(\frac{s}{a\lambda_n} + \frac{s^2}{a\lambda_n\mu_n}\right)\frac{x_n y_n}{(s+x_n)(s+y_n)}. \end{aligned} \tag{4.24}$$

Substituting (4.24) into the L-S transform of (4.22) yields

$$\widetilde{P}_{W_n}(s) = \frac{s}{a\lambda_n}\widetilde{G}_{n+1}(s) + \frac{s^2}{a\lambda_n\mu_n}\widetilde{G}_{n+1}(s). \tag{4.25}$$

By inverting (4.25), $P_{W_n}(t)$ is obtained as

$$\begin{aligned}P_{W_n}(t) &\equiv \Pr\{X(t) = W_n\} \\ &= \frac{1}{a\lambda_n}g_{n+1}(t) + \frac{1}{a\lambda_n\mu_n}g'_{n+1}(t), \end{aligned} \tag{4.26}$$

where $g_n(t)$ is the probability density function associated with $G_n(t)$ and $g'_n(t) \equiv dg_n(t)/dt$.

Using the similar procedure for the derivation of $P_{W_n}(t)$, we obtain the following renewal equations with respect to $P_{R_n}(t)$ and $P_{R_n,R_n}(t)$:

$$P_{R_n}(t) = G_n * Q_{W_n,R_n} * P_{R_n,R_n}(t), \tag{4.27}$$

$$P_{R_n,R_n}(t) = e^{-\mu_n t} + Q_{R_n,W_n} * Q_{W_n,R_n} * P_{R_n,R_n}(t). \tag{4.28}$$

From (4.28), the L-S transform of $P_{R_n,R_n}(t)$ is given by

$$\widetilde{P}_{R_n,R_n}(s) = \frac{s(s+\lambda_n)}{(s+x_n)(s+y_n)}. \tag{4.29}$$

Substituting (4.29) into the L-S transform of (4.27) yields

$$\begin{aligned}\widetilde{P}_{R_n}(s) &= \frac{\lambda_n}{(s+\lambda_n)} \cdot \frac{s(s+\lambda_n)}{(s+x_n)(s+y_n)}\widetilde{G}_n(s) \\ &= \frac{s}{a\mu_n}\widetilde{G}_{n+1}(s). \end{aligned} \tag{4.30}$$

By inverting (4.30), $P_{R_n}(t)$ is obtained as

$$\begin{aligned}P_{R_n}(t) &\equiv \Pr\{X(t) = R_n\} \\ &= \frac{1}{a\mu_n}g_{n+1}(t). \end{aligned} \tag{4.31}$$

The following equation holds for arbitrary time t:

$$\sum_{n=0}^{\infty}[P_{W_n}(t) + P_{R_n}(t)] = 1. \tag{4.32}$$

The instantaneous software availability [50] is defined as

$$A(t) \equiv \sum_{n=0}^{\infty} P_{W_n}(t), \tag{4.33}$$

which represents the probability that the software system is operating at specified time point t. Furthermore, the average software availability over $(0,\ t]$ [41] is defined as

$$A_{av}(t) \equiv \frac{1}{t}\int_0^t A(x)dx, \tag{4.34}$$

which represents the ratio of system's operating time to the time-interval $(0,\ t]$. Using (4.26) and (4.31), we can express (4.33) and (4.34) as

$$\begin{aligned} A(t) &= \sum_{n=0}^{\infty}\left[\frac{1}{a\lambda_n}g_{n+1}(t) + \frac{1}{a\lambda_n\mu_n}g'_{n+1}(t)\right] \\ &= 1 - \sum_{n=0}^{\infty}\frac{1}{a\mu_n}g_{n+1}(t), \end{aligned} \tag{4.35}$$

$$\begin{aligned} A_{av}(t) &= \frac{1}{t}\sum_{n=0}^{\infty}\left[\frac{1}{a\lambda_n}G_{n+1}(t) + \frac{1}{a\lambda_n\mu_n}g_{n+1}(t)\right] \\ &= 1 - \frac{1}{t}\sum_{n=0}^{\infty}\frac{1}{a\mu_n}G_{n+1}(t), \end{aligned} \tag{4.36}$$

respectively.

4.4 Parameter Estimation

Applying the method of maximum likelihood, we discuss the estimation of the unknown parameters D, k, E, and r.

Let V_l $(l = 1,\ 2,\ \ldots)$ be the random variable representing the time interval between the $(l-1)$-st and the l-th software failure-occurrences. It is noted that V_l is not always same as random variable T_{l-1} (see Fig. 4.1). Then, the hazard rate of V_l can be approximated as follows (see Section 3.4):

$$h_l(x) = D(ak+b)^{l-1}. \tag{4.37}$$

Suppose that m software failure-occurrence time-intervals v_l $(l = 1,\ 2,\ \ldots,\ m)$, i.e. the realization of V_l, are observed. The simultaneous probability density function, i.e. the likelihood function, is given by

$$L = D^m \prod_{l=1}^{m}(ak+b)^{l-1}\exp\left[-D\sum_{l=1}^{m}(ak+b)^{l-1}v_l\right]. \tag{4.38}$$

Taking the natural logarithm of (4.38) yields

$$\ln L = m \ln D + \frac{m(m-1)}{2} \ln(ak+b) - D \sum_{l=1}^{m} (ak+b)^{l-1} v_l. \tag{4.39}$$

The maximum-likelihood estimates $\widehat{D}$ and $\widehat{k}$ for the unknown parameters D and k can be obtained by solving the simultaneous likelihood equations $\partial \ln L/\partial D = \partial \ln L/\partial k = 0$, i.e.

$$D = \frac{m}{\sum_{l=1}^{m} (ak+b)^{l-1} v_l}, \tag{4.40}$$

$$\frac{m-1}{2(ak+b)} = \frac{\sum_{l=2}^{m} (l-1)(ak+b)^{l-2} v_l}{\sum_{l=1}^{m} (ak+b)^{l-1} v_l}, \tag{4.41}$$

which can be solved numerically.

Similarly, we can discuss the estimation of the unknown parameters E and r. Let Z_l ($l = 1, 2, \ldots$) be the random variable representing the restoration time for the l-th software failure-occurrence. Suppose that m restoration times z_l ($l = 1, 2, \ldots, m$) are observed. Then, the maximum-likelihood estimates $\widehat{E}$ and $\widehat{r}$ for the unknown parameters E and r can be obtained by solving the following simultaneous likelihood equations:

$$E = \frac{m}{\sum_{l=1}^{m} (ar+b)^{l-1} z_l}, \tag{4.42}$$

$$\frac{m-1}{2(ar+b)} = \frac{\sum_{l=2}^{m} (l-1)(ar+b)^{l-2} z_l}{\sum_{l=1}^{m} (ar+b)^{l-1} z_l}, \tag{4.43}$$

which can be solved numerically.

4.5 Numerical Examples

We show several application examples of the software availability model discussed above.

We use the data set consisting of 26 software failure-occurrence time-interval data w_l (days; $l = 1, 2, \ldots, 26$) cited by Goel & Okumoto [11]. However, restoration times are not explicitly shown in these data. Therefore, we assume that w_l includes both the software failure and the restoration time, i.e. $w_l = v_l + z_l$ ($l = 1, 2, \ldots, 26$). Accordingly, we generate v_l's and z_l's for respective w_l's, using the following procedure:

Step 1: Generate a random number ξ_l ($0 \leq \xi_l \leq 1$) following the beta distribution $BETA(\alpha, \beta)$ [42] having the following density function:

$$\left.\begin{aligned} f(\xi) &= \frac{\xi^{\alpha-1}(1-\xi)^{\beta-1}}{B(\alpha,\beta)} \\ B(\alpha,\beta) &= \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt \\ (\alpha > 0,\ \beta > 0,\ 0 \leq \xi \leq 1) & \end{aligned}\right\}. \tag{4.44}$$

Step 2: Let $v_l = \xi_l w_l$ and $z_l = (1 - \xi_l)w_l$.

We call the data set generated by $BETA(8, 2)$ DATA 1 and generated by $BETA(5, 5)$ DATA 2, respectively. The generated data sets and the maximum-likelihood estimates $\widehat{D}$, $\widehat{k}$, $\widehat{E}$, and $\widehat{r}$ for various values of a are shown in Tables 4.1 and 4.2, respectively. For example, in case of $a = 0.8$, the maximum-likelihood estimates are obtained as:

$$\widehat{D} = 0.246,\ \widehat{k} = 0.940,\ \widehat{E} = 1.114,\ \widehat{r} = 0.960 \quad \text{(DATA 1: } a = 0.8\text{)},$$
$$\widehat{D} = 0.365,\ \widehat{k} = 0.956,\ \widehat{E} = 0.500,\ \widehat{r} = 0.954 \quad \text{(DATA 2: } a = 0.8\text{)}.$$

Table 4.1. Generated data sets.

	l	1	2	3	4	5	6	7	8	9	10	11	12	13
	w_l	9	12	11	4	7	2	5	8	5	7	1	6	1
DATA1	v_l	6.8	11.1	8.0	3.0	6.0	1.8	3.8	6.6	4.2	6.9	0.8	4.9	0.9
	z_l	2.2	0.9	3.0	1.0	1.0	0.2	1.2	1.4	0.8	0.1	0.2	1.1	0.1
DATA2	v_l	6.1	5.5	7.2	2.1	4.1	0.4	2.3	3.0	2.5	4.2	0.6	3.8	0.4
	z_l	2.9	6.5	3.8	1.9	2.9	1.6	2.7	5.0	2.5	2.8	0.4	2.2	0.6

Table 4.1. (Continued)

	l	14	15	16	17	18	19	20	21	22	23	24	25	26
	w_l	9	4	1	3	3	6	1	11	33	7	91	2	1
DATA1	v_l	8.7	3.5	0.9	2.6	2.9	5.3	0.9	9.3	24.1	5.3	82.8	1.5	0.8
	z_l	0.3	0.5	0.1	0.4	0.1	0.7	0.1	1.7	8.9	1.7	8.2	0.5	0.2
DATA2	v_l	5.2	1.7	0.5	1.9	1.2	3.6	0.5	6.7	18.6	2.6	49.5	1.4	0.4
	z_l	3.8	2.3	0.5	1.1	1.8	2.4	0.5	4.3	14.4	4.4	41.5	0.6	0.6

Table 4.2. Maximum-likelihood estimates.

	DATA 1					DATA 2			
a	$\widehat{D}$	$\widehat{k}$	$\widehat{E}$	$\widehat{r}$	a	$\widehat{D}$	$\widehat{k}$	$\widehat{E}$	$\widehat{r}$
1.0	0.246	0.952	1.114	0.968	1.0	0.365	0.956	0.500	0.954
0.9	0.246	0.947	1.114	0.964	0.9	0.365	0.951	0.500	0.949
0.8	0.246	0.940	1.114	0.960	0.8	0.365	0.945	0.500	0.943
0.7	0.246	0.931	1.114	0.954	0.7	0.365	0.937	0.500	0.934
0.6	0.246	0.920	1.114	0.947	0.6	0.365	0.927	0.500	0.923
0.5	0.246	0.904	1.114	0.936	0.5	0.365	0.912	0.500	0.908
0.4	0.246	0.880	1.114	0.920	0.4	0.365	0.890	0.500	0.885
0.3	0.246	0.840	1.114	0.893	0.3	0.365	0.853	0.500	0.847
0.2	0.246	0.760	1.114	0.840	0.2	0.365	0.780	0.500	0.770
0.1	0.246	0.520	1.114	0.680	0.1	0.365	0.560	0.500	0.540

Figure 4.3 shows the estimated $\widehat{G_{26}(t)}$'s in (4.17) in case of $a = 0.8$, which represent the distribution functions of the time spent in correcting 26 faults where 26 software failures are observed at the testing termination time, which is 250 days. Letting the objective of the probability that 26 faults are corrected be 0.9, we can estimate that 213-day testing time for DATA 1 and 172-day testing time for DATA 2 must be added. Furthermore, we can estimate $\mathrm{E}[\widehat{S_{26}}] = 369.1$ (days) for DATA 1 and $\mathrm{E}[\widehat{S_{26}}] = 346.2$ (days) for DATA 2, respectively. Therefore, we can see that the case of DATA 2 is more stable in fault correctability than the case of DATA 1.

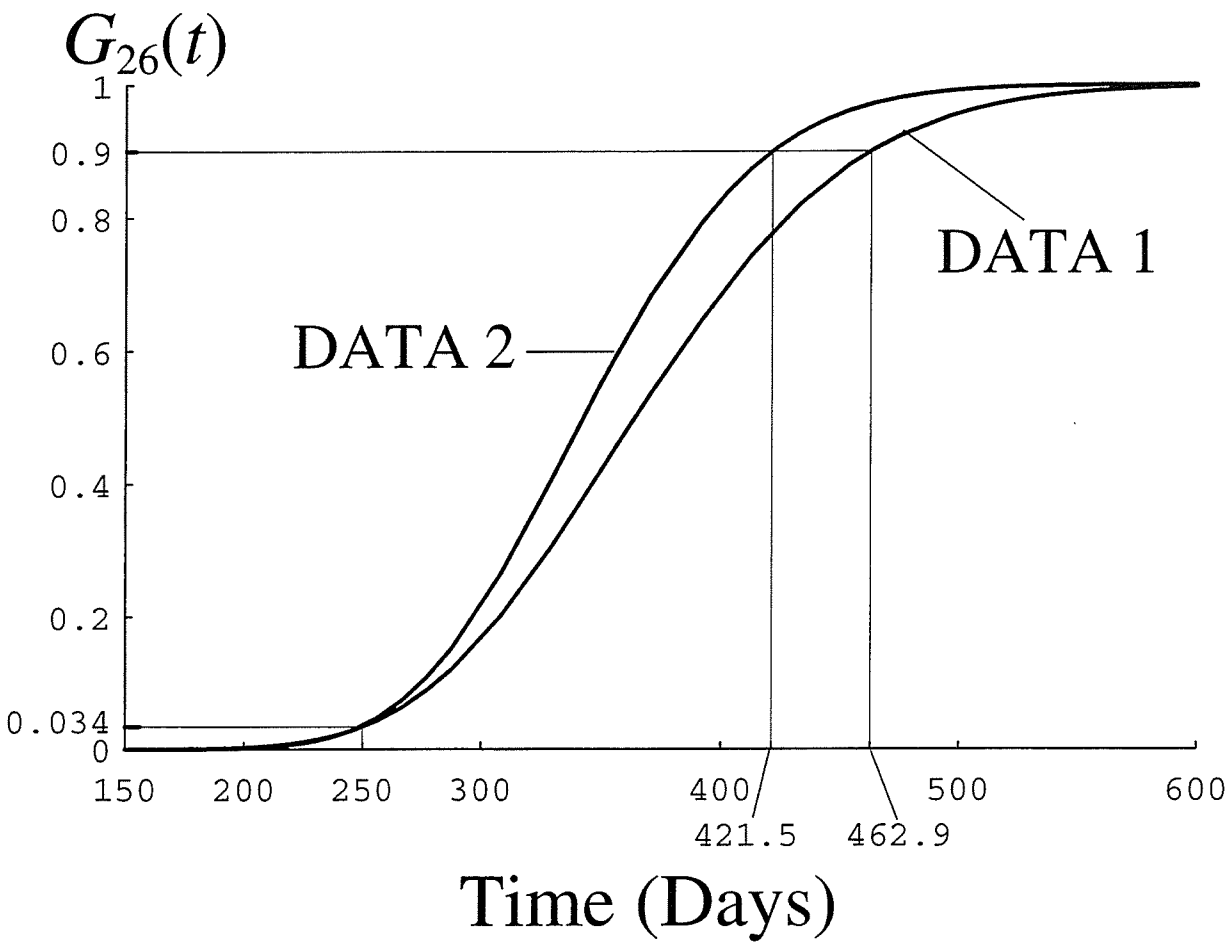

Fig. 4.3. Estimated $\widehat{G_{26}(t)}$ $(a = 0.8)$.

Figure 4.4 displays the operational state occupancy probabilities $P_{W_n}(t)$'s in (4.26) for various n's in the case of DATA 1. Figure 4.4 indicates that the maximum probabilities that the system is in each state make transitions with the lapse of time.

Figures 4.5 and 4.6 show the estimated instantaneous software availabilities $A(t)$'s in (4.35) for DATA 1 and DATA 2, respectively. Furthermore, Figs. 4.7 and 4.8 show the estimated average software availabilities $A_{av}(t)$'s in (4.36) for DATA 1 and DATA 2, respectively.

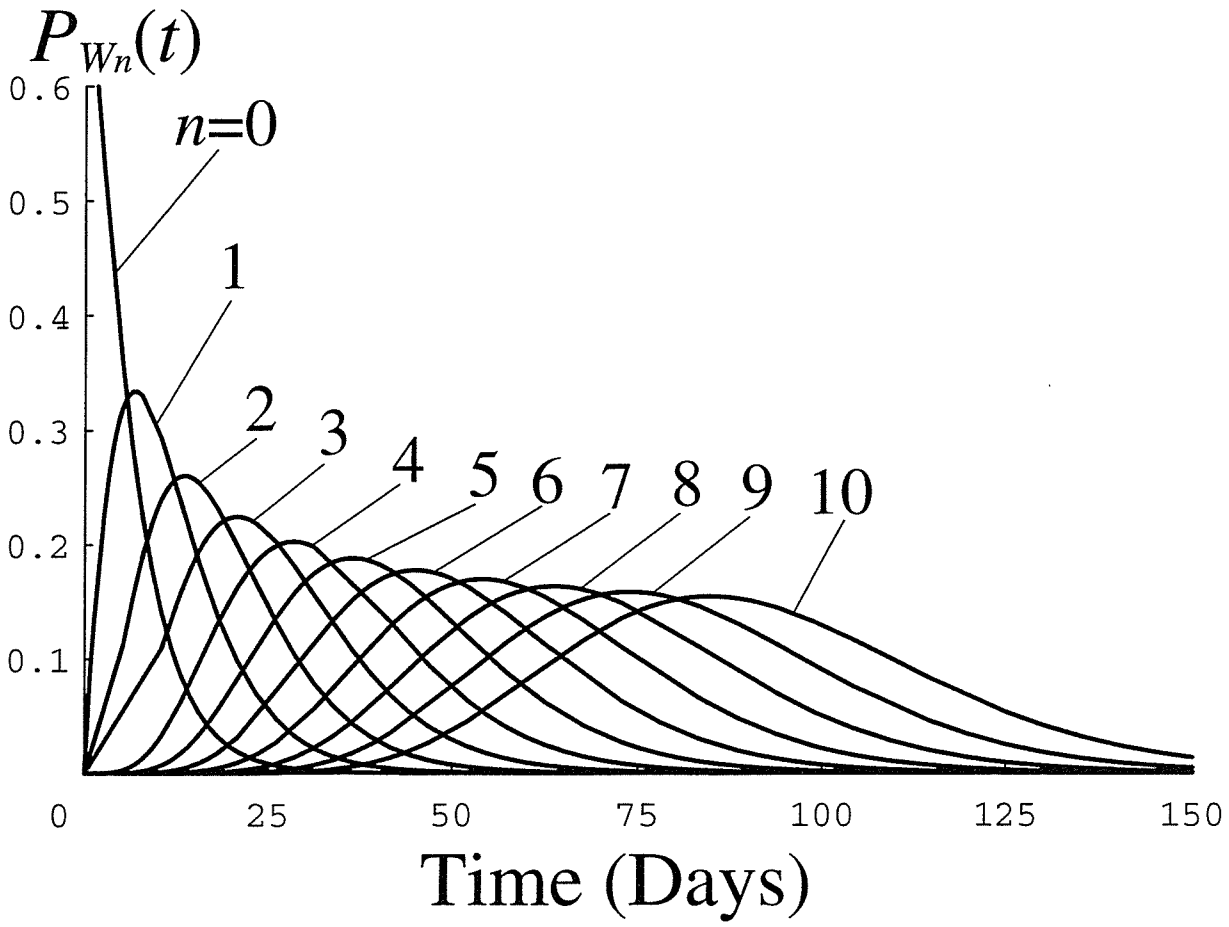

Fig. 4.4. Operational state occupancy probability $P_{W_n}(t)$ (DATA 1: $a = 0.8$).

We define the inherent availability for the next up-down cycle when n faults have been corrected as follows [55]:

$$\begin{aligned} A_I(n) &\equiv \frac{\mathrm{E}[T_n]}{\mathrm{E}[T_n] + \mathrm{E}[U_n]} \\ &= \frac{1/\lambda_n}{1/\lambda_n + 1/\mu_n} \\ &= \frac{1}{1+\rho_n} \quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{4.45}$$

$$\begin{aligned} \rho_n &\equiv \lambda_n/\mu_n \\ &= C\gamma^n \quad (C \equiv D/E,\ \gamma \equiv k/r), \end{aligned} \tag{4.46}$$

where we call ρ_n the maintenance factor and C and γ the initial maintenance factor and the availability improvement parameter, respectively. For DATA 1 and DATA 2, C and γ are given by

$$\widehat{C} = 0.221,\ \widehat{\gamma} = 0.979 \quad (\text{DATA 1: } a = 0.8),$$
$$\widehat{C} = 0.730,\ \widehat{\gamma} = 1.002 \quad (\text{DATA 2: } a = 0.8),$$

respectively. As shown in Figs. 4.5–4.8, in case of $\gamma < 1$ and $\gamma > 1$, the software availability improves and lowers with the lapse of time, respectively. Then, γ can be regarded as an index to determine whether or not the software availability grows and reflects the degree of difficulty of the software performance improvement.

The initial inherent availability, i.e. $A_I(0) = 1/(1 + C)$ depends on only the initial maintenance factor C. $A_I(0)$'s for DATA 1 and DATA 2 are given by

$$\widehat{A_I(0)} = 0.819 \quad (\text{DATA 1: } a = 0.8),$$
$$\widehat{A_I(0)} = 0.578 \quad (\text{DATA 2: } a = 0.8),$$

respectively. We can see that C determines the software availability in the early stage of operation.

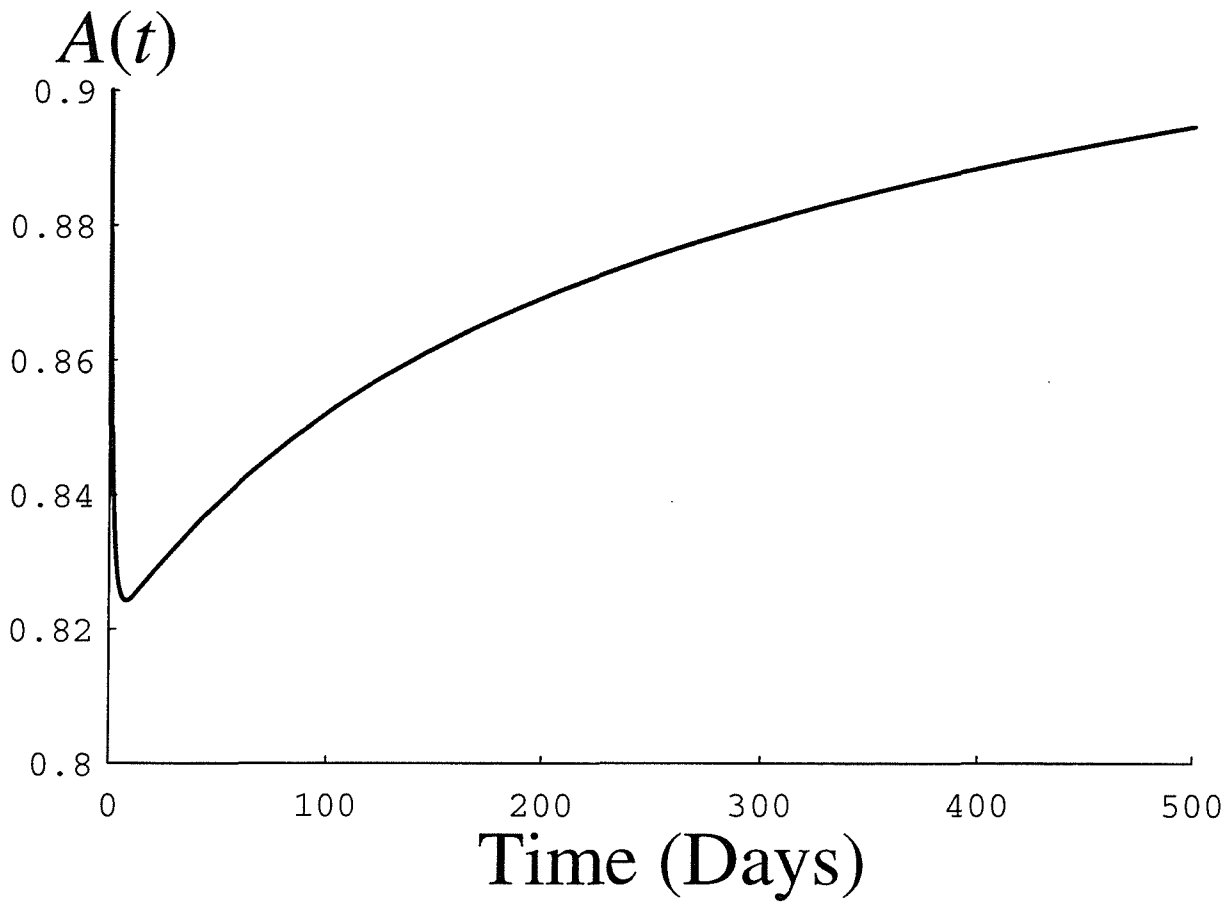

Fig. 4.5. Estimated $\widehat{A(t)}$ (DATA 1: $a = 0.8$).

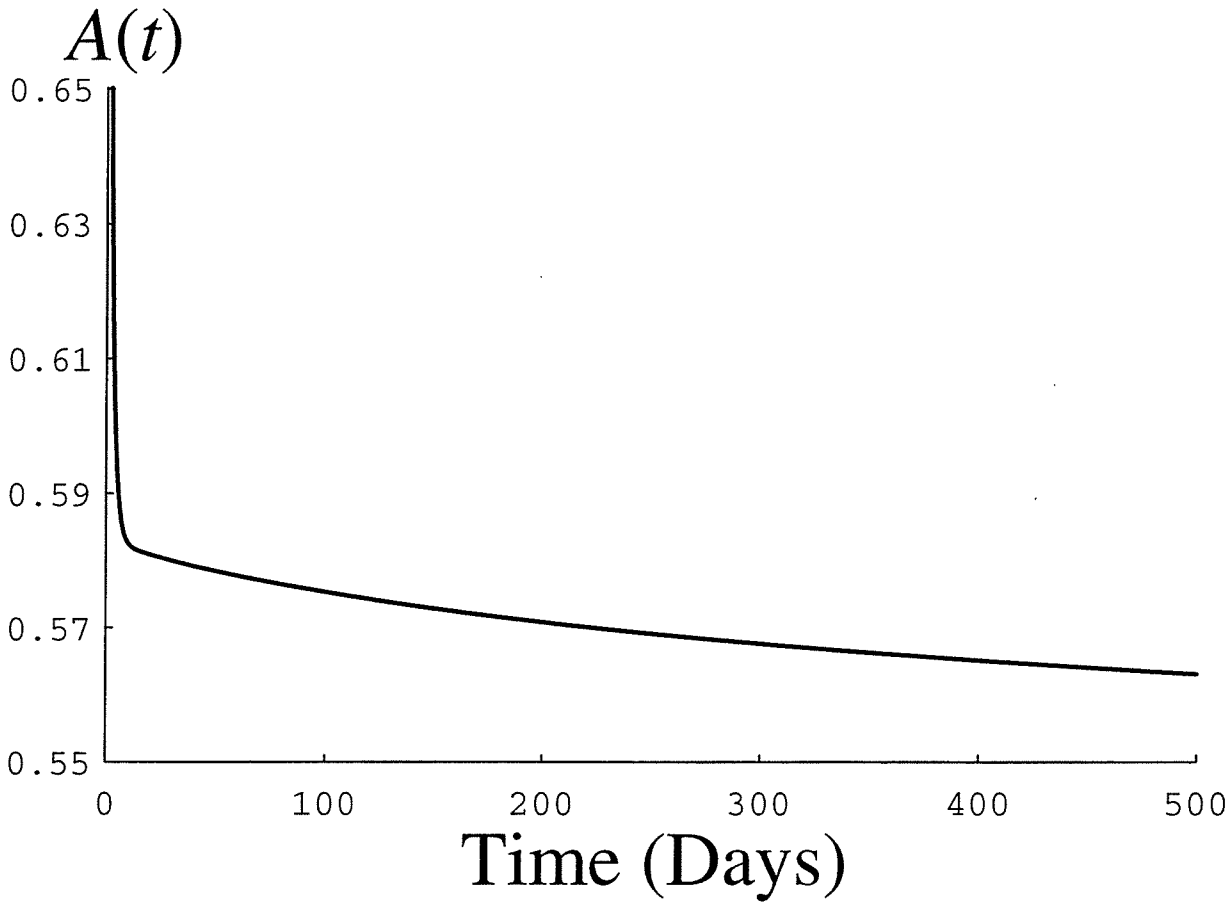

Fig. 4.6. Estimated $\widehat{A(t)}$ (DATA 2: $a = 0.8$).

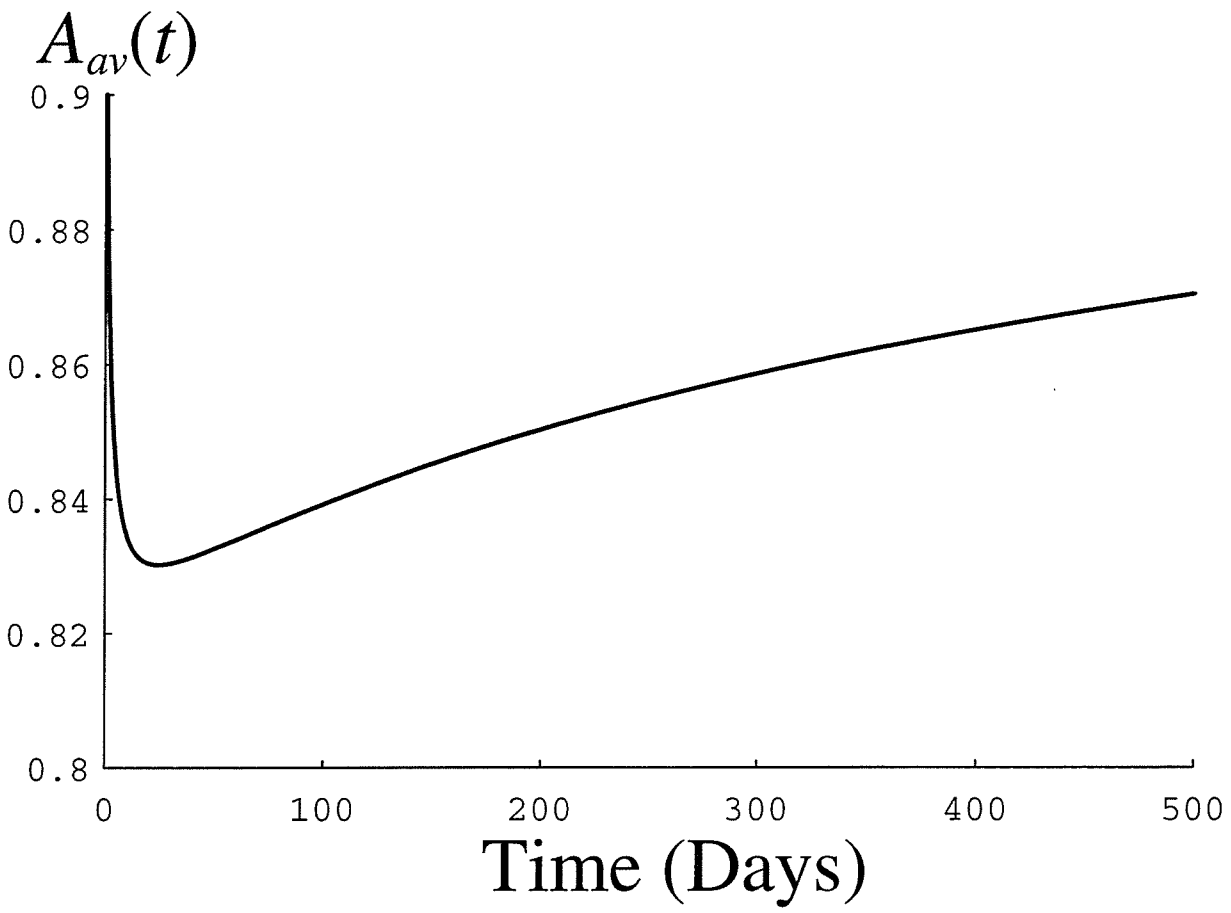

Fig. 4.7. Estimated $\widehat{A_{av}(t)}$ (DATA 1: $a = 0.8$).

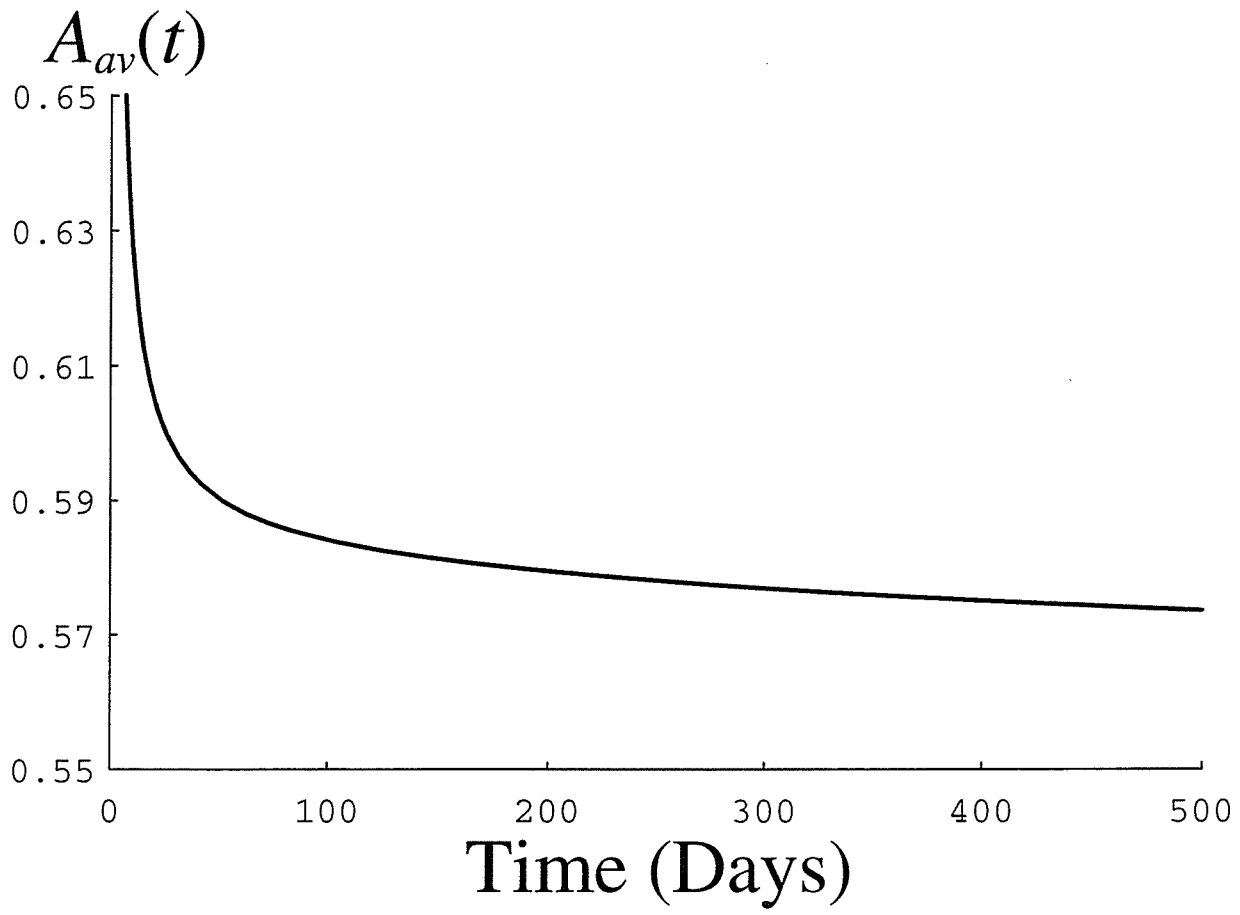

Fig. 4.8. Estimated $\widehat{A_{av}(t)}$ (DATA 2: $a = 0.8$).

4.6 Concluding Remarks

In this chapter, we have developed a software availability model considering a situation where the debugging activity is not always performed perfectly and the fault complexity increases with the progress of fault correction. Several useful stochastic quantities for software reliability/availability assessment have been derived from this model. Furthermore, the estimation of unknown parameters has been discussed. Numerical illustrations for software availability measurement have been also presented to show that these quantities are very useful for software performance assessment.

Chapter 5

Operational Software Availability Modeling with Two Types of Failures

5.1 Introduction

In complex computer systems, it is not too much to say that the ability of software components, which is a major one in computer systems, determines the performance of the overall system. It is crucial to achieve an appropriate level of software reliability efficiently and economically since software reliability is the most important aspect of software quality. Many analytical models have been developed and studied in order to capture reliability of implemented software products quantitatively [43]. A mathematical reliability assessment model is a software reliability growth model which describes a software fault-detection or a software failure-occurrence phenomenon in dynamic environment such as the testing phase in the software development process and the operation phase [35, 56]. A software failure is defined as an unacceptable departure from program operation caused by a fault remaining in the software system. This model is utilized for measuring the degree of achievement of software reliability, determining the testing time duration for software release, and estimating the maintenance cost for faults undetected during the testing phase.

Software development managers and customers have taken a growing interest in the software performance measures such as the possible utilization factor since today's computer systems have required nonstop operation. To construct a software reliability model incorporating recoverability is an interesting matter. This model can measure and assess software availability, which is defined as the probability that the software system is per-

forming successfully at a specified time point [25]. In particular, for telecommunication and switching systems, software availability is one of the software quality characteristics not to be negligible. Though extensive research has been done on performance evaluation techniques for hardware systems, few analytical models for measuring software availability are proposed [19, 21].

In this chapter, we construct a software availability model considering two types of software failures during the operation phase, based on the model discussed in Chapter 4. The causes of software failures include not only the faults that could not be detected/ corrected during the testing phase but also operation misses of customers or unexpectable operational conditions not described in the requirement and design specifications. Recently, it have been often reported that computer systems are down because situations not supposed in the system specification have occurred. Since software availability is a customer-oriented metric, it is interesting to consider such software failures in software availability modeling. The occurrence phenomena of two types of software failures mentioned above are described by a geometrically decreasing and a constant hazard rate, respectively. Furthermore, this model considers the imperfect debugging environment where faults are not always corrected and removed from the system when debugging activities are performed.

The time-dependent behavior in which the system alternates between the operational state (up state) that a system is operating regularly and the restoration state (down state) that a system is inoperable and restored can be modeled by a Markov process [47]. The description of software availability modeling is discussed in Section 5.2. Several stochastic quantities for software availability measurement are derived in Section 5.3. Numerical illustrations for software reliability/availability analyses are presented in Section 5.4.

5.2 Model Description

In this chapter, we assume that the following two types of software failures exist during the operation phase:

F1: software failures caused by the faults that could not be detected/corrected during the testing phase.

F2: software failures caused by the faults introduced by deviating from the expected operational use.

The following assumptions are made for software availability modeling:

A1. The software system breaks down and starts to be restored as soon as a software failure occurs, and the system can not operate until the restoration action is complete.

A2. The restoration action for F1 implies the debugging activity and software reliability growth occurs if a debugging activity is perfect. The restoration action for F2 has no effect on software reliability growth.

A3. The debugging activity for F1 is perfect with probability a $(0 < a \leq 1)$, while imperfect with probability $b(= 1 - a)$. We call a the perfect debugging rate. A perfect debugging activity corrects and removes one fault from the system.

A4. When n faults have been corrected, the next occurrence time-interval and the restoration time for F1 follow exponential distributions with means $1/\lambda_n$ and $1/\mu_n$, respectively.

A5. The occurrence time-interval and the restoration time for F2 follow exponential distributions with means $1/\theta$ and $1/\eta$, respectively.

A6. The probability that two or more software failures occur simultaneously is negligible.

We introduce a stochastic process $\{X(t),\ t \geq 0\}$ representing the state of the software system at time point t. The state space of the process $\{X(t),\ t \geq 0\}$ is defined as follows:

W_n: the system is operating,

R_n^1: the system is inoperable due to F1 and restored,

R_n^2: the system is inoperable due to F2 and restored,

where $n = 0,\ 1,\ 2,\ \ldots$ denotes the cumulative number of faults corrected during the operation phase.

From assumption A3, when a restoration action for F1 is complete in $\{X(t) = R_n^1\}$,

$$X(t) = \begin{cases} W_n & \text{(with probability } b\text{)} \\ W_{n+1} & \text{(with probability } a\text{).} \end{cases} \tag{5.1}$$

Next, we describe the software failure-occurrence phenomenon for F1. In this chapter, we use Moranda's model [31], i.e. the hazard rate λ_n is given by

$$\lambda_n = Dk^n \quad (n = 0\ ,1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{5.2}$$

where D and k are the initial hazard rate and the decreasing ratio of the hazard rate for F1, respectively. The description of (5.2) comes from the point of view that software reliability depends on the debugging efforts, not the residual fault content. We do not pay attention to the number of faults remaining in the software system. In conventional software availability modeling, e.g. Kim et al. [16] and Okumoto & Goel [39], it is often assumed that the hazard rate is proportional to the residual fault content and decreases by a constant amount with perfect debugging. Equation (5.2) describes a software failure-occurrence phenomenon where perfect debugging corresponding to software failures occurring in the early stage of the operation phase contributes largely to improvement of software reliability [25]. It is probable that (5.2) fits in with the operational environment where several functions executed with high frequency are relatively specified. This reason is that faults latent in modules with high frequency of execution are detected early and reliability of corresponding modules improves rapidly. In the above operational environment, reliability of the overall system also improves rapidly even if reliability of modules with low frequency of execution is low.

Furthermore, we describe the restoration characteristic for F1. Generally, the faults detected later tend to have higher complexity, i.e. it takes more time to isolate the faults and to check the fault correction [36]. Accordingly, it is appropriate that the restoration time for F1 becomes longer with increasing n. Then, we assume μ_n as follows:

$$\mu_n = Er^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ E > 0,\ 0 < r \leq 1), \tag{5.3}$$

where E and r are the initial restoration rate and the decreasing ratio of the restoration rate for F1, respectively.

Finally, we describe the failure and the restoration characteristic for F2. During the testing phase of the software development process, the implemented system is tested to verify whether or not the system operates in accordance with the specification and software reliability growth occurs with debugging activities. However, it is often that the operational conditions not described in the specifications occur during the operation phase; for instance, many transactions have rushed at some time point or the loads into some specified system resources have concentrated. It is assumed that the software failures due to the condition mentioned above occur randomly throughout the operation phase. Furthermore, for the above software failures, the system is often restored without the debugging activity when it is more important to shorten inoperable time than to adapt the system to operational environment different from the specifications. Even if the debugging activity is performed, it is often makeshift and the similar software failures may recur. Since there are various restoration procedures for F2, it is assumed that the restoration time-interval is also random and the restoration action for F2 has no bearing on software reliability growth throughout the operation phase.

Let $Q_{A,B}(\tau)$ ($A,\ B \in \{W_n,\ R_n^1,\ R_n^2;\ n = 0,\ 1,\ 2,\ \ldots\}$) denote the one-step transition probability that after making a transition into state A, the process $\{X(t),\ t \geq 0\}$ makes a transition into state B by time τ. The expressions for $Q_{A,B}(\tau)$'s are given as follows:

$$Q_{W_n,R_n^1}(\tau) = \frac{\lambda_n}{\theta + \lambda_n}[1 - e^{-(\theta+\lambda_n)\tau}], \tag{5.4}$$

$$Q_{W_n,R_n^2}(\tau) = \frac{\theta}{\theta + \lambda_n}[1 - e^{-(\theta+\lambda_n)\tau}], \tag{5.5}$$

$$Q_{R_n^1,W_{n+1}}(\tau) = a(1 - e^{-\mu_n\tau}), \tag{5.6}$$

$$Q_{R_n^1,W_n}(\tau) = b(1 - e^{-\mu_n\tau}), \tag{5.7}$$

$$Q_{R_n^2,W_n}(\tau) = 1 - e^{-\eta\tau}. \tag{5.8}$$

The sample state transition diagram of $X(t)$ is illustrated in Fig. 5.1.

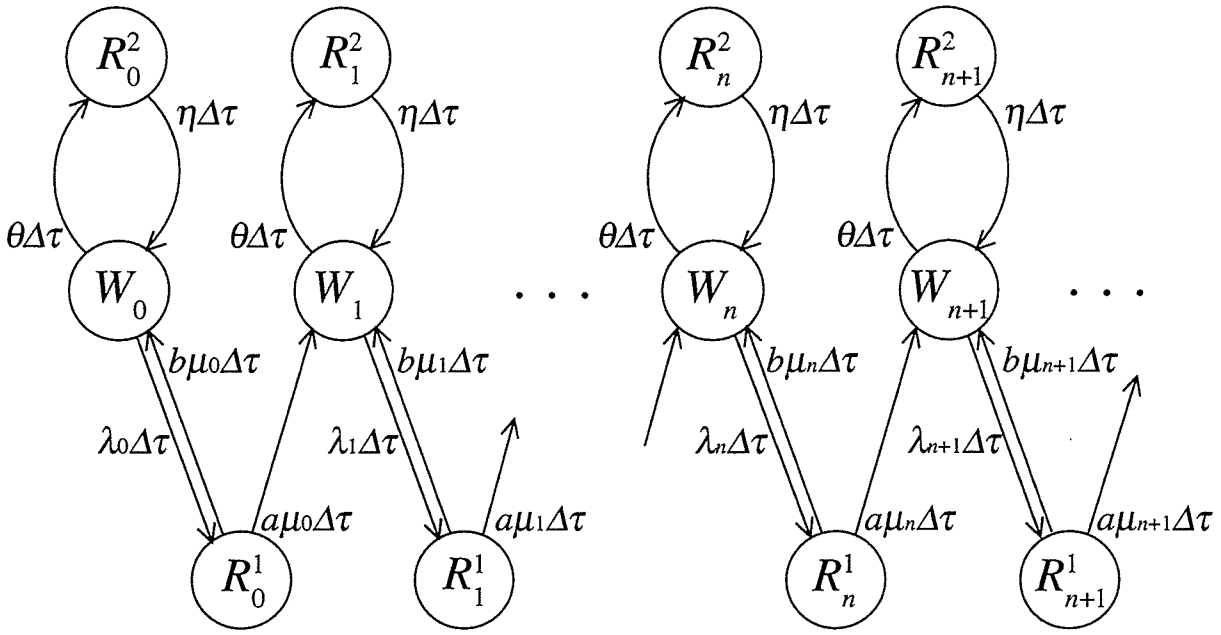

Fig. 5.1. A diagrammatic representation of state transitions between $X(t)$'s for operational software availability modeling (I).

5.3 Derivation of Software Performance Measures

5.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Let S_n $(n = 1, 2, \ldots; S_0 \equiv 0)$ be the random variable representing the time spent in correcting n faults and $G_{i,n}(t)$ be the distribution function associated with the probability that n faults are corrected in the time interval $(0, t]$ on the condition that $i(< n)$ faults have been already corrected at time zero. Then, we obtain the following renewal equation:

$$G_{i,n}(t) = Q_{W_i,R_i^1} * Q_{R_i^1,W_{i+1}} * G_{i+1,n}(t) + Q_{W_i,R_i^1} * Q_{R_i^1,W_i} * G_{i,n}(t) + Q_{W_i,R_i^2} * Q_{R_i^2,W_i} * G_{i,n}(t) \qquad (i = 0, 1, 2, \ldots, n-1), \tag{5.9}$$

where $*$ denotes a Stieltjes convolution and $G_{n,n}(t) = \mathbf{1}(t)$ (unit function) $(n = 1, 2, \ldots)$. Then, we get the Laplace-Stieltjes (L-S) transform [42] of (5.9) as

$$\tilde{G}_{i,n}(s) = \tilde{Q}_{W_i,R_i^1}(s)\tilde{Q}_{R_i^1,W_{i+1}}(s)\tilde{G}_{i+1,n}(s) + \tilde{Q}_{W_i,R_i^1}(s)\tilde{Q}_{R_i^1,W_i}(s)\tilde{G}_{i,n}(s) + \tilde{Q}_{W_i,R_i^2}(s)\tilde{Q}_{R_i^2,W_i}(s)\tilde{G}_{i,n}(s) \qquad (i = 0, 1, 2, \ldots, n-1). \tag{5.10}$$

From (5.4)–(5.8), the L-S transforms of $Q_{A,B}(t)$'s are respectively given as

$$\tilde{Q}_{W_i,R_i^1}(s) = \frac{\lambda_i}{s+\theta+\lambda_i}, \tag{5.11}$$

$$\tilde{Q}_{W_i,R_i^2}(s) = \frac{\theta}{s+\theta+\lambda_i}, \tag{5.12}$$

$$\tilde{Q}_{R_i^1,W_{i+1}}(s) = \frac{a\mu_i}{s+\mu_i}, \tag{5.13}$$

$$\tilde{Q}_{R_i^1,W_i}(s) = \frac{b\mu_i}{s+\mu_i}, \tag{5.14}$$

$$\tilde{Q}_{R_i^2,W_i}(s) = \frac{\eta}{s+\eta}. \tag{5.15}$$

Substituting (5.11)–(5.15) into (5.10) yields

$$\tilde{G}_{i,n}(s) = \frac{a\lambda_i\mu_i(s+\eta)}{(s+x_i)(s+y_i)(s+z_i)}\tilde{G}_{i+1,n}(s) \quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{5.16}$$

where $-x_i$, $-y_i$, and $-z_i$ are the distinct roots of the following third order equation:

$$s^3 + (\theta+\eta+\lambda_i+\mu_i)s^2 + (\theta\mu_i+\eta\lambda_i+\eta\mu_i+a\lambda_i\mu_i)s + a\eta\lambda_i\mu_i = 0. \tag{5.17}$$

Solving (5.16) recursively, we obtain the L-S transform of $G_{0,n}(t)$ as

$$\begin{aligned}\tilde{G}_{0,n}(s) &= \prod_{i=0}^{n-1} \frac{a\lambda_i\mu_i(s+\eta)}{(s+x_i)(s+y_i)(s+z_i)} \\ &= \sum_{i=0}^{n-1}\left(\frac{A_{n,i}^1 x_i}{s+x_i} + \frac{A_{n,i}^2 y_i}{s+y_i} + \frac{A_{n,i}^3 z_i}{s+z_i}\right),\end{aligned} \tag{5.18}$$

where constant coefficients $A_{n,i}^1$, $A_{n,i}^2$, and $A_{n,i}^3$ are given by

$$A_{n,i}^1 = \frac{[a(\eta-x_i)]^n \prod_{j=0}^{n-1}\lambda_j\mu_j}{x_i \prod_{\substack{j=0\\ j\neq i}}^{n-1}(x_j-x_i)\prod_{j=0}^{n-1}(y_j-x_i)(z_j-x_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{5.19}$$

$$A_{n,i}^2 = \frac{[a(\eta-y_i)]^n \prod_{j=0}^{n-1}\lambda_j\mu_j}{y_i \prod_{\substack{j=0\\ j\neq i}}^{n-1}(y_j-y_i)\prod_{j=0}^{n-1}(z_j-y_i)(x_j-y_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{5.20}$$

$$A^3_{n,i} = \frac{[a(\eta - z_i)]^n \prod_{j=0}^{n-1} \lambda_j \mu_j}{z_i \prod_{\substack{j=0 \\ j \neq i}}^{n-1} (z_j - z_i) \prod_{j=0}^{n-1} (x_j - z_i)(y_j - z_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{5.21}$$

respectively. By inverting (5.18) and rewriting $G_{0,n}(t)$ as $G_n(t)$, we obtain the distribution function for S_n as

$$\begin{aligned} G_n(t) &\equiv \Pr\{S_n \leq t\} \\ &= 1 - \sum_{i=0}^{n-1} (A^1_{n,i} e^{-x_i t} + A^2_{n,i} e^{-y_i t} + A^3_{n,i} e^{-z_i t}) \\ &\quad (n = 1,\ 2,\ \ldots;\ G_0(t) \equiv \mathbf{1}(t)), \end{aligned} \tag{5.22}$$

where we postulate $\prod_{\substack{j=0 \\ j \neq 0}}^{0} \cdot = 1$. It is noted that

$$\sum_{i=0}^{n-1} (A^1_{n,i} + A^2_{n,i} + A^3_{n,i}) = 1. \tag{5.23}$$

Furthermore, $\mathrm{E}[S_n]$ and $\mathrm{Var}[S_n]$ are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1} \left(\frac{1}{x_i} + \frac{1}{y_i} + \frac{1}{z_i} - \frac{1}{\eta} \right), \tag{5.24}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1} \left(\frac{1}{{x_i}^2} + \frac{1}{{y_i}^2} + \frac{1}{{z_i}^2} - \frac{1}{\eta^2} \right), \tag{5.25}$$

respectively.

5.3.2 Operational State Occupancy Probability and Software Availability

Let $P_{i,n}(t)$ be the conditional state occupancy probability that the system is operating at time point t when n faults have been corrected on the condition that the system was operating at time point zero when i faults had been already corrected, i.e.

$$P_{i,n}(t) \equiv \Pr\{X(t) = W_n | X(0) = W_i\} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n), \tag{5.26}$$

and $P_n(t) \equiv P_{0,n}(t)$ is called the operational state occupancy probability. Then, we obtain the following renewal equations:

$$P_n(t) = G_n * P_{n,n}(t), \tag{5.27}$$

$$P_{n,n}(t) = e^{-(\theta+\lambda_n)t} + Q_{W_n,R_n^1} * Q_{R_n^1,W_n} * P_{n,n}(t) + Q_{W_n,R_n^2} * Q_{R_n^2,W_n} * P_{n,n}(t). \tag{5.28}$$

From (5.28), the L-S transform of $P_{n,n}(t)$ is given by

$$\begin{aligned}\tilde{P}_{n,n}(s) &= \frac{s(s+\mu_n)(s+\eta)}{(s+x_n)(s+y_n)(s+z_n)} \\ &= \left(\frac{s}{a\lambda_n} + \frac{s^2}{a\lambda_n\mu_n}\right)\frac{a\lambda_n\mu_n(s+\eta)}{(s+x_n)(s+y_n)(s+z_n)}.\end{aligned} \tag{5.29}$$

Substituting (5.29) into the L-S transform of (5.27) yields

$$\tilde{P}_n(s) = \frac{s}{a\lambda_n}\tilde{G}_{n+1}(s) + \frac{s^2}{a\lambda_n\mu_n}\tilde{G}_{n+1}(s). \tag{5.30}$$

By inverting (5.30), the operational state occupancy probability is obtained as

$$\begin{aligned}P_n(t) &\equiv \Pr\{X(t) = W_n\} \\ &= \frac{1}{a\lambda_n}g_{n+1}(t) + \frac{1}{a\lambda_n\mu_n}g'_{n+1}(t),\end{aligned} \tag{5.31}$$

where $g_n(t)$ is the probability density function of the random variable S_n and $g'_n(t) \equiv dg_n(t)/dt$. Equation (5.31) is the same form as derived in Chapter 4.

The instantaneous software availability is defined as

$$A(t) \equiv \sum_{n=0}^{\infty} P_n(t), \tag{5.32}$$

which represents the probability that the software system is operable at specified time point t. Furthermore, the average software availability in the time interval $(0,\ t]$ is defined as

$$A_{av}(t) \equiv \frac{1}{t}\int_0^t A(x)dx, \tag{5.33}$$

which represents the ratio of system's operating time to the time interval $(0,\ t]$. Using (5.31), we can describe (5.32) and (5.33) as

$$A(t) = \sum_{n=0}^{\infty}\left[\frac{g_{n+1}(t)}{a\lambda_n} + \frac{g'_{n+1}(t)}{a\lambda_n\mu_n}\right], \tag{5.34}$$

$$A_{av}(t) = \frac{1}{t}\sum_{n=0}^{\infty}\left[\frac{G_{n+1}(t)}{a\lambda_n} + \frac{g_{n+1}(t)}{a\lambda_n\mu_n}\right], \tag{5.35}$$

respectively.

5.4 Numerical Examples

Using the software availability model discussed above, we show numerical illustrations for software availability measurement.

The distribution functions of the first passage time to the specified number of corrected faults, $G_n(t)$'s in (5.22) are shown in Fig. 5.2 for various perfect debugging rates, a's, where $n = 5$, $\theta = 0.01$, $\eta = 1.0$, $D = 0.1$, $E = 0.5$, and $r = 0.9$. Figure 5.2 shows that the smaller a becomes, the longer time it takes to remove faults from the system.

The instantaneous software availabilities, $A(t)$'s in (5.34) are shown in Fig. 5.3 for various values of a where $\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, and $r = 0.9$. In practical calculation of $A(t)$ and $A_{av}(t)$, we need to specify the supremum of n instead of infinity. We denote this value as N. Figure 5.3 displays the time-dependent behavior of $A(t)$ in time interval [0, 800]. In this case, $G_{20}(800) = 2.38 \times 10^{-12}$ when $a = 1.0$. That is, the probability that 20 faults are corrected up to time $t = 800$ are sufficiently small. Accordingly, we set $N = 20$. Figure 5.3 indicates that software availability is low immediately after the operation and increases gradually with the lapse of the operation time and that (5.34) can measure the degree of unstableness of the system in the early stage of the operation phase. This figure also shows that the higher certainty of the debugging activity becomes, the larger software availability becomes.

$A(t)$'s and the average software availabilities, $A_{av}(t)$'s in (5.35) are shown in Figs. 5.4 and 5.5 for various values of θ, where $a = 1.0$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, and $r = 0.9$, respectively. The hazard rate for F2, θ reflects the specification quality; i.e., the specification covers the user requirements well with decreasing θ. Figures 5.4 and 5.5 indicate that the utilization of the software system becomes larger as θ decreases.

$A(t)$'s for various values of r are shown in Fig. 5.6, where $a = 0.9$, $\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, and $E = 0.5$. As shown in Fig. 5.6, in case of $r > k$ and $r < k$, software availability improves and decreases with the lapse of time, respectively, and in case of $r = k$, this is constant. Then, the ratio of r to k can be regarded as an index to determine whether or not the software availability grows during the operation phase.

The hazard rate for the next software failure, which is not distinguished between F1

and F2, when n faults have been already corrected is denoted as

$$\alpha_n = \lambda_n + \theta \quad (n = 0,\ 1,\ 2,\ \ldots). \tag{5.36}$$

$A(t)$'s in the cases of (i) $D : \theta = 1 : 5$ and (ii) $D : \theta = 5 : 1$ on the condition that $\alpha_0 = D + \theta$ is the same value are shown in Fig. 5.7, where $a = 0.9$, $\eta = 1.0$, $k = 0.8$, $E = 0.5$, and $r = 0.9$. Figure 5.7 shows as follows: In the case of (i), software availability is stable when debugging is performed enough and the hazard rate for F1 is low. In the case of (ii), though software availability is lower than (i) in the early stage of the operation phase, it increases with the lapse of time and is larger than (i). The behavior shown in Fig. 5.7 is due to whether the system has much room for reliability growth or not. For example, $\alpha_0 = 0.06$ for both of (i) and (ii), but $\alpha_{10} = 0.0510$ for (i) and $\alpha_{10} = 0.0154$ for (ii). Then, (ii) has more room for reliability growth than (i).

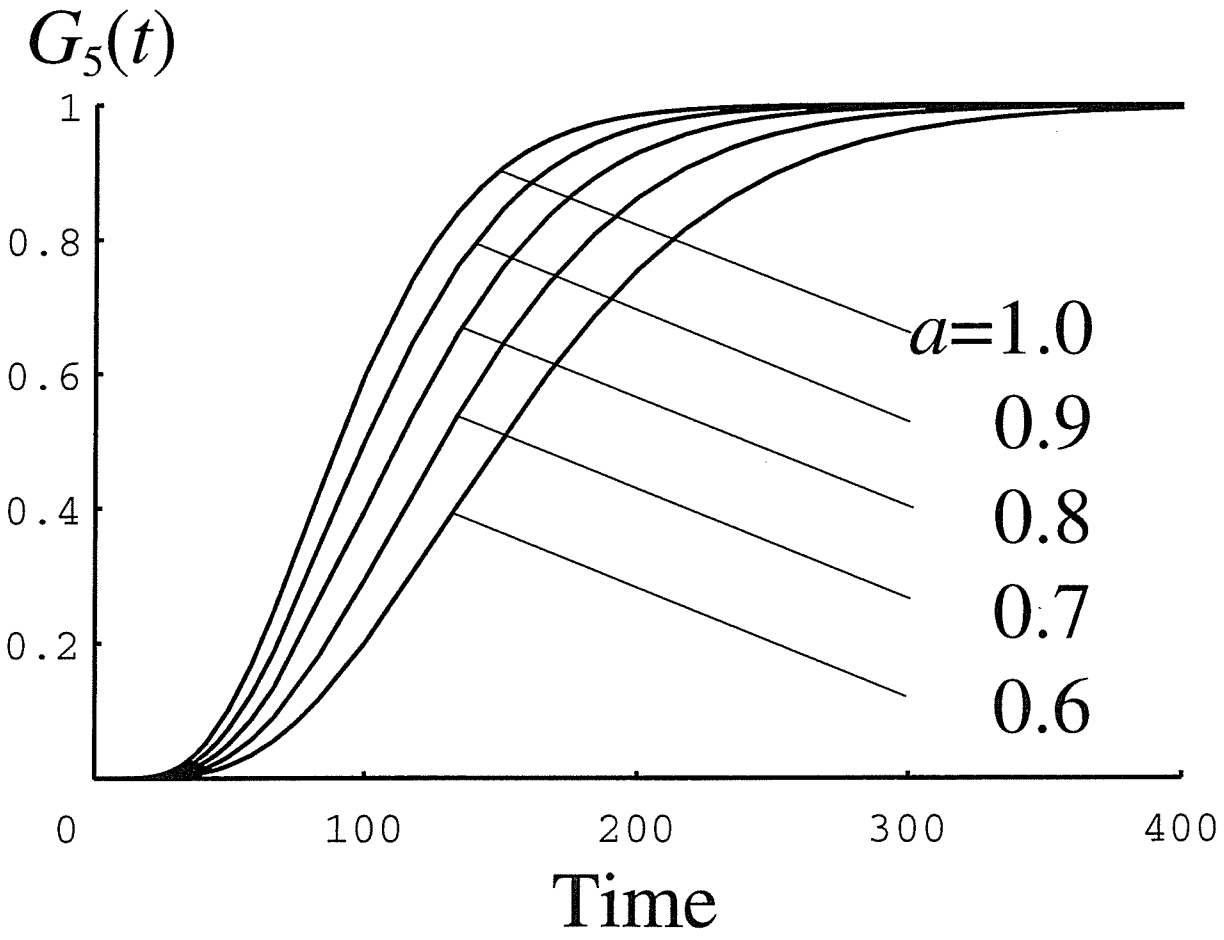

Fig. 5.2. Dependence of a on $G_n(t)$ ($n = 5$, $\theta = 0.01$, $\eta = 1.0$, $D = 0.1$, $k = 0.8$, $E = 0.5$, $r = 0.9$).

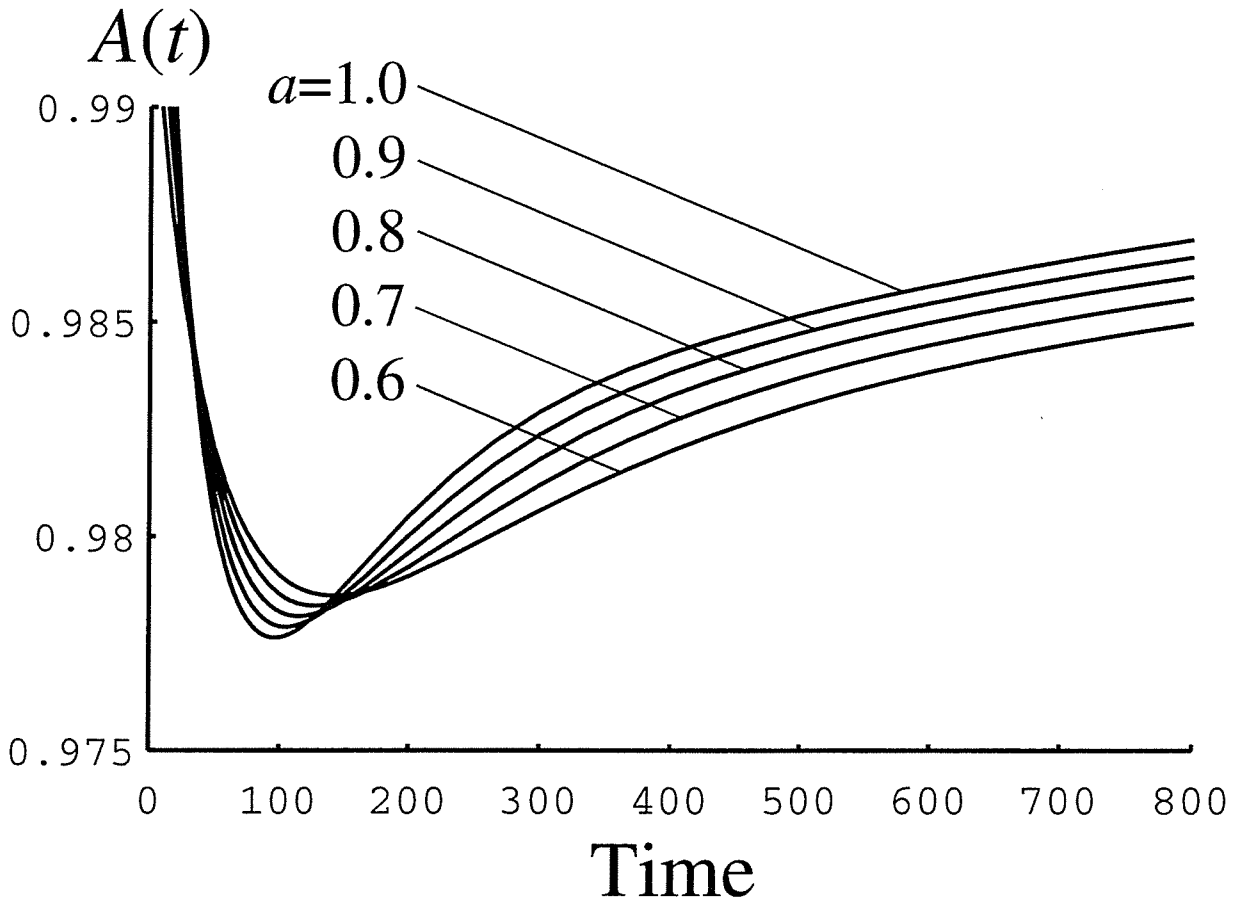

Fig. 5.3. Dependence of a on $A(t)$ ($\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, $r = 0.9$).

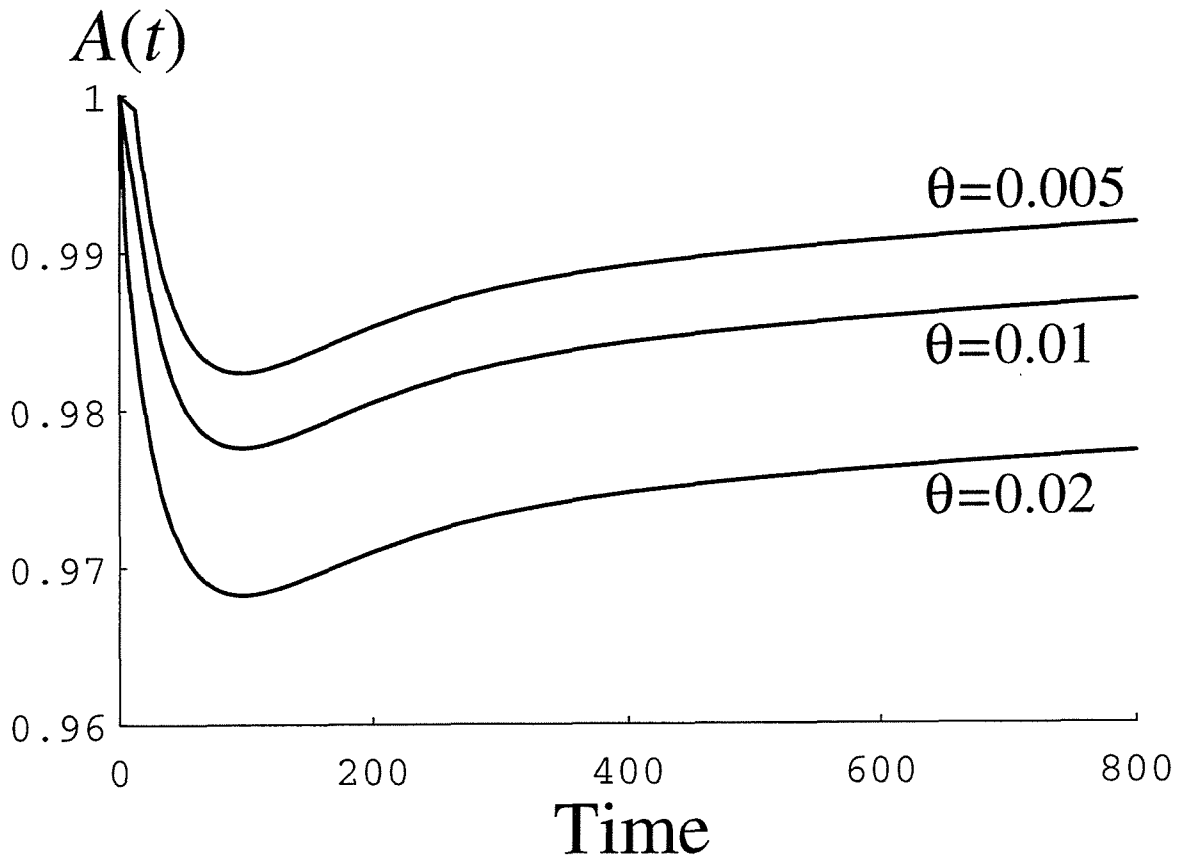

Fig. 5.4. Dependence of θ on $A_{av}(t)$ ($a = 1.0$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, $r = 0.9$).

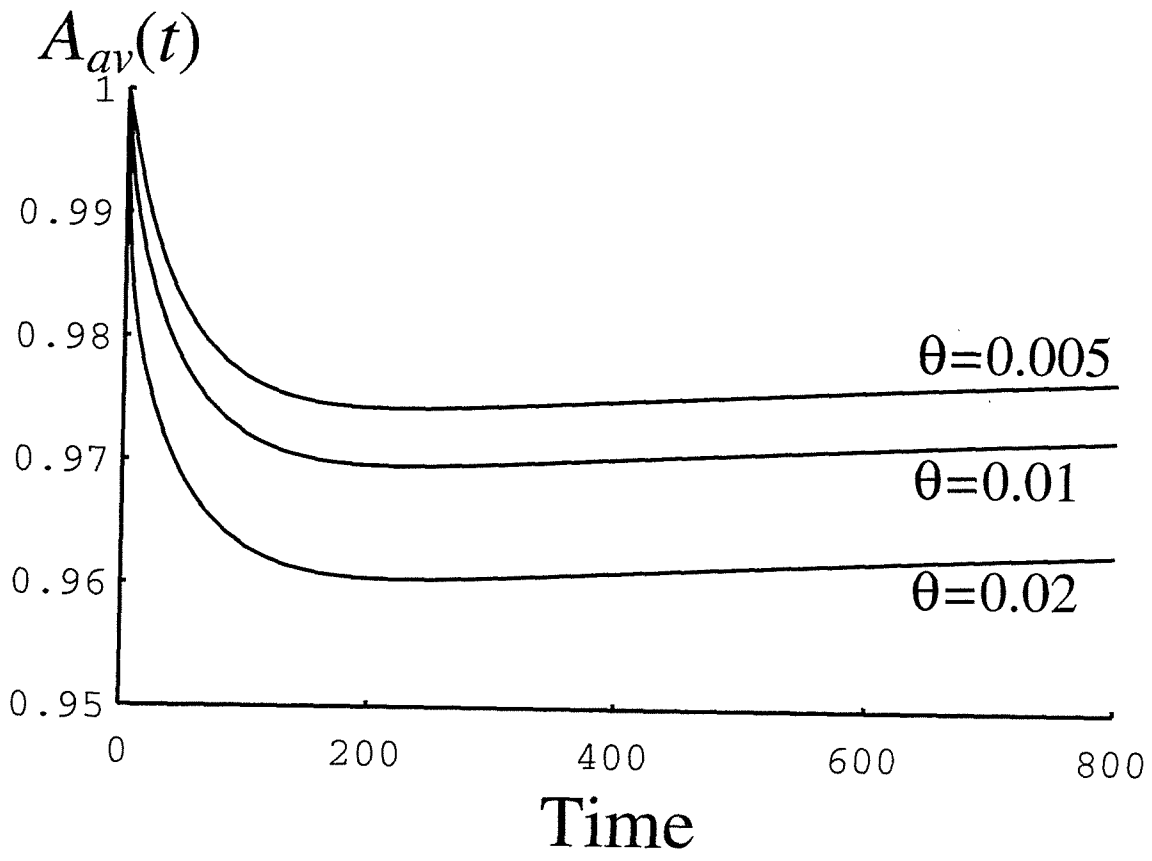

Fig. 5.5. Dependence of θ on $A_{av}(t)$ ($a = 1.0$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, $r = 0.9$).

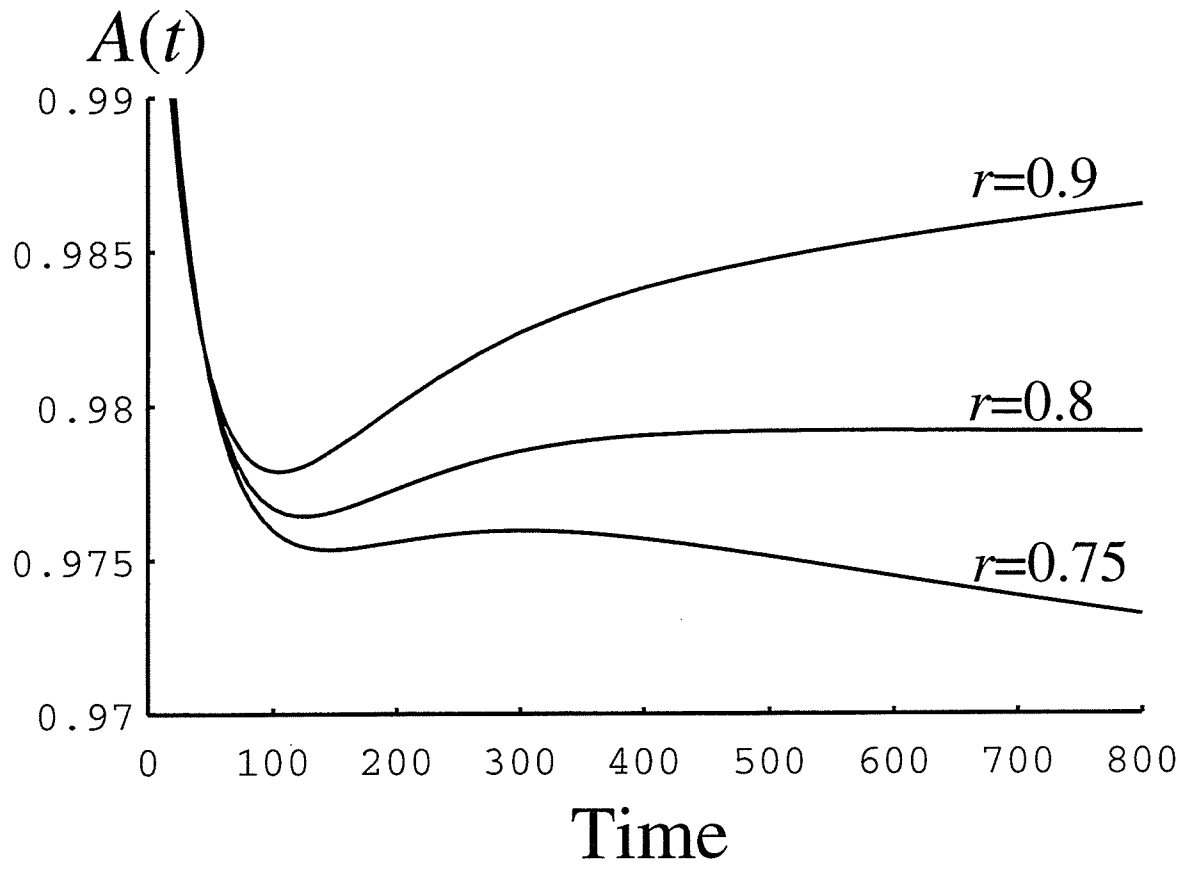

Fig. 5.6. Dependence of r on $A(t)$ ($a = 0.9$, $\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$).

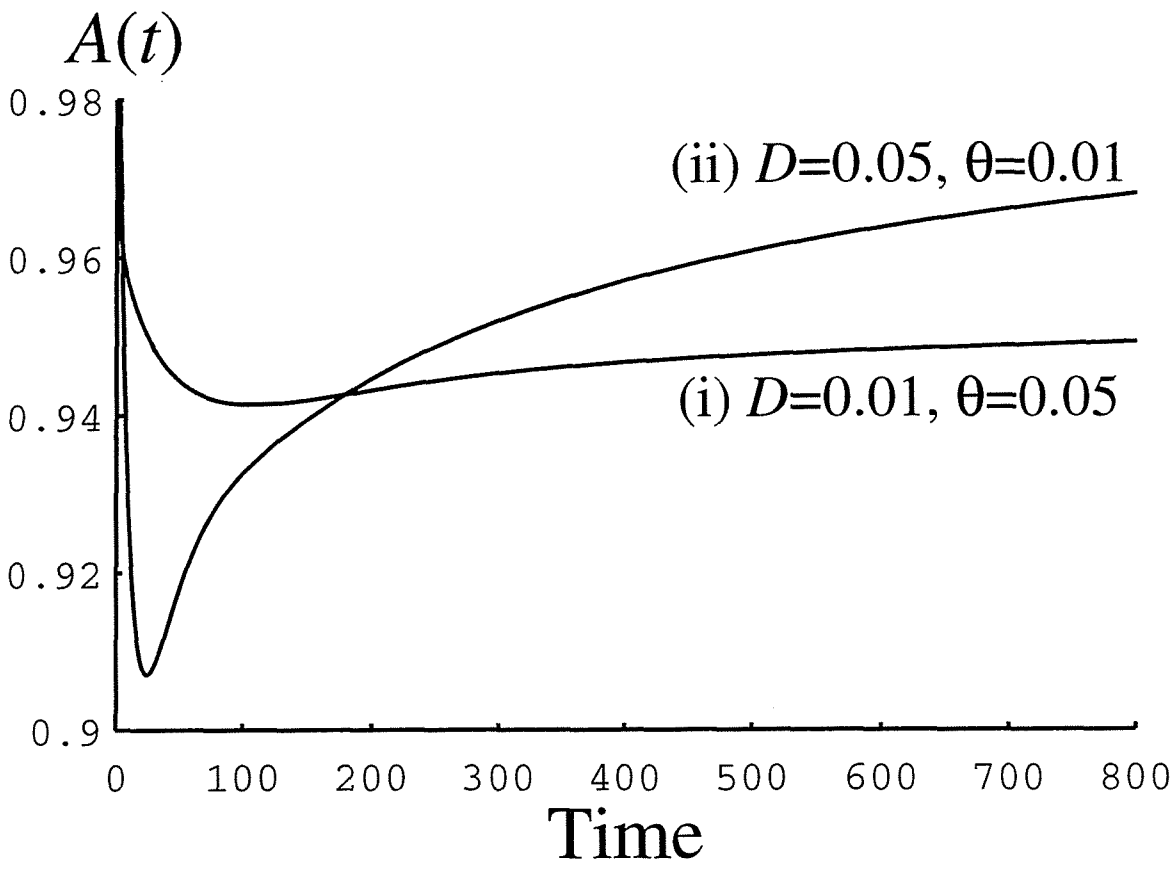

Fig. 5.7. Dependence of $D : \theta$ on $A(t)$ ($a = 0.9$, $\eta = 1.0$, $k = 0.8$, $E = 0.5$, $r = 0.9$).

5.5 Concluding Remarks

In this chapter, we have developed a software availability model integrating two different types of software failure-occurrence phenomena during the operation phase: the one caused by the faults remaining in the system is described by a geometrically decreasing hazard rate and the other caused by those introduced by deviating from the specification is described by a constant hazard rate. The dynamic behavior of the system has been modeled by a Markov process. Several quantitative measures for software performance assessment have been derived from this model. Numerical illustrations for software availability measurement have been also presented to show that these measures are very useful for software performance evaluation.

Chapter 6

Operational Software Availability Modeling with Two Types of Restorations

6.1 Introduction

In this chapter, we discuss another software availability model for operational use, taking notice of restoration scenarios during the operation phase. Debugging activities corresponding to software failures which occur during the operation phase are not always performed because protracting an inoperable time may much affect the customers. This is a different policy from the testing phase. Since software availability is one of the customer-oriented metrics, we need to reflect on operational environment in software availability modeling. Traditional software availability models [16, 39] do not reflect on the above situation very much. We consider two kinds of restoration actions during the operation phase: one involves debugging and the other does not.

The time-dependent behavior in which the system alternates between the operational state (up state) that a system is operating regularly and the restoration state (down state) that a system is inoperable and restored can be modeled by a Markov process [47]. Several stochastic quantities for software availability measurement are derived from this model. Finally, numerical illustrations for software availability analyses are presented.

6.2 Model Description

The following assumptions are made for operational software availability modeling:

A1. The software system breaks down and starts to be restored as soon as a software failure occurs, and the system can not operate until the restoration action is complete.

A2. When a software failure occurs, the restoration action with the debugging activity is performed with probability p $(0 < p < 1)$, while without the debugging activity is performed with probability $q(= 1 - p)$.

A3. The debugging activity is perfect with probability a $(0 < a \leq 1)$, while imperfect with probability $b(= 1 - a)$. We call a the perfect debugging rate. If the debugging activity is perfect, one fault is corrected and removed from the system.

A4. When n faults have been corrected, the time to the next software failure-occurrence and the restoration time with the debugging activity follow exponential distributions with means $1/\lambda_n$ and $1/\mu_n$, respectively.

A5. The restoration time without the debugging activity follows an exponential distribution with mean $1/\eta$.

A6. The probability that two or more software failures occur simultaneously is negligible.

Recall that the process $\{X(t),\ t \geq 0\}$ represents the state of the software system at time point t. The state space of $\{X(t),\ t \geq 0\}$ is defined over again as follows:

W_n: the system is operating.

R_n^1: the system is inoperable and restored with the debugging activity.

R_n^2: the system is inoperable and restored without the debugging activity,

where $n = 0,\ 1,\ 2,\ \ldots$ denotes the cumulative number of corrected faults.

From assumption A2, when the next software failure occurs in $\{X(t) = W_n\}$,

$$X(t) = \begin{cases} R_n^1 & \text{(with probability } p\text{)} \\ R_n^2 & \text{(with probability } q\text{).} \end{cases} \tag{6.1}$$

Furthermore, from assumption A3, when a restoration action with the debugging activity is complete in $\{X(t) = R_n^1\}$,

$$X(t) = \begin{cases} W_n & \text{(with probability } b\text{)} \\ W_{n+1} & \text{(with probability } a\text{).} \end{cases} \tag{6.2}$$

We use Moranda's model [31] to describe the software failure-occurrence phenomenon, which is the same as the description in Chapter 5, i.e. the hazard rate λ_n is given by

$$\lambda_n = Dk^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{6.3}$$

where D and k are the initial hazard rate and the decreasing ratio of the hazard rate, respectively.

We turn to the description of the restoration characteristic. There are various restoration scenarios according to various types of software failures during the operation phase. In this chapter, we pay attention to whether or not the restoration action includes the debugging activity. The restoration rate μ_n for the restoration time with debugging as follows:

$$\mu_n = Er^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ E > 0,\ 0 < r \leq 1), \tag{6.4}$$

where E and r are the initial restoration rate and the decreasing ratio of the restoration rate, respectively and this description has already been discussed in Chapter 5. On the other hand, there are cases where the system is often restored without debugging when it is more important to shorten inoperable time than to adapt the system to operational environment. We assume that it is probabilistic whether or not debugging activity is performed and that the restoration action without debugging is complete randomly throughout the operation phase.

Recall that $Q_{A,B}(\tau)$ $(A,\ B \in \{W_n,\ R_n^1,\ R_n^2;\ n = 0,\ 1,\ 2,\ \ldots\})$ denotes the one-step transition probability that after making a transition into state A, the process $\{X(t),\ t \geq 0\}$ makes a transition into state B by time τ. The expressions for $Q_{A,B}(\tau)$'s are given as follows:

$$Q_{W_n,R_n^1}(\tau) = p(1 - e^{-\lambda_n \tau}), \tag{6.5}$$

$$Q_{W_n,R_n^2}(\tau) = q(1 - e^{-\lambda_n \tau}), \tag{6.6}$$

$$Q_{R_n^1,W_{n+1}}(\tau) = a(1 - e^{-\mu_n \tau}), \tag{6.7}$$

$$Q_{R_n^1,W_n}(\tau) = b(1 - e^{-\mu_n \tau}), \tag{6.8}$$

$$Q_{R_n^2,W_n}(\tau) = 1 - e^{-\eta \tau}. \tag{6.9}$$

The sample state transition diagram of $X(t)$ is illustrated in Fig. 6.1.

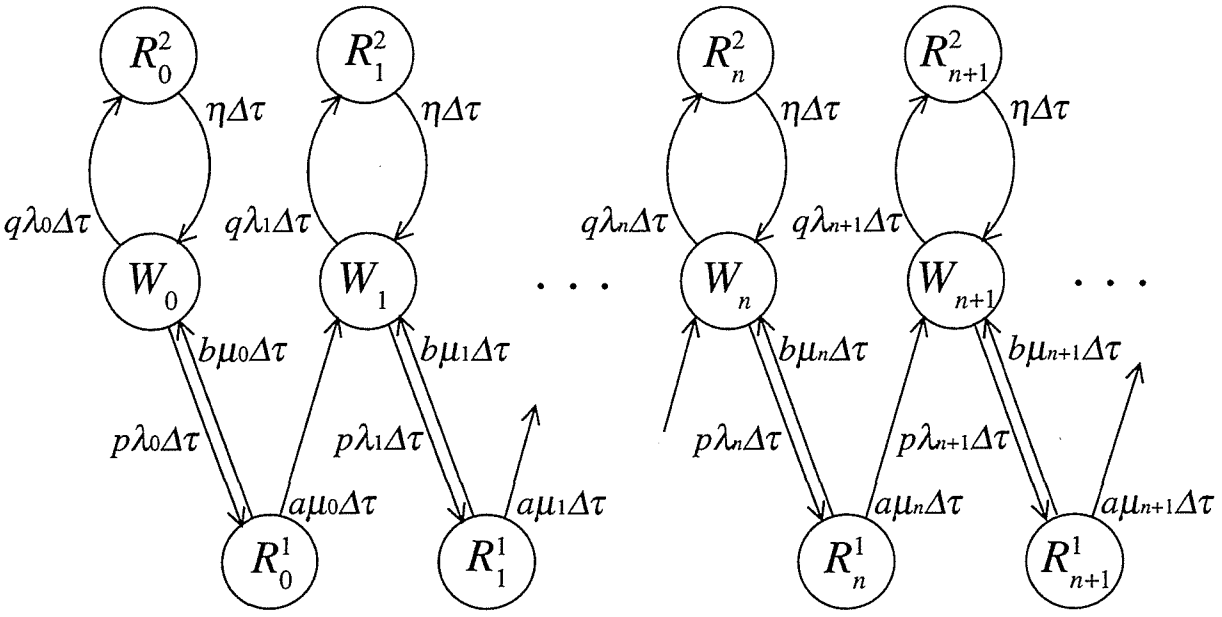

Fig. 6.1. A diagrammatic representation of state transitions between $X(t)$'s for operational software availability modeling (II).

6.3 Derivation of Software Performance Measures

6.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Recall that S_n $(n = 1,\ 2,\ \ldots;\ S_0 \equiv 0)$ denotes the random variable representing the time spent in correcting n faults and that $G_{i,n}(t)$ denotes a distribution function associated with the probability that n faults are corrected in the time interval $(0,\ t]$ on the condition that $i(< n)$ faults have been already corrected at time zero. Then, we obtain the following renewal equation:

$$G_{i,n}(t) = Q_{W_i,R_i^1} * Q_{R_i^1,W_{i+1}} * G_{i+1,n}(t) + Q_{W_i,R_i^1} * Q_{R_i^1,W_i} * G_{i,n}(t) + Q_{W_i,R_i^2} * Q_{R_i^2,W_i} * G_{i,n}(t) \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{6.10}$$

where $*$ denotes a Stieltjes convolution and $G_{n,n}(t) = 1(t)$ (unit function) $(n = 1,\ 2,\ \ldots)$.

Substituting the Laplace-Stieltjes (L-S) transforms of (6.5)–(6.9) into that of (6.10) yields

$$\tilde{G}_{i,n}(s) = \frac{pa\lambda_i\mu_i(s+\eta)}{(s+x_i)(s+y_i)(s+z_i)}\tilde{G}_{i+1,n}(s) \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{6.11}$$

where $-x_i$, $-y_i$, and $-z_i$ are the distinct roots of the following third order equation:

$$s^3 + (\eta + \lambda_i + \mu_i)s^2 + [p\eta\lambda_i + \eta\mu_i + (1 - pb)\lambda_i\mu_i]s + pa\eta\lambda_i\mu_i = 0. \tag{6.12}$$

Solving (6.11) recursively, we obtain the L-S transform of $G_{0,n}(t)$ as

$$\begin{aligned}\tilde{G}_{0,n}(s) &= \prod_{i=0}^{n-1} \frac{pa\lambda_i\mu_i(s+\eta)}{(s+x_i)(s+y_i)(s+z_i)} \\ &= \sum_{i=0}^{n-1} \left(\frac{A^1_{n,i}x_i}{s+x_i} + \frac{A^2_{n,i}y_i}{s+y_i} + \frac{A^3_{n,i}\mu_i}{s+z_i} \right),\end{aligned} \tag{6.13}$$

where constant coefficients $A^1_{n,i}$, $A^2_{n,i}$, and $A^3_{n,i}$ are given by

$$A^1_{n,i} = \frac{\prod_{j=0}^{n-1} pa\lambda_j\mu_j(\eta - x_i)}{x_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(x_j - x_i) \prod_{j=0}^{n-1}(y_j - x_i)(z_j - x_i)} \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{6.14}$$

$$A^2_{n,i} = \frac{\prod_{j=0}^{n-1} pa\lambda_j\mu_j(\eta - y_i)}{y_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(y_j - y_i) \prod_{j=0}^{n-1}(z_j - y_i)(x_j - y_i)} \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{6.15}$$

$$A^3_{n,i} = \frac{\prod_{j=0}^{n-1} pa\lambda_j\mu_j(\eta - z_i)}{z_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(z_j - z_i) \prod_{j=0}^{n-1}(x_j - z_i)(y_j - z_i)} \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{6.16}$$

respectively. By inverting (6.13) and rewriting $G_{0,n}(t)$ as $G_n(t)$, we obtain the distribution function for S_n as

$$\begin{aligned}G_n(t) &\equiv \Pr\{S_n \leq t\} \\ &= 1 - \sum_{i=0}^{n-1}(A^1_{n,i}e^{-x_it} + A^2_{n,i}e^{-y_it} + A^3_{n,i}e^{-z_it}) \\ &\quad (n = 1,\ 2,\ \ldots;\ G_0(t) \equiv \mathbf{1}(t)),\end{aligned} \tag{6.17}$$

where we postulate $\prod_{\substack{j=0 \\ j\neq 0}}^{0} \cdot = 1$. It is noted that

$$\sum_{i=0}^{n-1}(A^1_{n,i} + A^2_{n,i} + A^3_{n,i}) = 1 \quad (n \geq 1). \tag{6.18}$$

6.3.2 Operational State Occupancy Probability and Software Availability

Recall that $P_{i,n}(t)$ denotes the conditional state occupancy probability that the system is operating at time point t when n faults have been corrected on the condition that the system was operating at time point zero when i faults had been already corrected, i.e.

$$P_{i,n}(t) \equiv \Pr\{X(t) = W_n | X(0) = W_i\} \qquad (i = 0,\ 1,\ 2,\ \ldots,\ n), \tag{6.19}$$

and that $P_n(t) \equiv P_{0,n}(t)$ is called the operational state occupancy probability. Then, we obtain the following renewal equations:

$$P_n(t) = G_n * P_{n,n}(t), \tag{6.20}$$

$$P_{n,n}(t) = e^{-\lambda_n t} + Q_{W_n,R_n^1} * Q_{R_n^1,W_n} * P_{n,n}(t) + Q_{W_n,R_n^2} * Q_{R_n^2,W_n} * P_{n,n}(t). \tag{6.21}$$

From (6.21), the L-S transform of $P_{n,n}(t)$ is given by

$$\begin{aligned} \widetilde{P}_{n,n}(s) &= \frac{s(s+\mu_n)(s+\eta)}{(s+x_n)(s+y_n)(s+z_n)} \\ &= \left(\frac{s}{pa\lambda_n} + \frac{s^2}{pa\lambda_n\mu_n}\right) \frac{pa\lambda_n\mu_n(s+\eta)}{(s+x_n)(s+y_n)(s+z_n)}. \end{aligned} \tag{6.22}$$

Substituting (6.22) into the L-S transform of (6.20) yields

$$\widetilde{P}_n(s) = \frac{s}{pa\lambda_n}\widetilde{G}_{n+1}(s) + \frac{s^2}{pa\lambda_n\mu_n}\widetilde{G}_{n+1}(s). \tag{6.23}$$

By inverting (6.23), the operational state occupancy probability is obtained as

$$\begin{aligned} P_n(t) &\equiv \Pr\{X(t) = W_n\} \\ &= \frac{g_{n+1}(t)}{pa\lambda_n} + \frac{g'_{n+1}(t)}{pa\lambda_n\mu_n}, \end{aligned} \tag{6.24}$$

where $g_n(t)$ is the probability density function of random variable S_n and $g'_n(t) \equiv dg_n(t)/dt$.

The instantaneous software availability is defined as

$$A(t) \equiv \sum_{n=0}^{\infty} P_n(t), \tag{6.25}$$

which represents the probability that the software system is operable at specified time point t. Furthermore, the average software availability in the time interval $(0,\ t]$ is defined as

$$A_{av}(t) \equiv \frac{1}{t}\int_0^t A(x)dx, \tag{6.26}$$

which represents the ratio of system's operating time to the time interval $(0,\ t]$. Using (6.24), we can describe (6.25) and (6.26) as

$$A(t) = \sum_{n=0}^{\infty} \left[\frac{g_{n+1}(t)}{pa\lambda_n} + \frac{g'_{n+1}(t)}{pa\lambda_n\mu_n} \right], \tag{6.27}$$

$$A_{av}(t) = \frac{1}{t} \sum_{n=0}^{\infty} \left[\frac{G_{n+1}(t)}{pa\lambda_n} + \frac{g_{n+1}(t)}{pa\lambda_n\mu_n} \right], \tag{6.28}$$

respectively.

6.4 Numerical Examples

Using the operational software availability model discussed above, we show numerical illustrations for software availability measurement and assessment.

The instantaneous software availabilities, $A(t)$'s in (6.27) and the average software availabilities, $A_{av}(t)$'s in (6.28) for various perfect debugging rates, a's are shown in Figs. 6.2 and 6.3, respectively. These figures indicate that software availability drops rapidly immediately after operation and gradually increases after. This tells us that these measures can show unstableness of system performance in the early stage of the operation phase quantitatively. These figures also indicate that the increase of the certainty of debugging brings high software availability.

$A(t)$'s and $A_{av}(t)$'s for various values of p, which represents the probability that a debugging activity is performed when a software failure occurs, are shown in Figs. 6.4 and 6.5, respectively. We can see that software availability is lower in the early stage of the operation phase but more improves with the lapse of time as p increases. This reason is that software reliability growth occurs even during the operation phase though the restoration time tends to be longer.

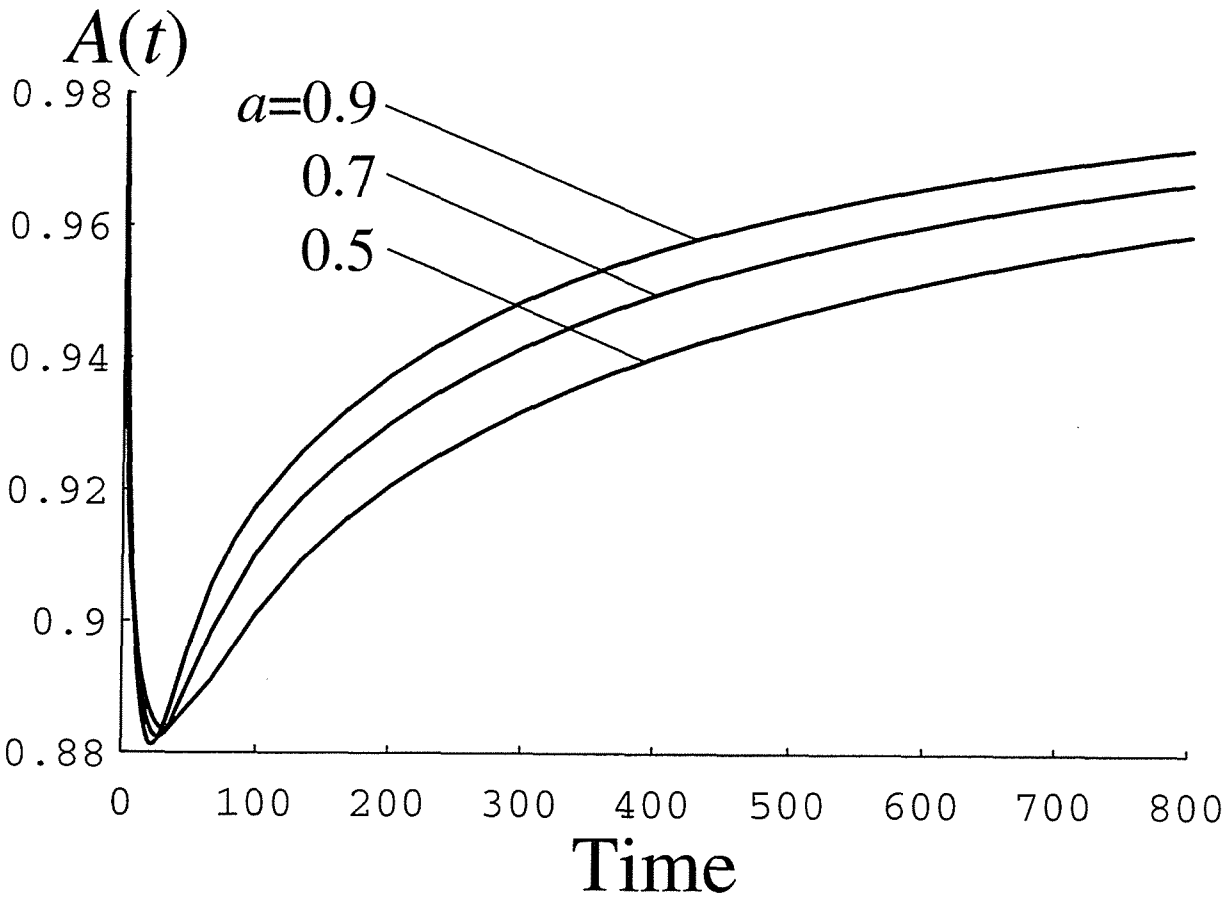

Fig. 6.2. Dependence of a on $A(t)$ ($p = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.5$, $r = 0.9$, $\eta = 1.0$).

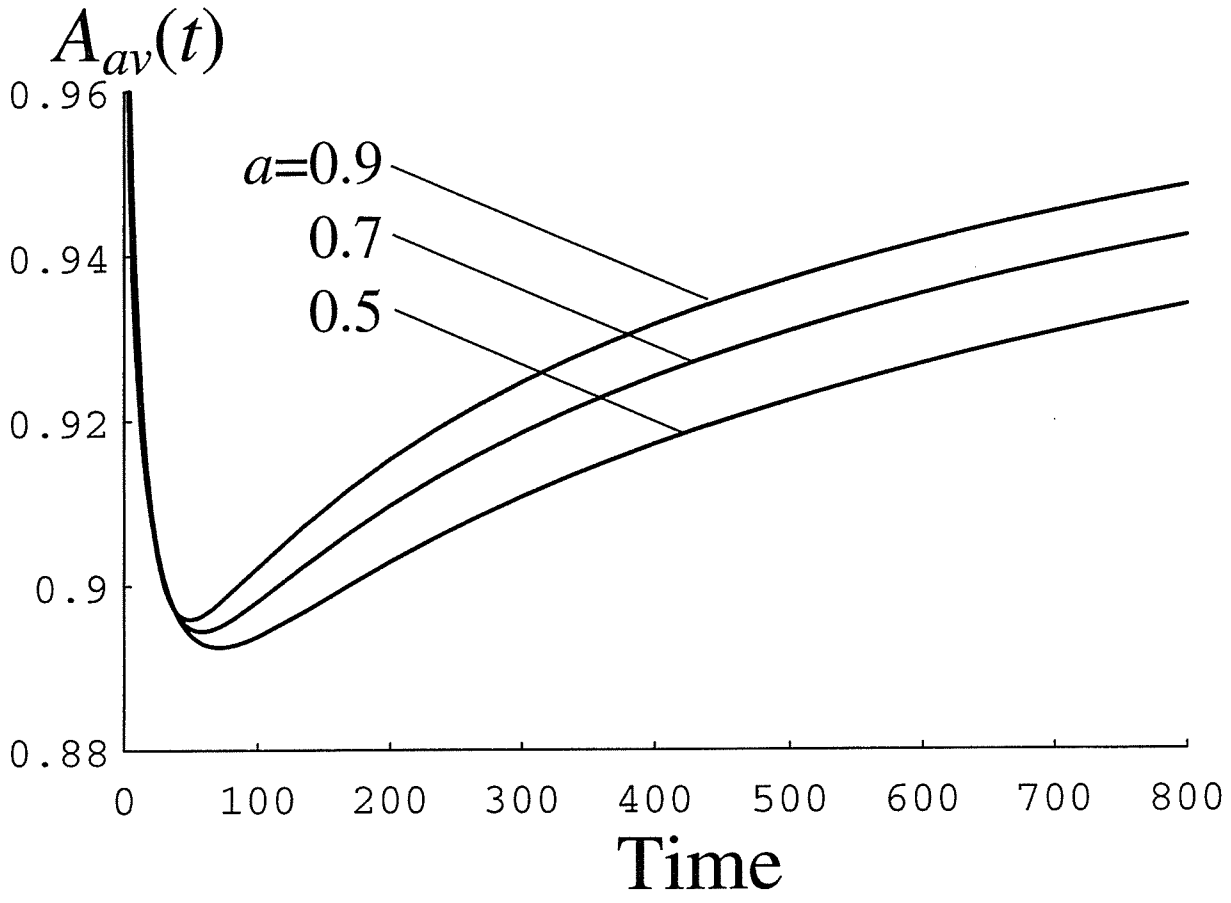

Fig. 6.3. Dependence of a on $A_{av}(t)$ ($p = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.5$, $r = 0.9$, $\eta = 1.0$).

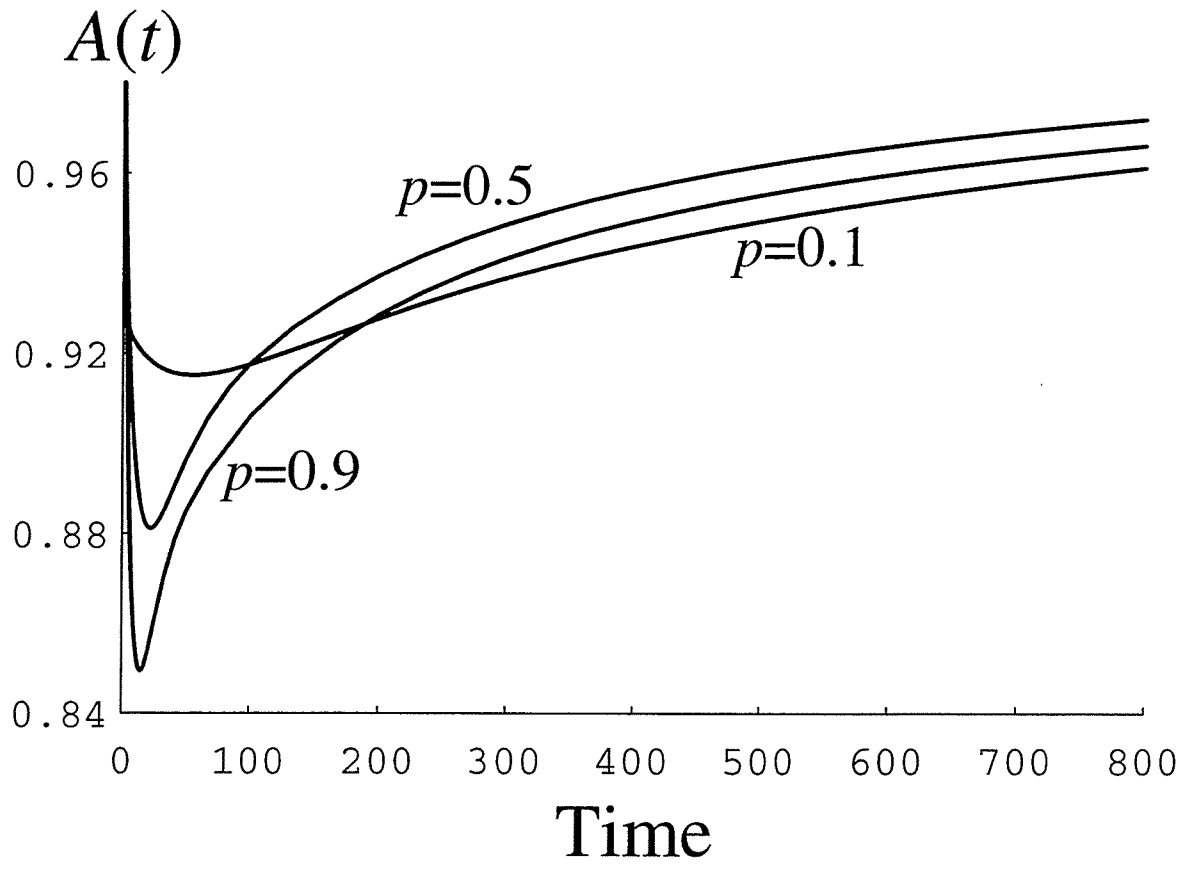

Fig. 6.4. Dependence of p on $A(t)$ ($a = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.5$, $r = 0.9$, $\eta = 1.0$).

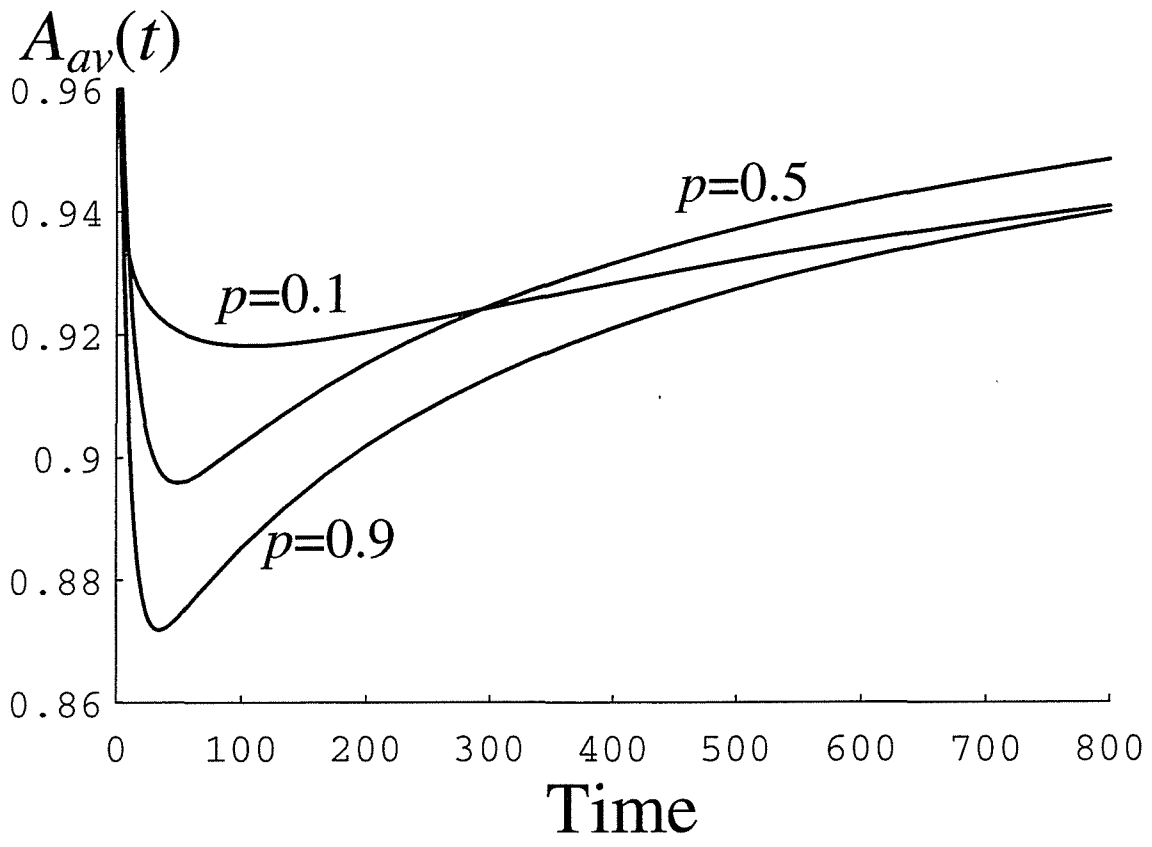

Fig. 6.5. Dependence of p on $A_{av}(t)$ ($a = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.5$, $r = 0.9$, $\eta = 1.0$).

6.5 Concluding Remarks

In this chapter, we have developed a software availability model considering two different kinds of restoration actions performed during the operation phase. The dynamic behavior of the software system has been described by a Markov process. Several useful quantitative measures for software performance assessment have been derived. Numerical illustrations for software availability measurement and assessment have been also presented to show that these measures are very useful for operational software performance evaluation.

Chapter 7

Software Availability Modeling with Performance Degeneration

7.1 Introduction

Software reliability is one of the most important software attributes in measuring software quality characteristics. A mathematical model for software reliability measurement is called a software reliability growth model which describes a software fault-detection or a software failure-occurrence phenomenon during the testing phase in the software development process and the operation phase. A software failure is defined as an unacceptable departure from program operation caused by a fault remaining in the system. A number of software reliability growth models have been proposed for the last a few decades [25, 26, 43].

The traditional software reliability assessment measures, such as the expected residual fault content and the mean time between software failures, are the developer-oriented ones. However, the customers take a great interest in the performance-related reliability measures of software systems. For the purpose of deriving such measures, we need to construct software reliability growth models incorporating maintainability and/or performance. One of the customer-oriented metrics is the software availability which is defined as the probability that the software system is performing successfully at a specified time point. Several stochastic models for measuring software availability have been proposed [19, 21]. Several reliability/performance evaluation measures for computer systems from the viewpoint of hardware configurations have also been proposed. Beaudry [3] has proposed the computation availability and the mean computation between failures for fault-tolerant computing systems. Meyer [29] has proposed the performability taking account of accomplishment levels from customer's viewpoint. Sols [51] has introduced the concept of degraded availability.

These studies consider that systems have several different performance levels. However, few quantitative measures considering simultaneous software reliability and performance have been proposed.

In this chapter, we develop a plausible software availability model considering the degeneration of system performance in user operation. Software availability models proposed so far assume only up and down state, taking software reliability growth process in consideration. But software systems could not always display their full performance when they are available in actual operational environment. Though systems do not fall into operation stoppage outwardly, some internal parts of systems may be unfavorable states. For instance, the system throughput decreases due to the concentration of loads into some specified system resources and some parts of system functions are unavailable due to maintenance of the corresponding software subcomponents. We assume that the software system has two operational states during the operation phase: one is providing with full performance and the other is with degenerated performance. The time-dependent behavior of the software system, which alternates between the operational and restoration state, is described by a Markov process. Then software reliability growth process is also incorporated into this model. Several quantitative availability/performance measures are derived analytically from this model. In particular, this model can derive computation software availability which is a measure taking account of reliability and performance simultaneously. This metric is defined as the expected amount of possible computation per unit time at a specified time point. Finally, numerical examples are presented for illustration of software availability/performance measurement and assessment.

7.2 Model Description

The following assumptions are made for software availability modeling:

A1. When the software system is operating, the time-interval of operation with full performance and the holding time of performance degeneration follow exponential distributions with means $1/\theta$ and $1/\eta$, respectively.

A2. The software system breaks down and starts to be restored as soon as a software failure occurs, and the system can not operate until the restoration action is complete.

A3. The restoration action implies the debugging activity and software reliability growth occurs if a debugging activity is perfect.

A4. The debugging activity is perfect with probability a $(0 < a \leq 1)$, while imperfect with probability $b(= 1 - a)$. We call a the perfect debugging rate. A perfect debugging activity corrects and removes one fault from the system.

A5. When n faults have been corrected, the next software failure-occurrence time-interval and the restoration time follow exponential distributions with means $1/\lambda_n$ and $1/\mu_n$, respectively.

A6. The probability that two or more software failures occur simultaneously is negligible.

We define the computation capacity as the amount of useful computation per unit time [3]. It is assume that the computation capacities are $C(> 0)$ and $C\delta$ $(0 < \delta < 1)$ when the system is operating with full and degenerated performance, respectively. We call δ the decreasing ratio of system's computation capacity.

We introduce a stochastic process $\{X(t),\ t \geq 0\}$ representing the state of the software system at time point t. The state space of the process $\{X(t),\ t \geq 0\}$ is defined as follows:

W_n: the system is operating with full performance,

L_n: the system is operating with degenerated performance,

R_n: the system is inoperable and restored,

where $n = 0,\ 1,\ 2,\ \ldots$ denotes the cumulative number of faults corrected during the operation phase.

From assumption A4, when a restoration action is complete in $\{X(t) = R_n\}$,

$$X(t) = \begin{cases} W_n & \text{(with probability } b\text{)} \\ W_{n+1} & \text{(with probability } a\text{).} \end{cases} \tag{7.1}$$

The descriptions of λ_n and μ_n are given by

$$\lambda_n = Dk^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{7.2}$$

$$\mu_n = Er^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ E > 0,\ 0 < r \leq 1), \tag{7.3}$$

respectively where D and k are the initial hazard rate and the decreasing ratio of the hazard rate, respectively and E and r are the initial restoration rate and the decreasing ratio of the restoration rate, respectively. The details of λ_n and μ_n have been discussed in Chapters 2, 4, and 5.

Let $Q_{A,B}(\tau)$ $(A,\ B \in \{W_n,\ L_n,\ R_n;\ n = 0,\ 1,\ 2,\ \ldots\})$ denote the one-step transition probability that after making a transition into state A, the process $\{X(t),\ t \geq 0\}$ makes a transition into state B by time τ. The expressions for $Q_{A,B}(\tau)$'s are given as follows:

$$Q_{W_n,L_n}(\tau) = \frac{\theta}{\lambda_n + \theta}[1 - e^{-(\lambda_n+\theta)\tau}], \tag{7.4}$$

$$Q_{W_n,R_n}(\tau) = \frac{\lambda_n}{\lambda_n + \theta}[1 - e^{-(\lambda_n+\theta)\tau}], \tag{7.5}$$

$$Q_{L_n,W_n}(\tau) = \frac{\eta}{\lambda_n + \eta}[1 - e^{-(\lambda_n+\eta)\tau}], \tag{7.6}$$

$$Q_{L_n,R_n}(\tau) = \frac{\lambda_n}{\lambda_n + \eta}[1 - e^{-(\lambda_n+\eta)\tau}], \tag{7.7}$$

$$Q_{R_n,W_{n+1}}(\tau) = a(1 - e^{-\mu_n\tau}), \tag{7.8}$$

$$Q_{R_n,W_n}(\tau) = b(1 - e^{-\mu_n\tau}). \tag{7.9}$$

The sample state transition diagram of $X(t)$ is illustrated in Fig. 7.1.

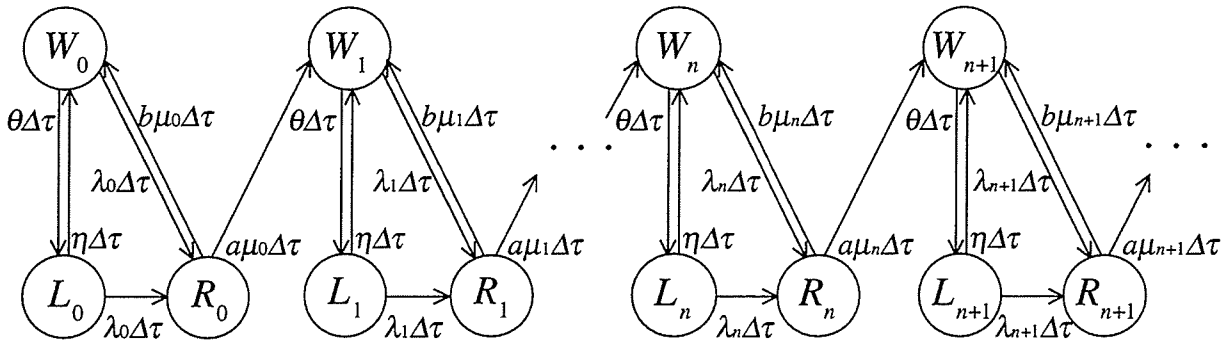

Fig. 7.1. A diagrammatic representation of state transitions between $X(t)$'s for software availability modeling with performance degeneration.

7.3 Software Availability Analysis

7.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Let $T_{i,i+1}$ and $G_{i,i+1}(t)$ be the random variable representing the time spent in making a transition from state W_i to state W_{i+1} and the distribution function of $T_{i,i+1}$, respectively. Then, we obtain the following renewal equations:

$$\left.\begin{aligned} G_{i,i+1}(t) &= H_{W_i,R_i} * Q_{R_i,W_{i+1}}(t) \\ &\quad + H_{W_i,R_i} * Q_{R_i,W_i} * G_{i,i+1}(t) \\ H_{W_i,R_i}(t) &= Q_{W_i,R_i}(t) + Q_{W_i,L_i} * Q_{L_i,R_i}(t) \\ &\quad + Q_{W_i,L_i} * Q_{L_i,W_i} * H_{W_i,R_i}(t) \\ &\quad (i = 0,\ 1,\ 2,\ \ldots) \end{aligned}\right\}, \tag{7.10}$$

where $*$ denotes the Stieltjes convolution and $H_{W_i,R_i}(t)$ represents the probability that the process $X(t)$ makes a transition from state W_i to state R_i in an amount of time less than or equal to t.

Substituting the Laplace-Stieltjes (L-S) transforms of (7.4)–(7.9) into that of (7.10) yields

$$\tilde{G}_{i,i+1}(s) = \frac{x_i y_i}{(s + x_i)(s + y_i)}, \tag{7.11}$$

where

$$\left.\begin{matrix} x_i \\ y_i \end{matrix}\right\} = \frac{1}{2}\left[(\lambda_i + \mu_i) \pm \sqrt{(\lambda_i + \mu_i)^2 - 4a\lambda_i\mu_i}\right]$$
$$\text{(double signs in same order).} \tag{7.12}$$

Let S_n and $G_n(t)$ $(n = 1,\ 2,\ \ldots;\ S_0 \equiv 0)$ be the random variable representing the time spent in correcting n faults and the distribution function of S_n, respectively. Then, the following relation are obtained:

$$S_n = \sum_{i=0}^{n-1} T_{i,i+1}. \tag{7.13}$$

Noting that $T_{i,i+1}$'s are mutually independent, we can get the L-S transform of $G_n(t)$ as

$$\begin{aligned}\tilde{G}_n(s) &= \prod_{i=0}^{n-1} \tilde{G}_{i,i+1}(s) \\ &= \prod_{i=0}^{n-1} \frac{x_i y_i}{(s+x_i)(s+y_i)} \\ &= \sum_{i=0}^{n-1} \left(\frac{A_{n,i}^1 x_i}{s+x_i} + \frac{A_{n,i}^2 y_i}{s+y_i} \right),\end{aligned} \tag{7.14}$$

where constant coefficients $A_{n,i}^1$ and $A_{n,i}^2$ are given by

$$A_{n,i}^1 = \frac{\prod_{j=0}^{n-1} x_j y_j}{x_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1} (x_j - x_i) \prod_{j=0}^{n-1} (y_j - x_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{7.15}$$

$$A_{n,i}^2 = \frac{\prod_{j=0}^{n-1} x_j y_j}{y_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1} (y_j - y_i) \prod_{j=0}^{n-1} (x_j - y_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{7.16}$$

respectively. By inverting (7.14), we have the distribution function of S_n as

$$\begin{aligned}G_n(t) &\equiv \Pr\{S_n \leq t\} \\ &= 1 - \sum_{i=0}^{n-1} (A_{n,i}^1 e^{-x_i t} + A_{n,i}^2 e^{-y_i t})\end{aligned}$$

$$(n = 1,\ 2,\ \ldots;\ G_0(t) \equiv \mathbf{1}(t)\ (\text{unit fuction})), \tag{7.17}$$

where we postulate $\prod_{\substack{j=0 \\ j\neq 0}}^{0} \cdot = 1$ for $n = 1$. It is noted that (7.17) is identical to the model discussed in Chapter 4 and has no bearing on parameters θ and η, which are related to performance degeneration and that

$$\sum_{i=0}^{n-1} (A_{n,i}^1 + A_{n,i}^2) = 1 \quad (n = 1,\ 2,\ \ldots). \tag{7.18}$$

Furthermore, the mean and the variance of S_n are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1} \left(\frac{1}{x_i} + \frac{1}{y_i} \right), \tag{7.19}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1} \left(\frac{1}{{x_i}^2} + \frac{1}{{y_i}^2} \right), \tag{7.20}$$

respectively.

7.3.2 State Occupancy Probability

Let $P_{A,B}(t)$ be the conditional state occupancy probability that $X(t)$ is in state B at time point t on the condition that $X(t)$ was in state A at time point zero, i.e.,

$$P_{A,B}(t) \equiv \Pr\{X(t) = B | X(0) = A\}$$
$$(A,\ B \in \{W_n,\ L_n,\ R_n;\ n = 0,\ 1,\ 2,\ \ldots\}), \tag{7.21}$$

and $P_{W_n}(t) \equiv P_{W_0,W_n}(t)$, $P_{L_n}(t) \equiv P_{W_0,L_n}(t)$, and $P_{R_n}(t) \equiv P_{W_0,R_n}(t)$ be the state occupancy probabilities that $X(t)$ is in states W_n, L_n, and R_n at time point t, respectively.

At first, we obtain the following renewal equations with respect of $P_{W_n}(t)$:

$$P_{W_n}(t) = G_n * P_{W_n,W_n}(t), \tag{7.22}$$

$$\begin{aligned} P_{W_n,W_n}(t) = {} & e^{-(\lambda_n+\theta)t} + Q_{W_n,L_n} * Q_{L_n,W_n} * P_{W_n,W_n}(t) \\ & + Q_{W_n,R_n} * Q_{R_n,W_n} * P_{W_n,W_n}(t) \\ & + Q_{W_n,L_n} * Q_{L_n,R_n} * Q_{R_n,W_n} * P_{W_n,W_n}(t). \end{aligned} \tag{7.23}$$

Then, the L-S transform of $P_{W_n}(t)$ is obtained as

$$\begin{aligned} \widetilde{P}_{W_n}(s) = {} & \frac{s(s+\lambda_n+\eta)(s+\mu_n)}{(s+\lambda_n+\theta+\eta)(s+x_n)(s+y_n)} \\ & \times \prod_{i=0}^{n-1} \frac{x_i y_i}{(s+x_i)(s+y_i)}. \end{aligned} \tag{7.24}$$

By inverting (7.24), we have $P_{W_n}(t)$ as

$$\begin{aligned} P_{W_n}(t) & \equiv \Pr\{X(t) = W_n\} \\ & = B^n e^{-(\lambda_n+\theta+\eta)t} + \sum_{i=0}^{n} (B^1_{n,i} e^{-x_i t} + B^2_{n,i} e^{-y_i t}) \\ & \quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{7.25}$$

where constant coefficients B^n, $B^1_{n,i}$, and $B^2_{n,i}$ are given by

$$B^n = \frac{-\theta(\mu_n - \lambda_n - \theta - \eta)\prod_{j=0}^{n-1} x_j y_j}{\prod_{j=0}^{n}(x_j - \lambda_n - \theta - \eta)(y_j - \lambda_n - \theta - \eta)} \quad (n = 0,\ 1,\ 2,\ \ldots), \tag{7.26}$$

$$B^1_{n,i} = \frac{(\lambda_n + \eta - x_i)(\mu_n - x_i)\prod_{j=0}^{n-1} x_j y_j}{(\lambda_n + \theta + \eta - x_i)\prod_{\substack{j=0 \\ j\neq i}}^{n}(x_j - x_i)\prod_{j=0}^{n}(y_j - x_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n), \tag{7.27}$$

$$B^2_{n,i} = \frac{(\lambda_n + \eta - y_i)(\mu_n - y_i)\prod_{j=0}^{n-1} x_j y_j}{(\lambda_n + \theta + \eta - y_i)\prod_{\substack{j=0 \\ j\neq i}}^{n}(y_j - y_i)\prod_{j=0}^{n}(x_j - y_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n), \tag{7.28}$$

respectively. It is noted that

$$\left.\begin{aligned} &B^0_0 + B^1_{0,0} + B^2_{0,0} = 1 \\ &B^0_n + \sum_{i=0}^{n}\left(B^1_{n,i} + B^2_{n,i}\right) = 0 \quad (n = 1,\ 2,\ \ldots) \end{aligned}\right\}. \tag{7.29}$$

Next, we obtain the following renewal equations with respect of $P_{R_n}(t)$:

$$P_{R_n}(t) = G_n * H_{W_n,R_n} * P_{R_n,R_n}(t), \tag{7.30}$$

$$P_{R_n,R_n}(t) = e^{-\mu_n t} + Q_{R_n,W_n} * H_{W_n,R_n} * P_{R_n,R_n}(t). \tag{7.31}$$

From (7.30) and (7.31), the L-S transform of $P_{R_n}(t)$ is obtained as

$$\begin{aligned} \widetilde{P}_{R_n}(s) &= \frac{s}{a\mu_n} \cdot \frac{a\lambda_n\mu_n}{(s + x_n)(s + y_n)} \cdot \widetilde{G}_n(s) \\ &= \frac{s}{a\mu_n}\widetilde{G}_{n+1}(s). \end{aligned} \tag{7.32}$$

Then, we have $P_{R_n}(t)$ as

$$\begin{aligned} P_{R_n}(t) &\equiv \Pr\{X(t) = R_n\} \\ &= \frac{1}{a\mu_n} g_{n+1}(t) \quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{7.33}$$

where $g_n(t)$ denotes the probability density function of S_n, i.e. $g_n(t) \equiv dG_n(t)/dt$.

Considering the stochastic process $\{Y(t),\ t \geq 0\}$ representing the cumulative number of faults corrected up to time t, we have the following equivalent relation:

$$\{Y(t) = n\} \iff \{X(t) = W_n\} \cup \{X(t) = L_n\} \cup \{X(t) = R_n\}, \tag{7.34}$$

Furthermore, since $\{Y(t),\ t \geq 0\}$ is a counting process,

$$\{S_n \leq t\} \iff \{Y(t) \geq n\}, \tag{7.35}$$

then,

$$\begin{aligned} P_n(t) &\equiv \Pr\{Y(t) = n\} \\ &= G_n(t) - G_{n+1}(t) \quad (n = 0,\ 1,\ 2,\ \ldots). \end{aligned} \tag{7.36}$$

Therefore, we have $P_{L_n}(t)$ as

$$\begin{aligned} P_{L_n}(t) &\equiv \Pr\{X(t) = L_n\} \\ &= G_n(t) - G_{n+1}(t) - P_{W_n}(t) - P_{R_n}(t) \\ &\quad (n = 0,\ 1,\ 2,\ \ldots). \end{aligned} \tag{7.37}$$

7.3.3 Instantaneous Software Availability and Computation Software Availability

The following identical equation holds for arbitrary time t:

$$\sum_{n=0}^{\infty} [P_{W_n}(t) + P_{L_n}(t) + P_{R_n}(t)] = 1. \tag{7.38}$$

The instantaneous software availability is defined as

$$A(t) \equiv \sum_{n=0}^{\infty} [P_{W_n}(t) + P_{L_n}(t)], \tag{7.39}$$

which represents the probability that the software system is operable at time point t. Using (7.38), we can describe $A(t)$ as

$$\begin{aligned} A(t) &= 1 - \sum_{n=0}^{\infty} P_{R_n}(t) \\ &= 1 - \sum_{n=0}^{\infty} \frac{g_{n+1}(t)}{a\mu_n}. \end{aligned} \tag{7.40}$$

It is noted that $A(t)$ has no bearing on parameters θ and η.

Furthermore, the computation software availability [3, 37, 41] is defined as

$$A_c(t) \equiv C \sum_{n=0}^{\infty} \left[P_{W_n}(t) + \delta P_{L_n}(t) \right], \tag{7.41}$$

which represents the expected value of the computation capacity of the system at time point t.

7.4 Numerical Examples

Using the operational software availability model discussed above, we show numerical illustrations for software availability/performance measurement and assessment.

The instantaneous software availabilities, $A(t)$'s in (7.40) and the computation software availabilities, $A_c(t)$'s in (7.41) for various perfect debugging rates, a's, are shown in Figs. 7.2 and 7.3, respectively. From these figures, we can observe that these quantities drop rapidly immediately after operation and gradually increases after. These tell us that these measures can show unstableness of software performance in the early stage of the operation phase quantitatively. These figures also indicate that the increase of the certainty of debugging brings high software performance.

$A_c(t)$'s for various values of δ, which represents the decreasing ratio. of the computation capacity, are shown in Fig. 7.4. We can see that the software performance becomes lower when δ is estimated smaller.

$A_c(t)$'s for various values of η, which is the parameter related with the holding time of performance degeneration, are shown in Fig. 7.5. This figure indicates that the software performance remains lower with decreasing η since smaller η means that the system tends to keep the degenerated operational state longer.

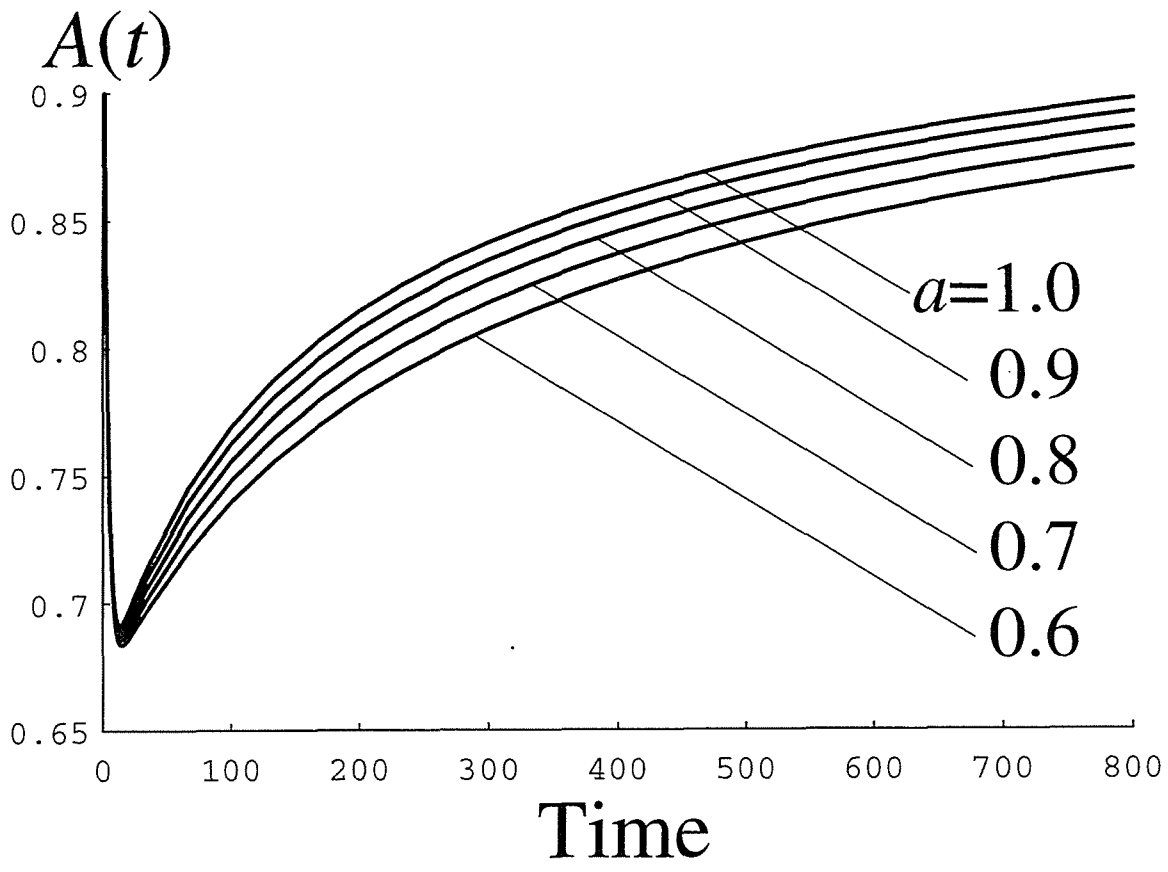

Fig. 7.2. Dependence of a on $A(t)$ ($D = 0.1,\ k = 0.8,\ E = 0.2,\ r = 0.9$).

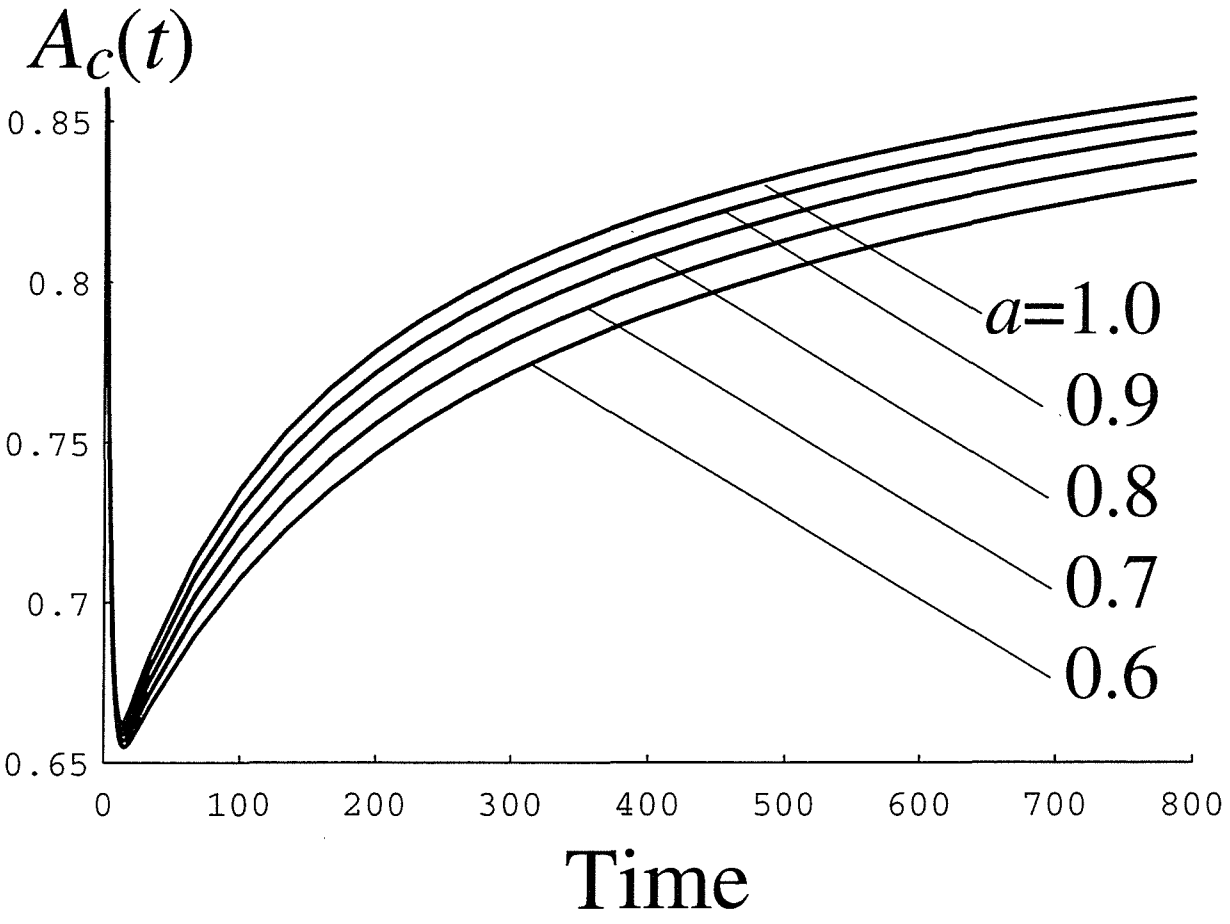

Fig. 7.3. Dependence of a on $A_c(t)$ ($D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.1$, $\eta = 1.0$, $C = 1.0$, $\delta = 0.5$).

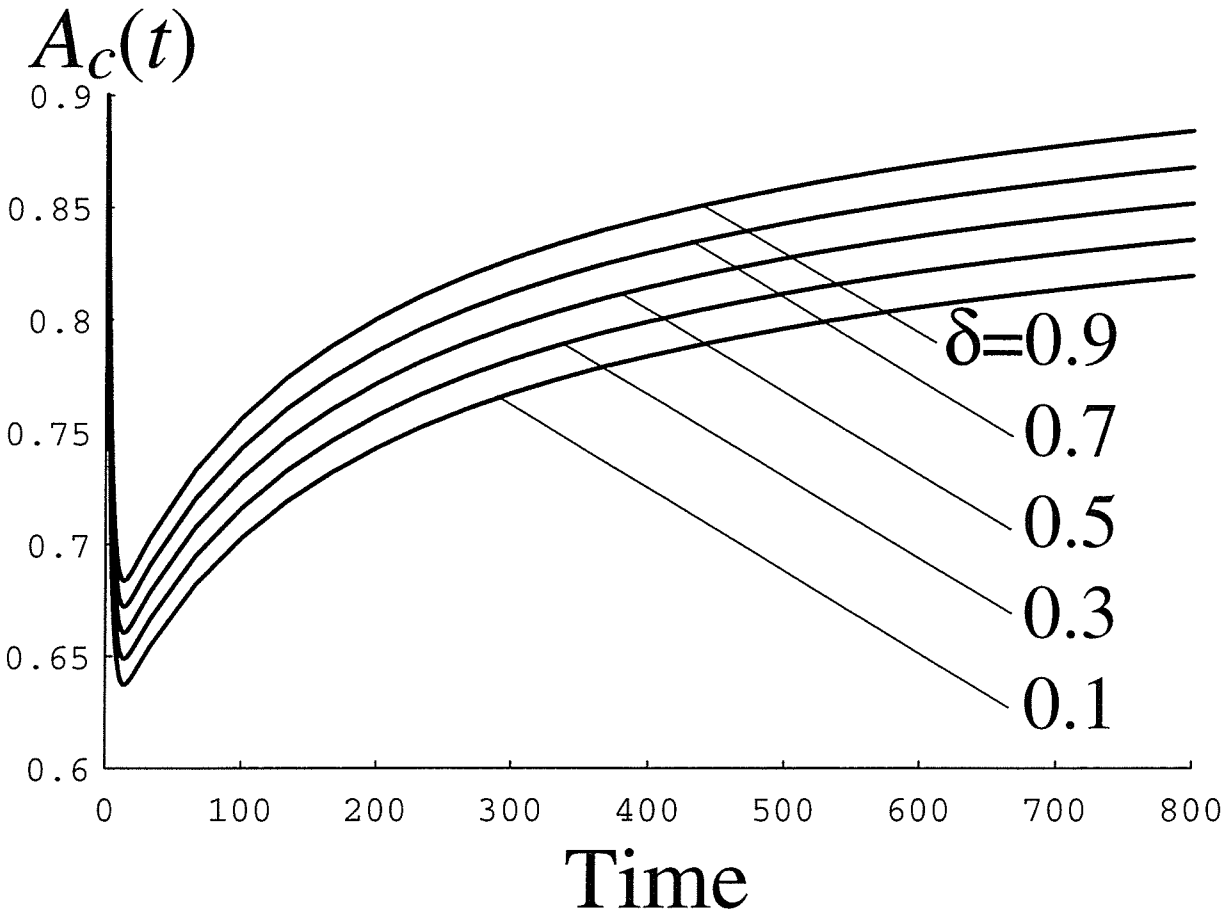

Fig. 7.4. Dependence of δ on $A_c(t)$ ($a = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.1$, $\eta = 1.0$, $C = 1.0$).

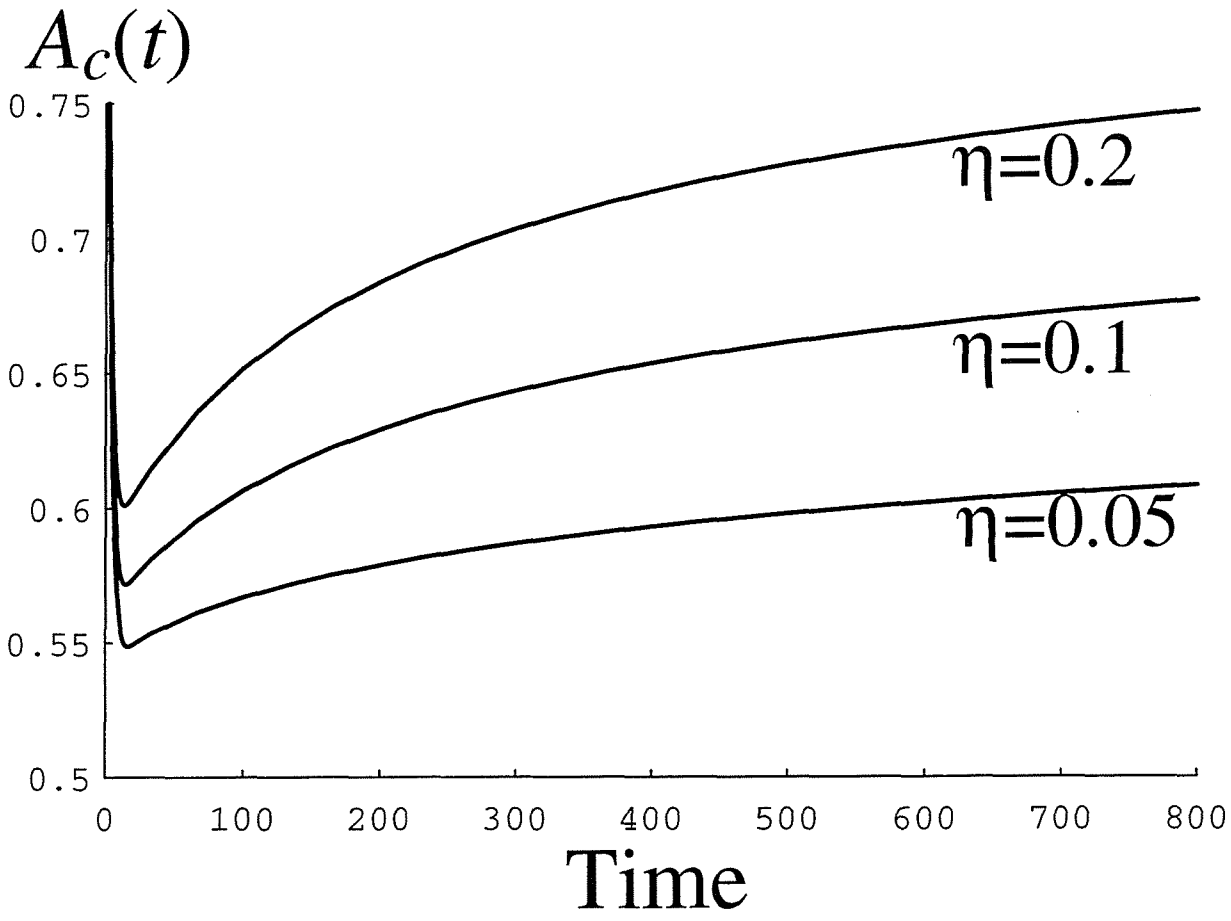

Fig. 7.5. Dependence of η on $A_c(t)$ ($a = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.1$, $C = 1.0$, $\delta = 0.5$).

7.5 Concluding Remarks

In this chapter, we have discussed the software availability modeling with two different operational levels: one is the operational state providing with full performance and the other is providing with degenerated performance, describing the software reliability growth process. Several stochastic quantities for software reliability/performance measurement have been derived from this model. In particular, it is meaningful that a new performance assessment measure for software systems considering both reliability and computation has been provided. Finally, their numerical examples have been illustrated.

Chapter 8

Availability Modeling for Hardware-Software System

8.1 Introduction

Recently, hardware and software components constituting a computer system have been designed not separately, but synchronously with considering each other. The design method considering the trade-off between hardware and software components is called hardware/software co-design [30]. Hardware/software co-design (hereafter referred to as co-design) is not a new concept, but it has received much attention since computer systems have grown in size and complexity and both of hardware and software systems have to be designed in order to use the mutual maximum performances. The concept of co-design is also important in system quality/performance measurement and assessment.

Generally, hardware systems fail due to wear out and/or deterioration and are renewed by repair or replacement of failed parts. Therefore, it is often assumed that the hardware failure- and maintenance-characteristics are not concerned with the numbers of failures and/or repair activities. On the other hand, software systems fail due to the defects or the mistakes latent in the software programs. A statement or a part of a statement in a program that causes a software failure is called a software fault. In other words, a software failure is defined as an unacceptable departure from program operation caused by a software fault (hereafter referred to as a fault) remaining in the software system. Furthermore, the restoration action for software systems includes not only the cause analysis, the data recovery, and the program reload but also the debugging activities for manifested faults. Then, the perfect debugging activity improves software reliability. Accordingly, the software failure- and maintenance-characteristics depend on the cumulative numbers

of software failures and/or fault corrections. Several software availability models involving software reliability growth have been proposed [19, 21].

In this chapter, we discuss an availability model considering hardware and software systems simultaneously [5, 6, 12]. The system is assumed to consist of one hardware and one software subsystems. The failure-occurrence phenomena of the hardware and the software subsystem are described by a constant and a geometrically decreasing hazard rate, respectively. The time-dependent behavior of the hardware-software system (hereafter referred to as a system), which alternates between the operational state (up state) that a system is operating regularly and the restoration state (down state) that a system is inoperable and restored, can be modeled by a Markov process [47]. The description of system availability modeling is discussed in Section 8.2. Several stochastic quantities for system performance measurement are derived in Section 8.3. Numerical illustrations for system reliability/availability analysis and measurement are presented in Section 8.4.

8.2 Model Description

The following assumptions are made for availability modeling for the system:

A1. The system is down and starts to be restored as soon as a software or hardware failure occurs, and the system can not operate until the restoration action is complete.

A2. The restoration action for the software subsystem implies the debugging activity, which is performed perfectly with probability a $(0 < a \leq 1)$ and imperfectly with probability $b(= 1 - a)$. We call a the perfect debugging rate. One fault is corrected and removed from the software subsystem when the debugging activity is perfect.

A3. The next software failure-occurrence and restoration time for the software subsystem, when n faults have been corrected, follow exponential distributions with means $1/\lambda_n$ and $1/\mu_n$, respectively.

A4. The failure time-interval and the restoration time for the hardware subsystem follow exponential distributions with means $1/\theta$ and $1/\eta$, respectively.

A5. The probability that two or more software or hardware failures occur simultaneously is negligible.

Consider a stochastic process $\{X(t),\ t \geq 0\}$ representing the state of the system at time point t. The state space of the process $\{X(t),\ t \geq 0\}$ is defined as follows:

W_n: the system is operating,

R_n^{S}: the system is inoperable due to a software failure and restored,

R_n^{H}: the system is inoperable due to a hardware failure and restored,

where $n = 0,\ 1,\ 2,\ \ldots$ denotes the cumulative number of corrected faults. A sample behavior of the system is illustrated in Fig. 8.1.

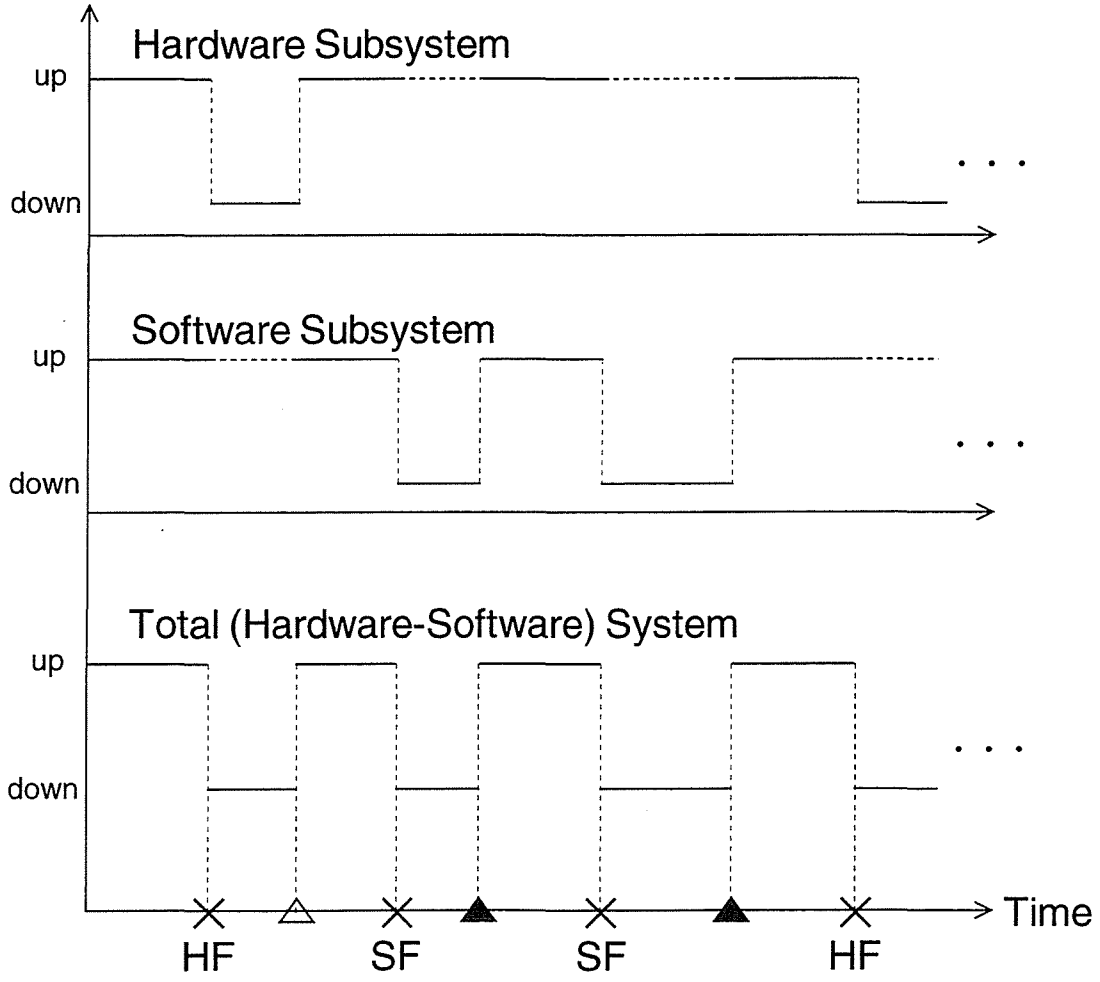

Fig. 8.1. A sample behavior of hardware-software system.

The descriptions of λ_n and μ_n are given by

$$\lambda_n = Dk^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{8.1}$$

$$\mu_n = Er^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ E > 0,\ 0 < r \leq 1), \tag{8.2}$$

respectively where D and k are the initial hazard rate and the decreasing ratio of the hazard rate for the software subsystem, respectively and E and r are the initial restoration rate and the decreasing ratio of the restoration rate for the software subsystem, respectively. The details of λ_n and μ_n have been discussed in Chapters 4 and 5.

Let $Q_{A,B}(\tau)$ $(A,\ B \in \{W_n,\ R_n^{\mathrm{S}},\ R_n^{\mathrm{H}};\ n = 0,\ 1,\ 2,\ \ldots\})$ denote the one-step transition probability that after making a transition into state A, the process $\{X(t),\ t \geq 0\}$ makes a transition into state B by time τ. The expressions for $Q_{A,B}(\tau)$'s are given as follows:

$$Q_{W_n,R_n^{\mathrm{S}}}(\tau) = \frac{\lambda_n}{\theta + \lambda_n}(1 - e^{-(\theta+\lambda_n)\tau}), \tag{8.3}$$

$$Q_{W_n,R_n^{\mathrm{H}}}(\tau) = \frac{\theta}{\theta + \lambda_n}(1 - e^{-(\theta+\lambda_n)\tau}), \tag{8.4}$$

$$Q_{R_n^{\mathrm{S}},W_{n+1}}(\tau) = a(1 - e^{-\mu_n\tau}), \tag{8.5}$$

$$Q_{R_n^{\mathrm{S}},W_n}(\tau) = b(1 - e^{-\mu_n\tau}), \tag{8.6}$$

$$Q_{R_n^{\mathrm{H}},W_n}(\tau) = 1 - e^{-\eta\tau}. \tag{8.7}$$

The sample state transition diagram of $X(t)$ is illustrated in Fig. 8.2.

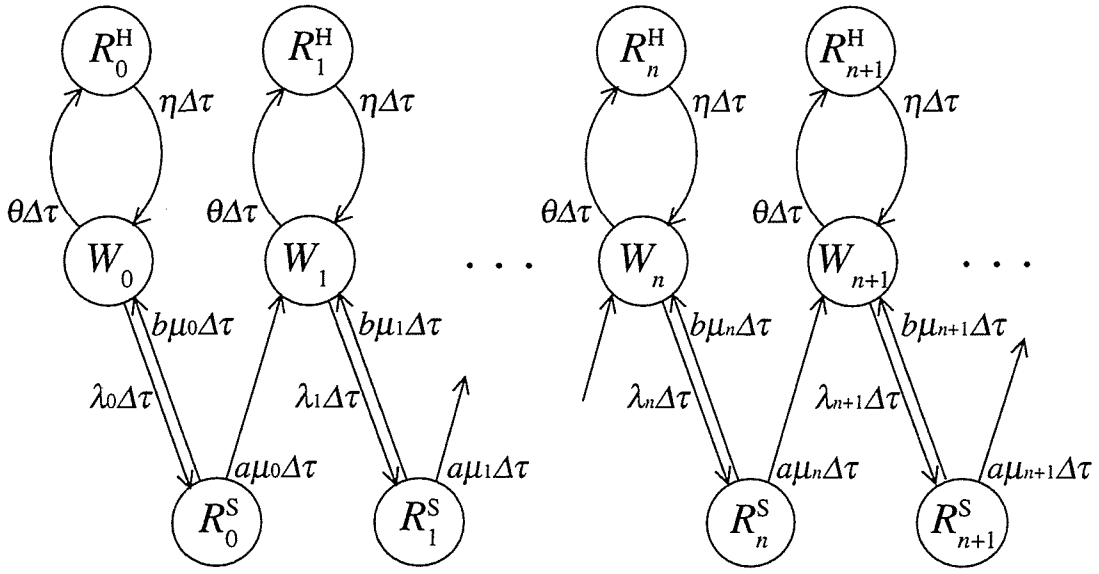

Fig. 8.2. A diagrammatic representation of state transitions between $X(t)$'s for availability modeling for hardware-software system.

8.3 Derivation of System Performance Measures

The availability analysis of this model is identical to the software availability modeling discussed in Chapter 5 by changing the definition of the state space of $X(t)$ from states R_n^{S} and R_n^{H} of this chapter to states R_n^1 and R_n^2 of Chapter 5, respectively.

8.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Let S_n $(n = 1, 2, \ldots; S_0 \equiv 0)$ be the random variable representing the time spent in correcting n faults. Then, we obtain the distribution function for S_n as

$$\begin{aligned} G_n(t) &\equiv \Pr\{S_n \le t\} \\ &= 1 - \sum_{i=0}^{n-1} (A_{n,i}^1 e^{-x_i t} + A_{n,i}^2 e^{-y_i t} + A_{n,i}^3 e^{-z_i t}) \\ &\quad (n = 1, 2, \ldots; G_0(t) \equiv \mathbf{1}(t)), \end{aligned} \tag{8.8}$$

where $-x_i$, $-y_i$, and $-z_i$ are the distinct roots of the following third order equation:

$$s^3 + (\theta + \eta + \lambda_i + \mu_i)s^2 + (\theta\mu_i + \eta\lambda_i + \eta\mu_i + a\lambda_i\mu_i)s + a\eta\lambda_i\mu_i = 0, \tag{8.9}$$

and constant coefficients $A_{n,i}^1$, $A_{n,i}^2$, and $A_{n,i}^3$ are given by

$$A_{n,i}^1 = \frac{[a(\eta - x_i)]^n \prod_{j=0}^{n-1} \lambda_j \mu_j}{x_i \prod_{\substack{j=0 \\ j \neq i}}^{n-1} (x_j - x_i) \prod_{j=0}^{n-1} (y_j - x_i)(z_j - x_i)} \quad (i = 0, 1, 2, \ldots, n-1), \tag{8.10}$$

$$A_{n,i}^2 = \frac{[a(\eta - y_i)]^n \prod_{j=0}^{n-1} \lambda_j \mu_j}{y_i \prod_{\substack{j=0 \\ j \neq i}}^{n-1} (y_j - y_i) \prod_{j=0}^{n-1} (z_j - y_i)(x_j - y_i)} \quad (i = 0, 1, 2, \ldots, n-1), \tag{8.11}$$

$$A_{n,i}^3 = \frac{[a(\eta - z_i)]^n \prod_{j=0}^{n-1} \lambda_j \mu_j}{z_i \prod_{\substack{j=0 \\ j \neq i}}^{n-1} (z_j - z_i) \prod_{j=0}^{n-1} (x_j - z_i)(y_j - z_i)} \quad (i = 0, 1, 2, \ldots, n-1), \tag{8.12}$$

respectively. It is noted that

$$\sum_{i=0}^{n-1}(A_{n,i}^1 + A_{n,i}^2 + A_{n,i}^3) = 1 \quad (n \geq 1). \tag{8.13}$$

Furthermore, $\mathrm{E}[S_n]$ and $\mathrm{Var}[S_n]$ are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1}\left(\frac{1}{x_i} + \frac{1}{y_i} + \frac{1}{z_i} - \frac{1}{\eta}\right), \tag{8.14}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1}\left(\frac{1}{{x_i}^2} + \frac{1}{{y_i}^2} + \frac{1}{{z_i}^2} - \frac{1}{\eta^2}\right), \tag{8.15}$$

respectively.

8.3.2 Operational State Occupancy Probability and System Availability

The operational state occupancy probability, which is defined as the probability that $X(t)$ is in state W_n at time point t, is obtained as

$$\begin{aligned} P_n(t) &\equiv \Pr\{X(t) = W_n\} \\ &= \frac{1}{a\lambda_n} g_{n+1}(t) + \frac{1}{a\lambda_n\mu_n} g'_{n+1}(t), \end{aligned} \tag{8.16}$$

where $g_n(t)$ is the probability density function of the random variable S_n and $g'_n(t) \equiv dg_n(t)/dt$.

The instantaneous system availability is defined as

$$A(t) \equiv \sum_{n=0}^{\infty} P_n(t), \tag{8.17}$$

which represents the probability that the system is operating at specified time point t. Furthermore, the average system availability in the time interval $(0,\ t]$ is defined as

$$A_{av}(t) \equiv \frac{1}{t}\int_0^t A(x)dx, \tag{8.18}$$

which represents the ratio of system's operating time to the time interval $(0,\ t]$. Using (8.16), we can describe (8.17) and (8.18) as

$$A(t) = \sum_{n=0}^{\infty}\left[\frac{g_{n+1}(t)}{a\lambda_n} + \frac{g'_{n+1}(t)}{a\lambda_n\mu_n}\right], \tag{8.19}$$

$$A_{av}(t) = \frac{1}{t}\sum_{n=0}^{\infty}\left[\frac{G_{n+1}(t)}{a\lambda_n} + \frac{g_{n+1}(t)}{a\lambda_n\mu_n}\right], \tag{8.20}$$

respectively.

8.4 Numerical Examples

Using the system availability model discussed above, we show numerical illustrations for system availability measurement and assessment.

The distribution functions of the first passage time to the specified number of corrected faults, $G_n(t)$'s, in (8.8) are shown in Fig. 8.3 for various perfect debugging rates, a, where $n = 5$, $\theta = 0.01$, $\eta = 1.0$, $D = 0.1$, $E = 0.5$, and $r = 0.9$. We can see that the smaller perfect debugging rate a becomes, the more difficult it is to remove faults from the system.

The instantaneous system availabilities, $A(t)$'s in (8.19) are shown in Fig. 8.4 for various a's, where $\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, and $r = 0.9$. System availability lowers just after the operation and (8.19) can evaluate the degree of unstableness of the system in the early stage of the operation phase. This figure shows that the higher certainty of the restoration action for the software subsystem becomes, the larger system availability becomes.

The dependence of the decreasing ratio of the restoration rate, r, on $A(t)$ is shown in Fig. 8.5, where $\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, and $a = 0.9$. As shown in Fig. 8.5, in case of $r > k$ and $r < k$, system availability improves and decreases with the lapse of time, respectively, and in case of $r = k$, this is constant. Then, the ratio of r to k can be regarded as an index to determine whether the system performance grows or not and reflects the degree of difficulty of the system dynamic-quality improvement.

The hazard rate for the next system failure when n faults have been already corrected is denoted as $\alpha_n = \theta + \lambda_n$. $A(t)$'s in case of $\theta : D = 5 : 1$ and $\theta : D = 1 : 5$ on the condition that $\alpha_0 = \theta + D$ is the same value are shown in Fig. 8.6, where $\eta = 1.0$, $k = 0.8$, $E = 0.5$, $r = 0.9$, and $a = 0.9$. Figure 8.6 shows as follows: In case of (i), system availability is stable when the software debugging is performed enough and the hazard rate for the software subsystem is low. In case of (ii), though system availability is low in the early stage of the operation phase, it increases with the lapse of time. The behavior shown in Fig. 8.6 is due to whether the system has much room for reliability growth or not. For example, $\alpha_0 = 0.06$ for both of (i) and (ii), but $\alpha_{10} = 0.0510$ for (i) and $\alpha_{10} = 0.0154$ for (ii). Then, (ii) has more room for reliability growth than (i).

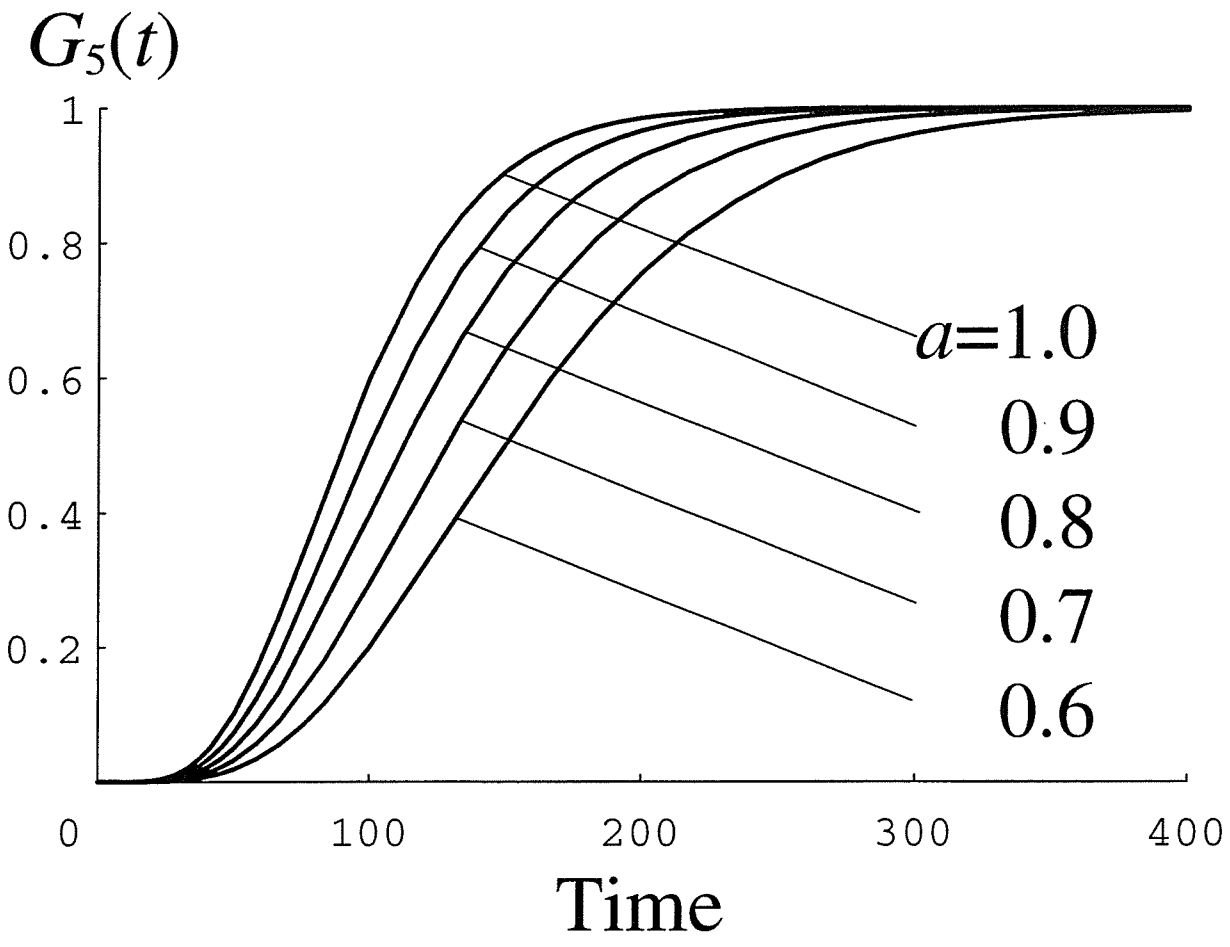

Fig. 8.3. Dependence of a on $G_n(t)$ ($n = 5$, $\theta = 0.01$, $\eta = 1.0$, $D = 0.1$, $k = 0.8$, $E = 0.5$, $r = 0.9$).

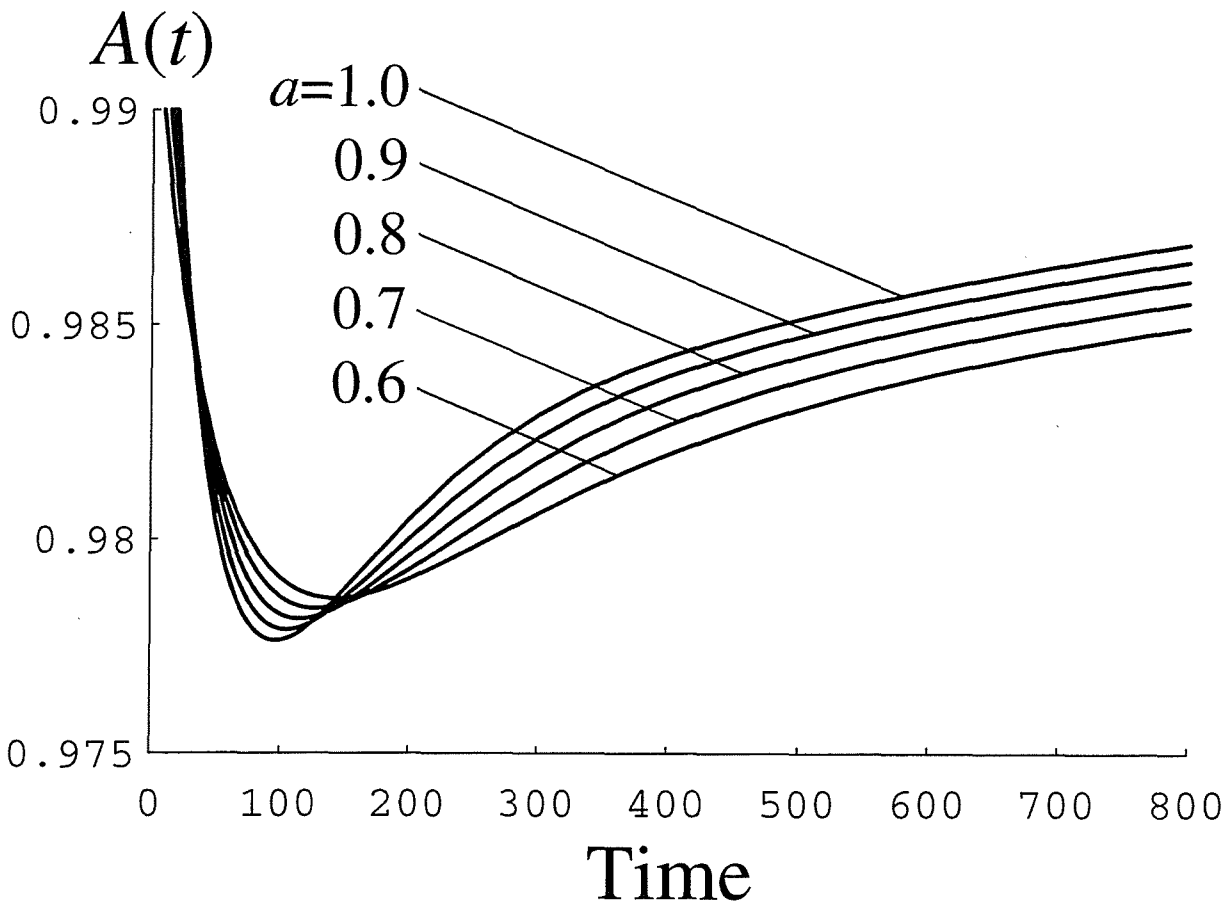

Fig. 8.4. Dependence of a on $A(t)$ ($\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, $r = 0.9$).

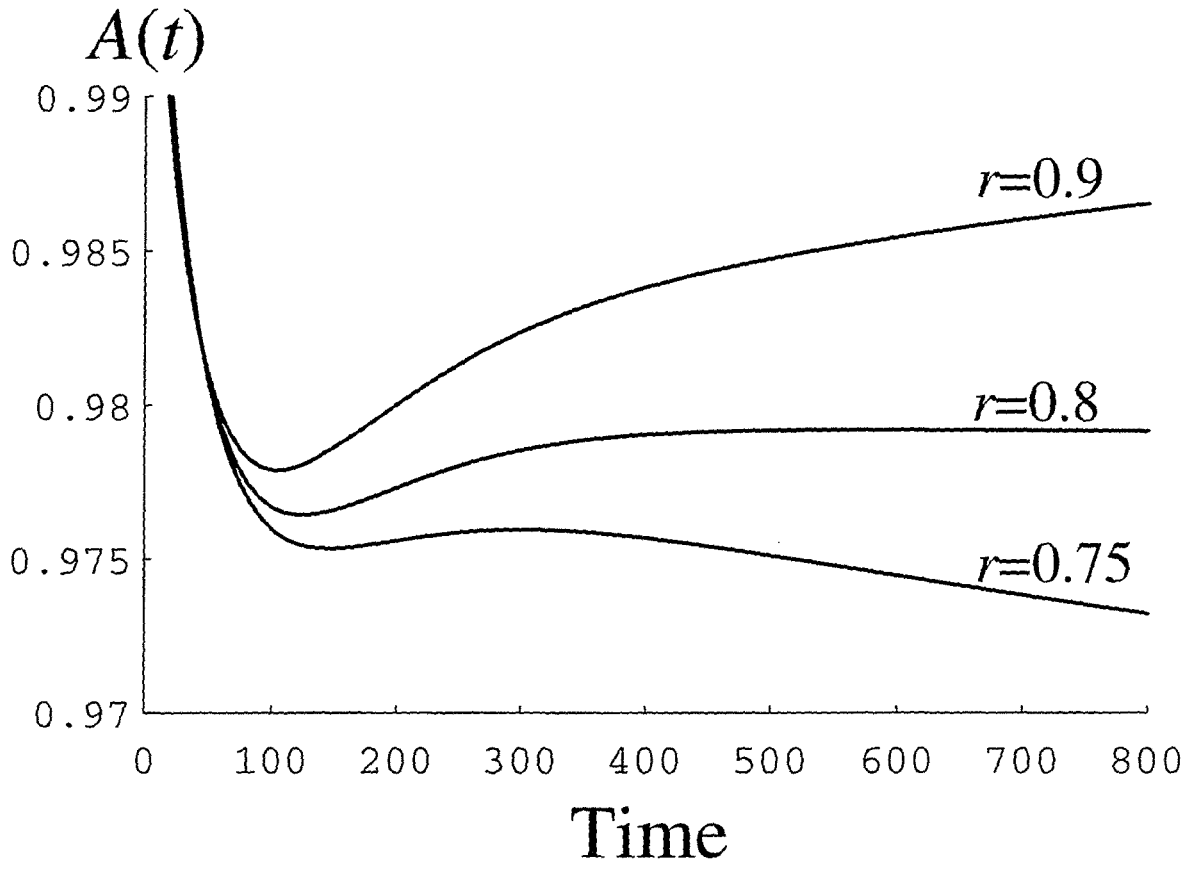

Fig. 8.5. Dependence of r on $A(t)$ ($\theta = 0.01$, $\eta = 1.0$, $D = 0.01$, $k = 0.8$, $E = 0.5$, $a = 0.9$).

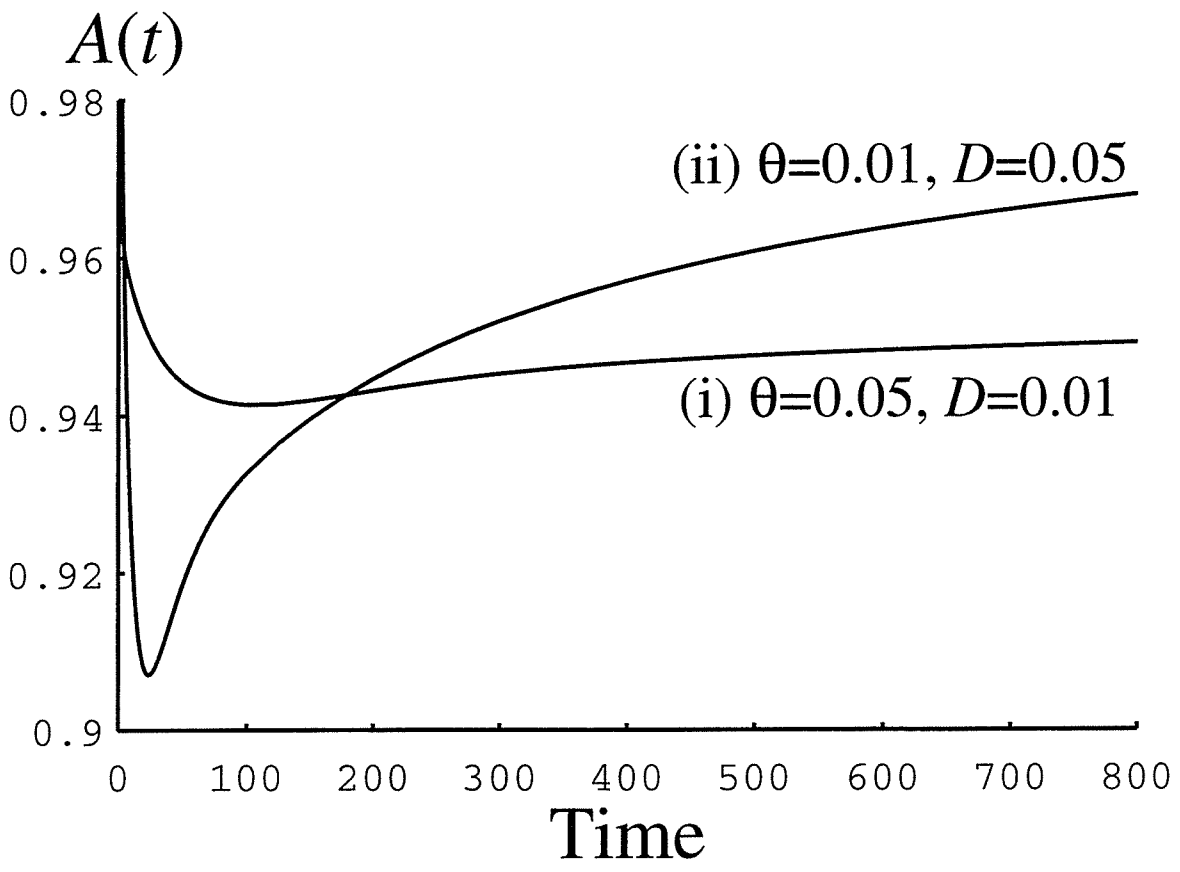

Fig. 8.6. Dependence of $\theta : D$ on $A(t)$ ($\eta = 1.0$, $k = 0.8$, $E = 0.5$, $r = 0.9$, $a = 0.9$).

8.5 Concluding Remarks

In this chapter, we have developed a system availability model for a computer system which has one hardware and one software subsystem. The failure-occurrence phenomenon for a software subsystem has been described by a geometrically decreasing hazard rate and that for a hardware subsystem by a constant hazard rate. Several useful quantitative measures for system performance assessment have been derived. Numerical illustrations for system availability measurement and assessment have been also presented to show that these measures are very useful for system performance assessment.

Part III

SOFTWARE SAFETY MODELING

Chapter 9

Software Safety Modeling Related to Failure Occurrences

9.1 Introduction

It has been very hard to forecast the degree of hazardous effects on our social lives in occurrence of accidents connected with computer systems since present-day life has been highly information-oriented and computer systems have grown in size and complexity. Recently, failures and accidents due to defects and errors latent in software systems have been increasing remarkably. Introducing such defects and errors is inescapable since software systems are developed by labor-intensive techniques and source programs and documents are intellectual products. Therefore, the concept of software engineering has been emphasized. Software engineering aims to manage the software life-cycle comprehensively, considering productivity, quality, cost, and delivery simultaneously. In particular, great importance has been attached to quality management techniques among the software production technology and methodology for the purpose of improving software reliability, that is, the characteristics that computer systems continue operating regularly without occurrence of failures on software systems [25, 35, 46, 57, 61].

Both software safety and reliability so far tended to be treated in the same concept. However, software safety differs from software reliability and is regarded as an important quality characteristic since software systems have controlled safety-critical systems. Software safety is defined as the nonoccurrence of unsafe states in software systems. Software systems in unsafe states lead to fatal accidents, mishaps, and hazards. Software reliability is the attribute that software systems do not engender software failures, whereas software safety is the attribute that software systems do not fall into hazardous conditions. A soft-

ware failure is defined as an unacceptable departure from program operation caused by a fault remaining in the software system. Accordingly, we can evaluate software safety in terms of the probability that a software system does not fall into a state that induces hazards whether or not the system is performing its intended functions [22, 58].

Fault Tree Analysis (FTA) and Failure Mode and Effect Analysis (FMEA) are representative qualitative safety-assessment techniques [15]. These are effective tools for investigation and verification of reliability and safety in the specification and the design phase. However, quantitative methods of software safety evaluation in dynamic environment during the testing phase and the user operation in the field are seldom discussed.

In this chapter, we discuss a quantitative safety assessment model for software systems by assuming that the software failures occurring in dynamic environment are classified into two types: one leads to unsafe states and the other does not. This model describes the relationship between the operating time and software safety/availability by taking the time to restore to an operable state after a software failure-occurrence (i.e. the debugging time) into consideration [50]. This model is formulated as a Markov process [47] and the metrics of software safety and availability are derived. Finally, numerical examples for software safety assessment using this model are provided.

9.2 Model Description

First, we describe the process in which a system is restored to an operable state after a software failure-occurrence. In the case where a software failure occurs and a system falls into an unsafe state, an action to avoid an unsafe state is first performed and then a restoration action is performed so that a system operates regularly. On the other hand, if a system does not fall into an unsafe state, only a restoration action is performed. The following assumptions are made for software safety/availability assessment modeling:

A1. A system does not fall into any unsafe states when the system is operating. If a software failure occurs, a system falls into an unsafe state with probability p_1 $(0 < p_1 < 1)$, and does not fall into an unsafe state with probability $p_2 (= 1 - p_1)$.

A2. The restoration action includes the debugging activity and a fault causing a software failure is corrected and removed from a system with a perfect debugging activity.

A debugging activity is performed perfectly with probability a $(0 < a \leq 1)$, while imperfectly with probability $b(= 1 - a)$. We call a the perfect debugging rate.

A3. When n faults have been corrected, the time to the next software failure-occurrence and the restoration time follow exponential distributions with means $1/\lambda_n$ and $1/\mu_n$, respectively.

A4. The time interval of an action to avoid an unsafe state follows an exponential distribution with mean $1/\gamma$.

A5. The probability that two or more software failures occur simultaneously is negligible.

We introduce a stochastic process $\{X(t),\ t \geq 0\}$ to represent the states of a software system in dynamic environment [47]. The state space of process $\{X(t),\ t \geq 0\}$ is defined as follows:

W_n: the system is operating regularly and safely,

U_n: the system is in an unsafe state,

R_n: the system is restored,

where $n = 0,\ 1,\ 2,\ \ldots$ denotes the cumulative number of corrected faults.

From assumption A1, if a software failure occurs in $\{X(t) = W_n\}$, then

$$X(t) = \begin{cases} U_n & \text{(with probability } p_1) \\ R_n & \text{(with probability } p_2). \end{cases} \tag{9.1}$$

Furthermore from assumption A2, if a restoration action is complete in $\{X(t) = R_n\}$, then

$$X(t) = \begin{cases} W_n & \text{(with probability } b) \\ W_{n+1} & \text{(with probability } a). \end{cases} \tag{9.2}$$

The descriptions of λ_n and μ_n are given by

$$\lambda_n = Dk^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{9.3}$$

$$\mu_n = Er^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ E > 0,\ 0 < r \leq 1), \tag{9.4}$$

respectively where D and k are the initial hazard rate and the decreasing ratio of the hazard rate, respectively and E and r are the initial restoration rate and the decreasing ratio of the restoration rate, respectively. The details of λ_n and μ_n have been discussed in Chapters 4 and 5.

Let $Q_{A,B}(\tau)$ $(A,\ B \in \{W_n,\ U_n,\ R_n;\ n = 0,\ 1,\ 2,\ \ldots\})$ denote the one-step transition probability that $X(t)$ makes a transition into state B during time interval $(0, \tau]$ after $X(t)$ was in state A at time zero. The expressions for $Q_{A,B}(\tau)$'s are given as follows:

$$Q_{W_n,U_n}(\tau) = p_1(1 - e^{-\lambda_n \tau}), \tag{9.5}$$

$$Q_{W_n,R_n}(\tau) = p_2(1 - e^{-\lambda_n \tau}), \tag{9.6}$$

$$Q_{U_n,R_n}(\tau) = 1 - e^{-\gamma \tau}, \tag{9.7}$$

$$Q_{R_n,W_{n+1}}(\tau) = a(1 - e^{-\mu_n \tau}), \tag{9.8}$$

$$Q_{R_n,W_n}(\tau) = b(1 - e^{-\mu_n \tau}). \tag{9.9}$$

Figure 9.1 illustrates the sample state transition diagram of $X(t)$ forming a Markov process.

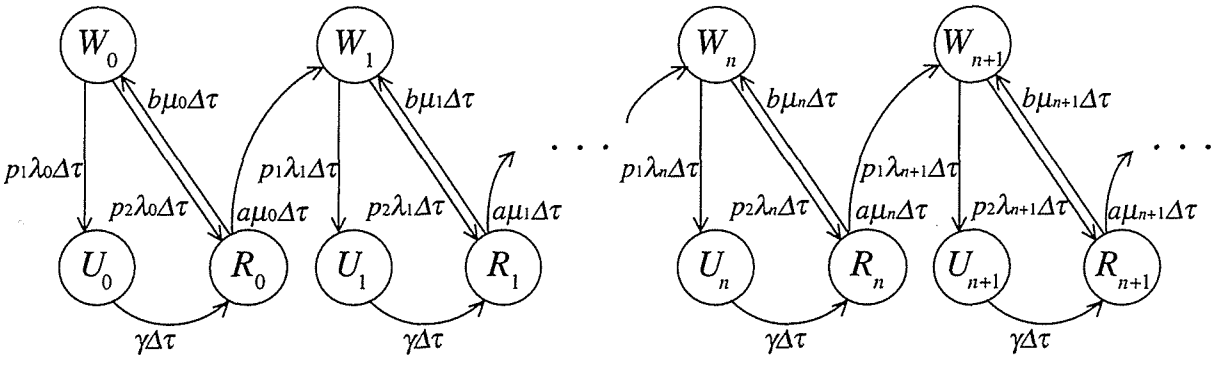

Fig. 9.1. A diagrammatic representation of state transitions between $X(t)$'s for software safety modeling related to failure occurrences.

9.3 Software Safety/Availability Analysis

9.3.1 Distribution of the First Passage Time to the Specified Number of Corrected Faults

Let $G_{i,n}(t)$ denote the probability that n faults are corrected during time interval $(0,\ t]$ on the condition that $i(< n)$ faults have already been corrected at time zero. Then, we obtain the following renewal equations from Fig. 9.1:

$$\left.\begin{aligned} G_{i,n}(t) &= H_{W_i,R_i} * Q_{R_i,W_{i+1}} * G_{i+1,n}(t) \\ &\quad + H_{W_i,R_i} * Q_{R_i,W_i} * G_{i,n}(t) \\ H_{W_i,R_i}(t) &= Q_{W_i,R_i}(t) + Q_{W_i,U_i} * Q_{U_i,R_i}(t) \\ &\quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1) \end{aligned}\right\}, \tag{9.10}$$

where $*$ denotes a Stieltjes convolution.

Substituting the Laplace-Stieltjes (L-S) transforms of (9.5)–(9.9) into that of (9.10) yields

$$\widetilde{G}_{i,n}(s) = \frac{a\lambda_i\mu_i(p_2 s + \gamma)}{(s+x_i)(s+y_i)(s+z_i)}\widetilde{G}_{i+1,n}(s) \quad (i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{9.11}$$

where $-x_i$, $-y_i$, and $-z_i$ are the distinct roots of the following third order equation:

$$s^3 + (\lambda_i + \mu_i + \gamma)s^2 + (\lambda_i\mu_i + \mu_i\gamma + \gamma\lambda_i - bp_2\lambda_i\mu_i)s + a\lambda_i\mu_i\gamma = 0. \tag{9.12}$$

Solving (9.11) recursively, we obtain the L-S transform of $G_{0,n}(t)$ as

$$\begin{aligned} \widetilde{G}_{0,n}(s) &= \prod_{i=0}^{n-1} \frac{a\lambda_i\mu_i(p_2 s + \gamma)}{(s+x_i)(s+y_i)(s+z_i)} \\ &= \sum_{i=0}^{n-1} \left(\frac{A^1_{n,i}x_i}{s+x_i} + \frac{A^2_{n,i}y_i}{s+y_i} + \frac{A^3_{n,i}z_i}{s+z_i} \right), \end{aligned} \tag{9.13}$$

where constant coefficients $A^1_{n,i}$, $A^2_{n,i}$, and $A^3_{n,i}$ are given by

$$\begin{aligned} A^1_{n,i} = &\frac{\prod_{j=0}^{n-1} a\lambda_j\mu_j(\gamma - p_2 x_i)}{x_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(x_j - x_i) \prod_{j=0}^{n-1}(y_j - x_i)(z_j - x_i)} \\ &(i = 0,\ 1,\ 2,\ \ldots,\ n-1), \end{aligned} \tag{9.14}$$

$$A^2_{n,i} = \frac{\prod_{j=0}^{n-1} a\lambda_j\mu_j(\gamma - p_2 y_i)}{y_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(y_j - y_i) \prod_{j=0}^{n-1}(z_j - y_i)(x_j - y_i)}$$
$$(i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{9.15}$$

$$A^3_{n,i} = \frac{\prod_{j=0}^{n-1} a\lambda_j\mu_j(\gamma - p_2 z_i)}{z_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1}(z_j - z_i) \prod_{j=0}^{n-1}(x_j - z_i)(y_j - z_i)}$$
$$(i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{9.16}$$

respectively and we postulate $\prod_{\substack{j=0 \\ j\neq 0}}^{0} \cdot = 1$. It is noted that

$$\sum_{i=0}^{n-1}(A^1_{n,i} + A^2_{n,i} + A^3_{n,i}) = 1 \quad (n = 1,\ 2,\ \ldots). \tag{9.17}$$

Inverting (9.13) and rewriting $G_{0,n}(t)$ as $G_n(t)$, we obtain the distribution function of the random variable S_n ($n = 1,\ 2,\ \ldots;\ S_0 \equiv 0$) representing the time to be spent in correcting n faults as

$$\begin{aligned} G_n(t) &\equiv \Pr\{S_n \leq t\} \\ &= 1 - \sum_{i=0}^{n-1}(A^1_{n,i}e^{-x_i t} + A^2_{n,i}e^{-y_i t} + A^3_{n,i}e^{-z_i t}) \\ &\quad (n = 1,\ 2,\ \ldots;\ G_0(t) \equiv \mathbf{1}(t)\ \text{(unit function)}), \end{aligned} \tag{9.18}$$

Furthermore, the expectation $\mathrm{E}[S_n]$ and the variance $\mathrm{Var}[S_n]$ are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1}\left(\frac{1}{x_i} + \frac{1}{y_i} + \frac{1}{z_i} - \frac{p_2}{\gamma}\right), \tag{9.19}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1}\left(\frac{1}{{x_i}^2} + \frac{1}{{y_i}^2} + \frac{1}{{z_i}^2} - \frac{{p_2}^2}{\gamma^2}\right), \tag{9.20}$$

respectively.

9.3.2 State Occupancy Probability

Here, we derive the probabilities that $X(t)$ is in the respective states.

Let $P_{A,B}(t)$ be the conditional probability that $X(t)$ is in state A at time point t on the condition that $X(t)$ was in state B at time point zero, i.e.

$$P_{A,B}(t) \equiv \Pr\{X(t) = B \mid X(0) = A\}$$
$$(A,\ B \in \{W_n,\ U_n,\ R_n;\ n = 0,\ 1,\ 2,\ \ldots\}), \tag{9.21}$$

and let $P_{W_n}(t) \equiv P_{W_0,W_n}(t)$, $P_{R_n}(t) \equiv P_{W_0,R_n}(t)$, and $P_{U_n}(t) \equiv P_{W_0,U_n}(t)$ denote the state occupancy probabilities that $X(t)$ is in states W_n, R_n, and U_n at time point t, respectively.

Initially, we obtain the following renewal equations with respect to $P_{W_n}(t)$:

$$P_{W_n}(t) = G_n * P_{W_n,W_n}(t), \tag{9.22}$$
$$P_{W_n,W_n}(t) = e^{-\lambda_n t} + H_{W_n,R_n} * Q_{R_n,W_n} * P_{W_n,W_n}(t). \tag{9.23}$$

From (9.23), the L-S transform of $P_{W_n,W_n}(t)$ is given by

$$\tilde{P}_{W_n,W_n}(s) = \frac{s(s+\mu_n)(s+\gamma)}{(s+x_n)(s+y_n)(s+z_n)}. \tag{9.24}$$

Substituting (9.24) into the L-S transform of (9.22) yields

$$\begin{aligned}\tilde{P}_{W_n}(s) &= \frac{s(s+\gamma)(s+\mu_n)\prod_{i=0}^{n-1} a\lambda_i\mu_i(p_2 s+\gamma)}{\prod_{i=0}^{n}(s+x_i)(s+y_i)(s+z_i)} \\ &= \sum_{i=0}^{n}\left(\frac{-B^1_{n,i}x_i}{s+x_i} + \frac{-B^2_{n,i}y_i}{s+y_i} + \frac{-B^3_{n,i}z_i}{s+z_i}\right),\end{aligned} \tag{9.25}$$

where constant coefficients $B^1_{n,i}$, $B^2_{n,i}$, and $B^3_{n,i}$ are given by

$$\begin{aligned}B^1_{n,i} &= \frac{(\gamma - x_i)(\mu_n - x_i)\prod_{j=0}^{n-1} a\lambda_j\mu_j(\gamma - p_2 x_i)}{\prod_{\substack{j=0\\ j\neq i}}^{n}(x_j - x_i)\prod_{j=0}^{n}(y_j - x_i)(z_j - x_i)} \\ &\quad (i = 0,\ 1,\ 2,\ \ldots,\ n),\end{aligned} \tag{9.26}$$
$$B^2_{n,i} = \frac{(\gamma - y_i)(\mu_n - y_i)\prod_{j=0}^{n-1} a\lambda_j\mu_j(\gamma - p_2 y_i)}{\prod_{\substack{j=0\\ j\neq i}}^{n}(y_j - y_i)\prod_{j=0}^{n}(z_j - y_i)(x_j - y_i)}$$

$$(i = 0,\ 1,\ 2,\ \ldots,\ n), \tag{9.27}$$

$$B^3_{n,i} = \frac{(\gamma - z_i)(\mu_n - z_i)\prod_{j=0}^{n-1} a\lambda_j\mu_j(\gamma - p_2 z_i)}{\prod_{\substack{j=0 \\ j\neq i}}^{n}(z_j - z_i)\prod_{j=0}^{n}(x_j - z_i)(y_j - z_i)}$$

$$(i = 0,\ 1,\ 2,\ \ldots,\ n), \tag{9.28}$$

respectively. Inverting (9.25), we have the state occupancy probability that $X(t)$ is in state W_n at time point t as

$$\begin{aligned} P_{W_n}(t) &\equiv \Pr\{X(t) = W_n\} \\ &= \sum_{i=0}^{n}\left[B^1_{n,i}e^{-x_i t} + B^2_{n,i}e^{-y_i t} + B^3_{n,i}e^{-z_i t}\right] \\ &\quad (n = 0,\ 1,\ 2,\ \ldots). \end{aligned} \tag{9.29}$$

It is noted that

$$\left.\begin{aligned} &B^1_{0,0} + B^2_{0,0} + B^3_{0,0} = 1 \\ &\sum_{i=0}^{n}\left(B^1_{n,i} + B^2_{n,i} + B^3_{n,i}\right) = 0 \quad (n = 1,\ 2,\ \ldots) \end{aligned}\right\}. \tag{9.30}$$

Following steps similar to those used in the derivation of $P_{W_n}(t)$, we obtain the following renewal equations with respect to $P_{R_n}(t)$:

$$P_{R_n}(t) = G_n * H_{W_n,R_n} * P_{R_n,R_n}(t), \tag{9.31}$$

$$P_{R_n,R_n}(t) = e^{-\mu_n t} + Q_{R_n,W_n} * H_{W_n,R_n} * P_{R_n,R_n}(t). \tag{9.32}$$

From (9.31) and (9.32), the L-S transform of $P_{R_n}(t)$ is given by

$$\begin{aligned} \widetilde{P}_{R_n}(s) &= \frac{s}{a\mu_n}\cdot\frac{a\lambda_n\mu_n(p_2 s + \gamma)}{(s + x_n)(s + y_n)(s + z_n)}\cdot\widetilde{G}_n(s) \\ &= \frac{s}{a\mu_n}\cdot\widetilde{G}_{n+1}(s). \end{aligned} \tag{9.33}$$

Inverting (9.33), we have the state occupancy probability that $X(t)$ is in state R_n at time point t as

$$\begin{aligned} P_{R_n}(t) &\equiv \Pr\{X(t) = R_n\} \\ &= \frac{1}{a\mu_n}\cdot g_{n+1}(t) \\ &\quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{9.34}$$

where $g_n(t)$ denotes the probability density function of S_n i.e. $g_n(t) \equiv dG_n(t)/dt$.

Let $\{Y(t),\ t \geq 0\}$ denote the stochastic process representing the cumulative number of faults corrected perfectly up to time t. Then, we obtain the following equivalent relation:

$$\{Y(t) = n\} \iff \{X(t) = W_n\} \cup \{X(t) = R_n\} \cup \{X(t) = U_n\}. \tag{9.35}$$

Furthermore, since $\{Y(t),\ t \geq 0\}$ is a counting process, we also obtain the following equivalent relation:

$$\{S_n \leq t\} \iff \{Y(t) \geq n\}. \tag{9.36}$$

From (9.18) and (9.36), the probability that n faults are corrected up to time t is given by

$$\begin{aligned} P_n(t) &\equiv \Pr\{Y(t) = n\} \\ &= G_n(t) - G_{n+1}(t) \quad (n = 0,\ 1,\ 2,\ \ldots). \end{aligned} \tag{9.37}$$

Therefore, we have the state occupancy probability that $X(t)$ is in state U_n at time point t as

$$\begin{aligned} P_{U_n}(t) &\equiv \Pr\{X(t) = U_n\} \\ &= G_n(t) - G_{n+1}(t) - P_{W_n}(t) - P_{R_n}(t) \quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{9.38}$$

since $\{X(t) = W_n\}$, $\{X(t) = R_n\}$, and $\{X(t) = U_n\}$ are mutually exclusive.

9.3.3 Software Safety

The following equation holds for arbitrary time point t:

$$\sum_{n=0}^{\infty} [P_{W_n}(t) + P_{R_n}(t) + P_{U_n}(t)] = 1. \tag{9.39}$$

In this chapter, the software safety is defined as the probability that a system does not fall into any unsafe states at time point t and given by

$$S(t) \equiv \sum_{n=0}^{\infty} [P_{W_n}(t) + P_{R_n}(t)]. \tag{9.40}$$

On the other hand, the software unsafety is defined as the probability that a system falls into unsafe states at time point t and given by

$$U(t) \equiv \sum_{n=0}^{\infty} P_{U_n}(t). \tag{9.41}$$

From (9.40), (9.41) can be expressed by

$$\begin{aligned} U(t) &= 1 - S(t) \\ &= 1 - \sum_{n=0}^{\infty} \left[P_{W_n}(t) + P_{R_n}(t) \right]. \end{aligned} \tag{9.42}$$

9.3.4 Instantaneous Software Availability

From this model, we can also derive a software availability assessment measure. That is, the instantaneous software availability defined as the probability that a system is operating regularly at time point t is given by

$$A(t) \equiv \sum_{n=0}^{\infty} P_{W_n}(t). \tag{9.43}$$

9.4 Numerical Examples

We offer several numerical examples for software assessment based on the stochastic model discussed above.

Figures 9.2 and 9.3 represent the software safety in (9.40) and the instantaneous software availability in (9.43) for various values of p_1 denoting the probability that a system falls into an unsafe state in a software failure-occurrence, respectively. These figures indicate that the software safety and availability drop rapidly immediately after operation. This means that system performance is unstable in the early stage of the operation phase. These figures also suggest that a system has higher availability and safety with decreasing p_1. Figures 9.4 and 9.5 show the software safety and the instantaneous software availability for various values of γ. As these figures indicate, shortening the action time to avoid an unsafe state raises system safety and availability.

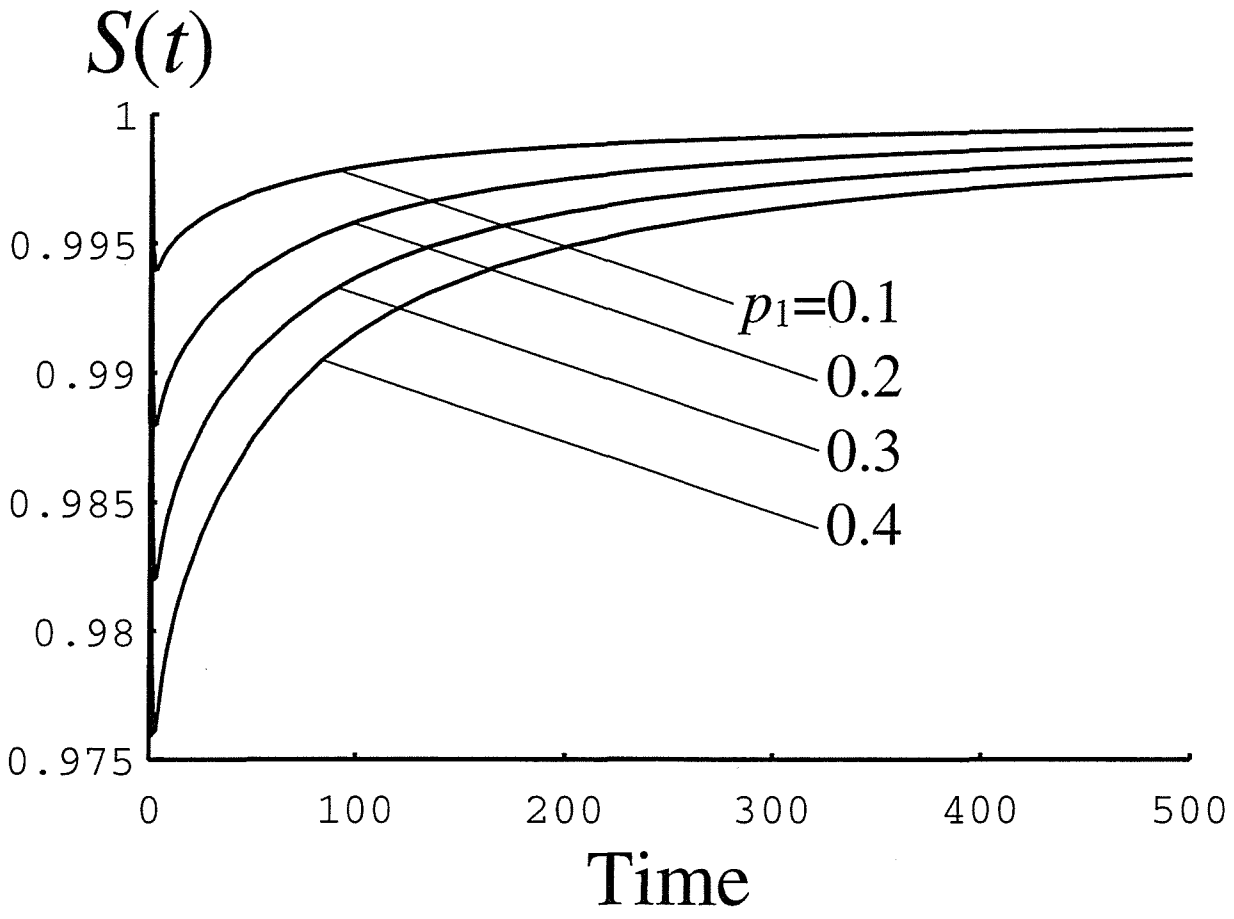

Fig. 9.2. Dependence of p_1 on software safety $S(t)$ ($D = 0.1$, $k = 0.8$, $E = 1.0$, $r = 0.9$, $a = 0.9$, $\gamma = 1.5$).

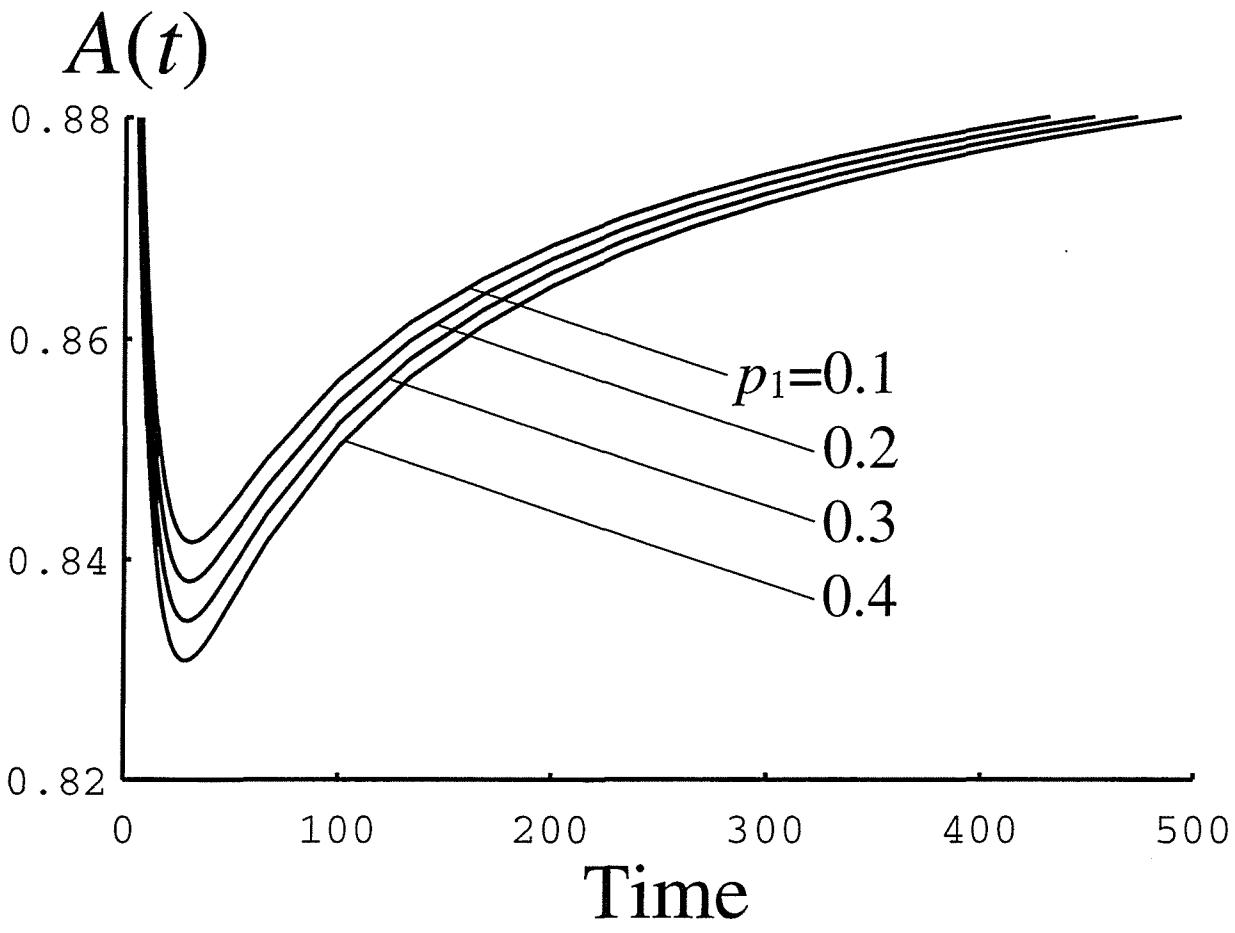

Fig. 9.3. Dependence of p_1 on instantaneous software availability $A(t)$ ($D = 0.1$, $k = 0.8$, $E = 1.0$, $r = 0.9$, $a = 0.9$, $\gamma = 1.5$).

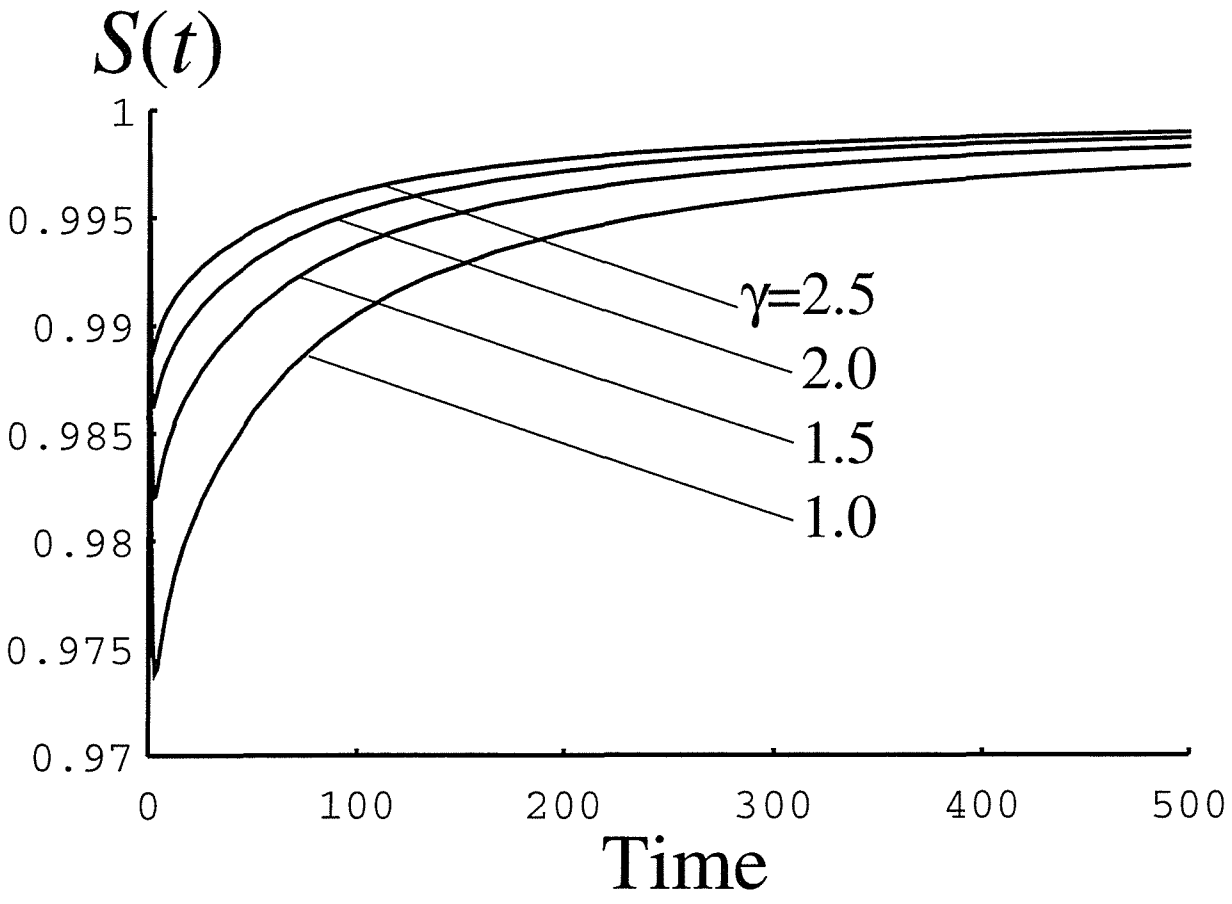

Fig. 9.4. Dependence of γ on software safety $S(t)$ ($D = 0.1$, $k = 0.8$, $E = 1.0$, $r = 0.9$, $p_1 = 0.3$, $a = 0.9$).

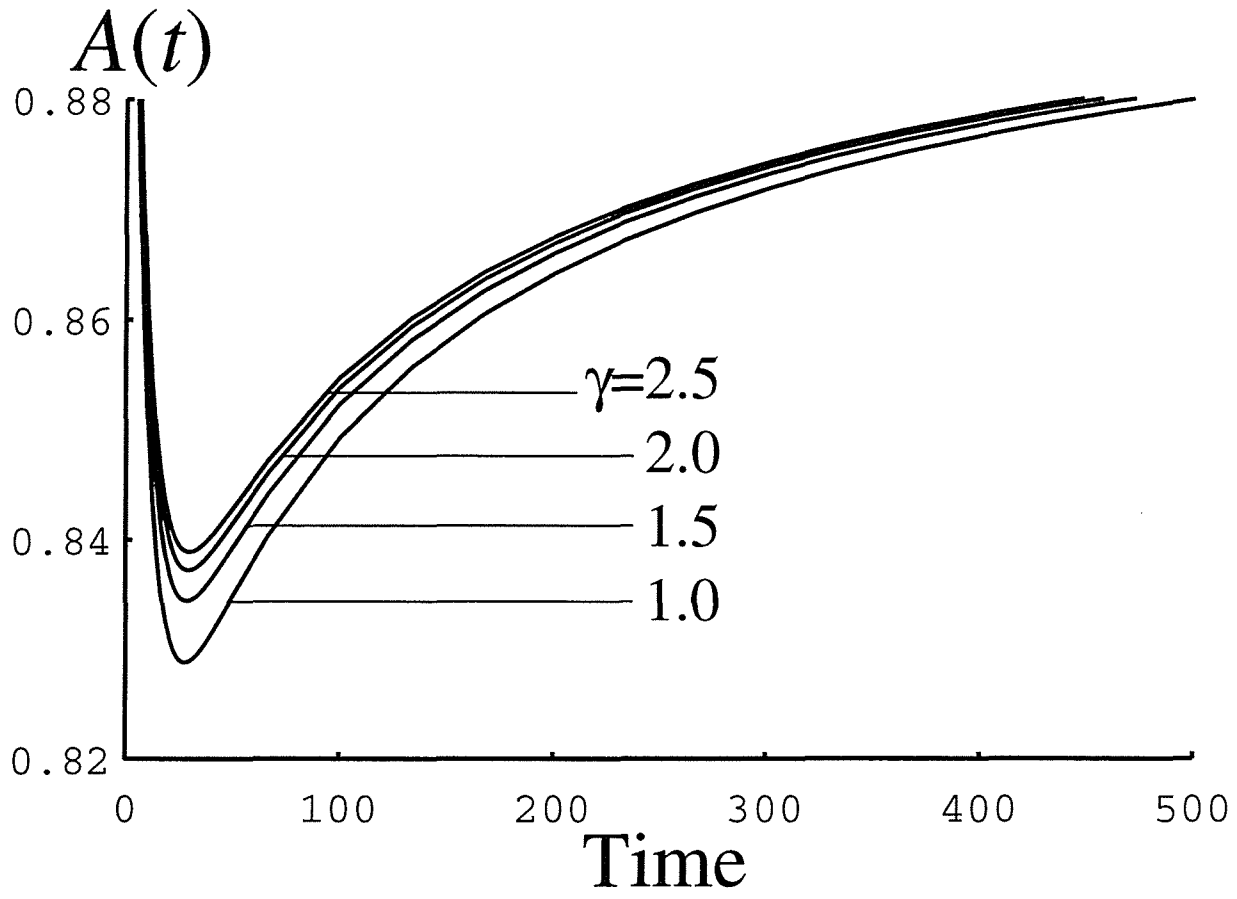

Fig. 9.5. Dependence of γ on instantaneous software availability $A(t)$ ($D = 0.1$, $k = 0.8$, $E = 1.0$, $r = 0.9$, $p_1 = 0.3$, $a = 0.9$).

9.5 Concluding Remarks

In this chapter, we have proposed a quantitative software safety assessment model, taking notice of whether or not software failure-occurrences lead to unsafe states. The stochastic behavior of the system in dynamic environment has been described by a Markov process. The software safety and availability metrics have been derived from this model and enabled us to evaluate quantitative software safety as well as qualitative analyses so far [58]. In particular, it is very meaningful that quantitative evaluation techniques for software avail-

ability measurement have been correlated with software safety, which is the attribute of nonoccurrence of catastrophic consequences. This model has hinted at the possibility of establishing total evaluation techniques for software performance assessment. Furthermore, numerical examples of software safety/availability measurement have been presented.

Chapter 10

Software Reliability/Availability Modeling with Safety

10.1 Introduction

Software reliability is one of the most important quality characteristics and its evaluation methods have been much discussed [43]. A mathematical model for software reliability measurement is called a software reliability growth model which describes a software fault-detection or a software failure-occurrence phenomenon in a dynamic environment such as the testing phase of software development and the actual operation phase. A software failure is defined as an unacceptable departure from program operation caused by a fault remaining in the software system.

The notion of software safety begins being distinct from that of software reliability and being regarded as an important quality characteristic since software systems have controlled safety-critical systems such as nuclear power generation systems, defense systems, vehicle control systems, and so on. Software reliability is the attribute that software systems do not incur software failures, while software safety is the attribute that software systems do not fall into hazardous conditions whether or not the systems are performing intended functions [22]. Accordingly, software failure-occurrences do not always cause the problems related to safety, and software systems functioning in accordance with the specifications do not always ensure safety. There exist Fault Tree Analysis (FTA) and Failure Mode and Effect Analysis (FMEA) in qualitative techniques for evaluating and assessing software safety [15]. But safety-critical systems have many restrictions concerning "time" and there is a limitation to perform dynamic analyses with FTA and FMEA. Any quantitative methods of software safety evaluation in dynamic environments are scarcely

discussed. In the preceding chapter, we have developed the quantitative safety assessment model describing the relationship between software failure-occurrences and safety.

In this chapter, we develop two stochastic software safety assessment models based on the models discussed in Chapters 2 and 4: the software reliability assessment model with safety and the availability-intensive safety assessment model. Our attention in this chapter is directed to the event that the system causes hazardous conditions randomly in operation, not that the system may fall into an unsafe state when the system is down due to software failure-occurrence. These two models are formulated by Markov processes [47] to describe the time-dependent behaviors of the software system, taking into account the software reliability growth process. These models can provide the metrics of software safety defined as the probability that the system does not fall into hazardous states at a specified time point. Numerical illustrations are presented to show that these models are useful for software safety/reliability measurement and assessment.

10.2 Software Reliability Assessment Model with Safety

10.2.1 Model Description

Referring to the model discussed in Chapter 2, we give the following assumptions to construct the software reliability assessment model with safety taking notice of only operational states:

A1. When the software system is operating, the holding times of the safe and the unsafe state follow exponential distributions with means $1/\theta$ and $1/\eta$, respectively.

A2. A debugging activity is performed when a software failure occurs. Debugging activities are perfect with probability a $(0 < a \leq 1)$, while imperfect with probability $b(= 1 - a)$. We call a the perfect debugging rate.

A3. Software reliability growth occurs in case of perfect debugging. The time interval between software failure-occurrences follows an exponential distribution with mean $1/\lambda_n$, where $n = 0,\ 1,\ 2,\ \ldots$ denotes the cumulative number of corrected faults.

A4. The probability that two or more software failures occur simultaneously is negligible.

A5. Only one fault is corrected and removed from the system per one activity of perfect debugging. Debugging activities perform in the safe condition and the debugging time is not considered.

Consider a stochastic process $\{X(t),\ t \geq 0\}$ representing the state of the software system at time point t. The state space of $\{X(t),\ t \geq 0\}$ is defined as follows [47]:

W_n: the system is operating safely,

U_n: the system falls into the unsafe state.

From assumption A2, when the next software failure occurs in $\{X(t) = W_n\}$ or $\{X(t) = U_n\}$,

$$X(t) = \begin{cases} W_n & \text{(with probability } b\text{)} \\ W_{n+1} & \text{(with probability } a\text{).} \end{cases} \tag{10.1}$$

The description of λ_n is given by

$$\lambda_n = Dk^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{10.2}$$

where D and k are the initial hazard rate and the decreasing ratio of the hazard rate, respectively (see Section 2.2).

In the preceding chapter, we have developed the model assuming the situation where software failure-occurrences may lead to unsafe states and the system in operation does not fall into any unsafe states. On the other hand, in this chapter we consider that the system may induce unsafe states in operation. Furthermore, we consider that the occurrences of the software failure and the unsafe conditions are independent mutually. In fact, some faults which cause software failures may be those in connection with safety, and debugging such faults may contribute to the improvement of software safety. However, debugging activities are to correct the program so that the system functions in accordance with the specifications and do not aim at the improvement of software safety. Therefore, we assume that parameters θ and η, which are related to software safety, are constant regardless of the software reliability growth process.

Let $Q_{A,B}(\tau)$ ($A,\ B \in \{W_n,\ U_n;\ n = 0,\ 1,\ 2,\ \ldots\}$) denote the one-step transition probability that after making a transition into state A, the process $\{X(t),\ t \geq 0\}$ makes a transition into state B by time τ. The expressions for $Q_{A,B}(\tau)$'s are given as follows:

$$Q_{W_n,U_n}(\tau) = \frac{\theta}{\lambda_n + \theta}\left[1 - e^{-(\lambda_n+\theta)\tau}\right], \tag{10.3}$$

$$Q_{W_n,W_{n+1}}(\tau) = \frac{a\lambda_n}{\lambda_n + \theta}\left[1 - e^{-(\lambda_n+\theta)\tau}\right], \tag{10.4}$$

$$Q_{W_n,W_n}(\tau) = \frac{b\lambda_n}{\lambda_n + \theta}\left[1 - e^{-(\lambda_n+\theta)\tau}\right], \tag{10.5}$$

$$Q_{U_n,W_{n+1}}(\tau) = \frac{a\lambda_n}{\lambda_n + \eta}\left[1 - e^{-(\lambda_n+\eta)\tau}\right], \tag{10.6}$$

$$Q_{U_n,W_n}(\tau) = \frac{b\lambda_n}{\lambda_n + \eta}\left[1 - e^{-(\lambda_n+\eta)\tau}\right], \tag{10.7}$$

$$Q^*_{U_n,W_n}(\tau) = \frac{\eta}{\lambda_n + \eta}\left[1 - e^{-(\lambda_n+\eta)\tau}\right], \tag{10.8}$$

where $Q_{U_n,W_n}(\tau)$ denotes the case where a software failure occurs and debugging is imperfect, whereas $Q^*_{U_n,W_n}(\tau)$ denotes the case where the system returns to the safe state before a software failure occurs. The sample state transition diagram of $X(t)$ is illustrated in Fig. 10.1.

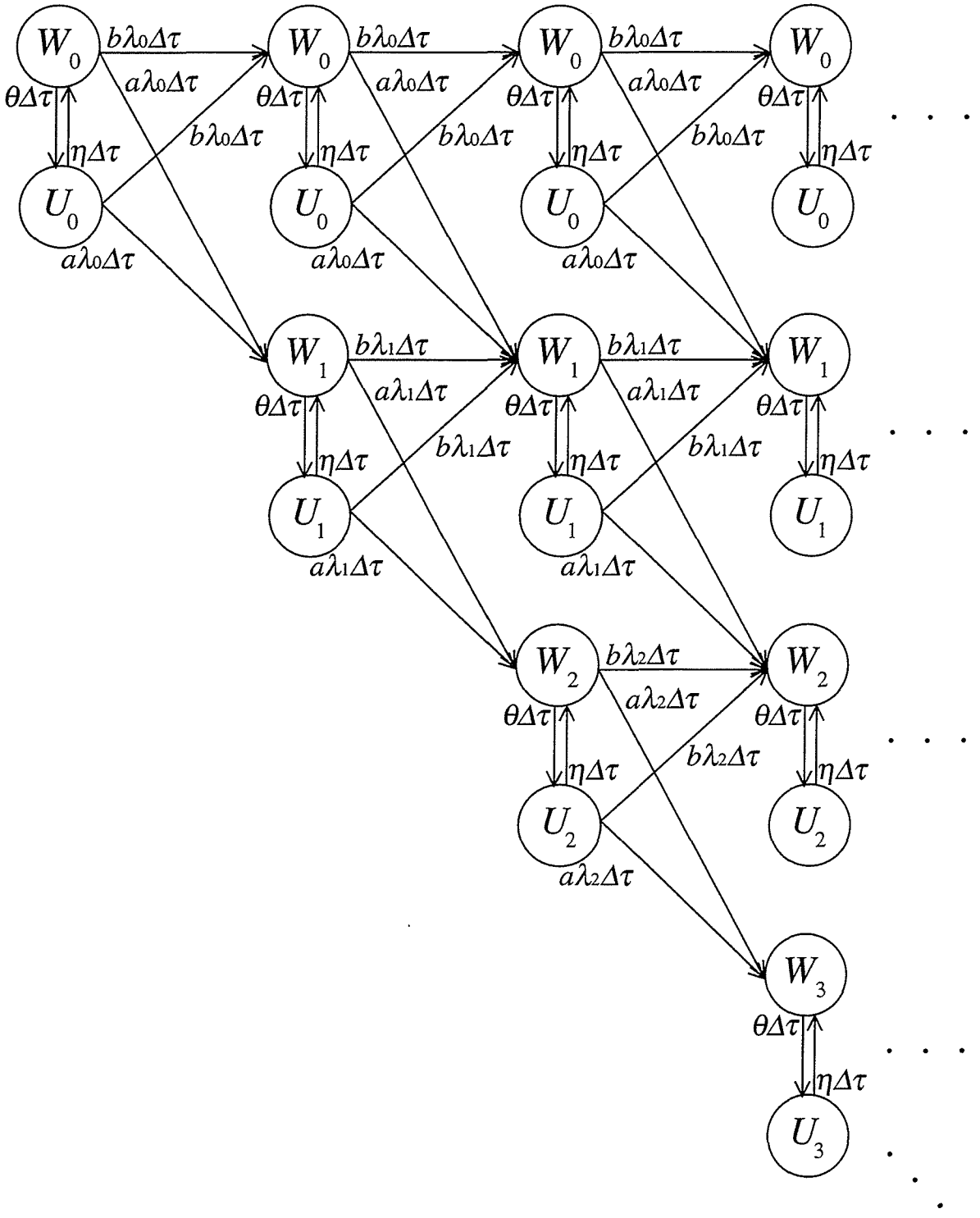

Fig. 10.1. A diagrammatic representation of state transitions between $X(t)$'s for the software reliability assessment model with safety.

10.2.2 Derivation of Safety/Reliability Measures

Distribution of the First Passage Time to a Specified Number of Corrected Faults

Let S_n ($n = 1,\ 2,\ \ldots;\ S_0 \equiv 0$) be the random variable representing the first passage time to state W_n, in other words the time spent in correcting n faults, and $G_n(t)$ be the distribution function of S_n. Furthermore, let $G_{i,n}(t)$ be the distribution function associated with the probability that n faults are corrected in the time interval $(0,\ t]$ on the condition that i faults have already been corrected at time zero. Then, we obtain the following renewal equation:

$$\begin{aligned} G_{i,n}(t) = {} & Q_{W_i,W_i} * G_{i,n}(t) + Q_{W_i,U_i} * Q^*_{U_i,W_i} * G_{i,n}(t) \\ & + Q_{W_i,U_i} * Q_{U_i,W_i} * G_{i,n}(t) + Q_{W_i,W_{i+1}} * G_{i+1,n}(t) \\ & + Q_{W_i,U_i} * Q_{U_i,W_{i+1}} * G_{i+1,n}(t) \\ & (i = 0,\ 1,\ 2,\ \ldots,\ n-1). \end{aligned} \tag{10.9}$$

Applying the Laplace-Stieltjes (L-S) transforms [42] to (10.9) recursively, we obtain the L-S transform of $G_n(t)$ as

$$\begin{aligned} \tilde{G}_n(s) &= \prod_{i=0}^{n-1} \frac{a\lambda_i}{s + a\lambda_i} \\ &= \sum_{i=0}^{n-1} A_i^n \frac{a\lambda_i}{s + a\lambda_i}, \end{aligned} \tag{10.10}$$

where

$$\left.\begin{aligned} & A_0^1 \equiv 1 \\ & A_i^n = \prod_{\substack{j=0 \\ j \neq i}}^{n-1} \frac{\lambda_j}{\lambda_j - \lambda_i} \\ & (n = 2,\ 3,\ \ldots;\ i = 0,\ 1,\ 2,\ \ldots,\ n-1) \end{aligned}\right\}. \tag{10.11}$$

By inverting (10.10), we have the distribution function of the first passage time when n faults are corrected:

$$\begin{aligned} G_n(t) &\equiv \Pr\{S_n \leq t\} \\ &= \sum_{i=0}^{n-1} A_i^n \left[1 - e^{-a\lambda_i t}\right] \\ & (t \geq 0;\ n = 1,\ 2,\ \ldots;\ G_0(t) \equiv \mathbf{1}(t)\ \text{(unit fuction)}), \end{aligned} \tag{10.12}$$

Equation (10.12) is identical to the result obtained in Chapter 2 and has no bearing on parameters θ and η, which are related to safety.

Furthermore, the mean and the variance of S_n are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1} \frac{1}{a\lambda_i}, \tag{10.13}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1} \frac{1}{(a\lambda_i)^2}, \tag{10.14}$$

respectively.

State Occupancy Probability and Software Safety Metrics

Let $P_{A,B}(t)$ be the conditional state occupancy probability that the system is in state B at time point t on the condition that the system was in state A at time point zero, i.e.

$$P_{A,B}(t) \equiv \Pr\{X(t) = B | X(0) = A\}$$
$$(A,\ B \in \{W_n,\ U_n;\ n = 0,\ 1,\ 2,\ \ldots\}), \tag{10.15}$$

and it is denoted that $P_{W_n}(t) \equiv P_{W_0,W_n}(t)$ and $P_{U_n}(t) \equiv P_{W_0,U_n}(t)$. Then, we obtain the following renewal equations:

$$P_{W_n}(t) = G_n * P_{W_n,W_n}(t), \tag{10.16}$$

$$\begin{aligned} P_{W_n,W_n}(t) &= e^{-(\lambda_n+\theta)t} + Q_{W_n,W_n} * P_{W_n,W_n}(t) \\ &\quad + Q_{W_n,U_n} * Q_{U_n,W_n} * P_{W_n,W_n}(t) \\ &\quad + Q_{W_n,U_n} * Q^*_{U_n,W_n} * P_{W_n,W_n}(t). \end{aligned} \tag{10.17}$$

By applying the L-S transforms to (10.16) and (10.17), we obtain the L-S transform of $P_{W_n}(t)$ as

$$\widetilde{P}_{W_n}(s) = \frac{s(s+\lambda_n+\eta)}{(s+\lambda_n+\theta+\eta)(s+a\lambda_n)} \prod_{i=0}^{n-1} \frac{a\lambda_i}{s+a\lambda_i}. \tag{10.18}$$

By inverting (10.18), we have $P_{W_n}(t)$ as

$$\begin{aligned} P_{W_n}(t) &\equiv \Pr\{X(t) = W_n\} \\ &= B^n e^{-(\lambda_n+\theta+\eta)t} + \sum_{i=0}^{n} B_i^n e^{-a\lambda_i t} \\ &\quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{10.19}$$

where constant coefficients B^n and B_i^n are given by

$$B^n = \frac{-\theta \prod_{j=0}^{n-1} a\lambda_j}{\prod_{j=0}^{n}(a\lambda_j - \lambda_n - \theta - \eta)} \quad (n = 0,\ 1,\ 2,\ \ldots), \tag{10.20}$$

$$B_i^n = \frac{(\lambda_n + \eta - a\lambda_i) \prod_{j=0}^{n-1} \lambda_j}{(\lambda_n + \theta + \eta - a\lambda_i) \prod_{\substack{j=0 \\ j\neq i}}^{n}(\lambda_j - \lambda_i)} \quad (i = 0,\ 1,\ 2,\ \ldots,\ n), \tag{10.21}$$

respectively, and we postulate $\prod_{j=0}^{-1} \cdot = \prod_{\substack{j=0 \\ j\neq 0}}^{0} \cdot = 1$. It is noted that

$$\left.\begin{aligned} &B^0 + B_0^0 = 1 \\ &B^n + \sum_{i=0}^{n} B_i^n = 0 \quad (n = 1,\ 2,\ \ldots) \end{aligned}\right\}. \tag{10.22}$$

The following equation holds for arbitrary time point t:

$$\sum_{n=0}^{\infty} [P_{W_n}(t) + P_{U_n}(t)] = 1. \tag{10.23}$$

In this section, the software safety is defined as

$$S_1(t) \equiv \sum_{n=0}^{\infty} P_{W_n}(t), \tag{10.24}$$

which represents the probability that the system does not fall into any unsafe states at time point t. The software unsafety is defined as

$$U_1(t) \equiv \sum_{n=0}^{\infty} P_{U_n}(t), \tag{10.25}$$

which represents the probability that the system falls into unsafe states at time point t. Using (10.23), we get

$$\begin{aligned} U_1(t) &= 1 - S_1(t) \\ &= 1 - \sum_{n=0}^{\infty} P_{W_n}(t). \end{aligned} \tag{10.26}$$

Software Reliability and MTBSF

Let X_l ($l = 1, 2, \ldots$) be the random variable representing the time interval between the $(l-1)$-st and the l-th software failure-occurrences. Then, the software reliability, i.e. $\Pr\{X_l > x\}$ and the mean time between software failures (MTBSF) are given by the same results obtained in Section 2.3.4, i.e.

$$\begin{aligned} R_l(x) &\equiv \Pr\{X_l > x\} \\ &= \sum_{i=0}^{l-1} \binom{l-1}{i} a^i b^{l-1-i} e^{-\lambda_i x}, \end{aligned} \tag{10.27}$$

$$\mathrm{E}[X_l] = \frac{(a/k+b)^{l-1}}{D}, \tag{10.28}$$

respectively where $\binom{l-1}{i} \equiv (l-1)!/[(l-1-i)!i!]$ denotes a binomial coefficient. $R_l(x)$ and $\mathrm{E}[X_l]$ also have no bearing on parameters θ and η.

10.3 Availability-Intensive Safety Assessment Model

In this section, we describe the behavior of a software system which alternates between operable and inoperable states in consideration of software safety, referring to Chapter 4.

10.3.1 Model Description

The following assumptions are made for availability-intensive safety assessment modeling:

B1. When the software system is operating, the holding times of the safe and the unsafe state follow exponential distributions with means $1/\theta$ and $1/\eta$, respectively.

B2. The software system breaks down and starts to be restored as soon as a software failure occurs, and the system can not operate until the restoration action is complete.

B3. The restoration action implies the debugging activity and software reliability growth occurs if a debugging activity is perfect

B4. The debugging activity is perfect with probability a $(0 < a \leq 1)$, while imperfect with probability $b(= 1 - a)$. A perfect debugging activity corrects and removes one fault from the system.

B5. When n faults have been corrected, the next software failure-occurrence time interval and the restoration time follow exponential distributions with means $1/\lambda_n$ and $1/\mu_n$, respectively.

B6. The probability that two or more software failures occur simultaneously is negligible.

B7. The restoration actions are performed in safe states.

The state space of the process $\{X(t),\ t \geq 0\}$ representing the state of the software system at time point t is defined over again as follows:

W_n: the system is operating in a safe state,

U_n: the system is operating in an unsafe state,

R_n: the system is inoperable and restored,

where $n = 0,\ 1,\ 2,\ \ldots$ denotes the cumulative number of faults corrected during the operation phase.

From assumption B4, when a restoration action is complete in $\{X(t) = R_n\}$,

$$X(t) = \begin{cases} W_n & \text{(with probability } b\text{)} \\ W_{n+1} & \text{(with probability } a\text{).} \end{cases} \tag{10.29}$$

The descriptions of λ_n and μ_n are given by

$$\lambda_n = Dk^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ D > 0,\ 0 < k < 1), \tag{10.30}$$

$$\mu_n = Er^n \quad (n = 0,\ 1,\ 2,\ \ldots;\ E > 0,\ 0 < r \leq 1), \tag{10.31}$$

respectively where D and k are the initial hazard rate and the decreasing ratio of the hazard rate, respectively and E and r are the initial restoration rate and the decreasing ratio of the restoration rate, respectively (see Section 4.2).

The expressions of one-step transition probabilities $Q_{A,B}(\tau)$'s ($A,\ B \in \{W_n, U_n, R_n;\ n = 0,\ 1,\ 2,\ \ldots\}$) are given as follows:

$$Q_{W_n,U_n}(\tau) = \frac{\theta}{\lambda_n + \theta}[1 - e^{-(\lambda_n+\theta)\tau}], \tag{10.32}$$

$$Q_{W_n,R_n}(\tau) = \frac{\lambda_n}{\lambda_n + \theta}[1 - e^{-(\lambda_n+\theta)\tau}], \tag{10.33}$$

$$Q_{U_n,W_n}(\tau) = \frac{\eta}{\lambda_n + \eta}[1 - e^{-(\lambda_n+\eta)\tau}], \tag{10.34}$$

$$Q_{U_n,R_n}(\tau) = \frac{\lambda_n}{\lambda_n + \eta}[1 - e^{-(\lambda_n+\eta)\tau}], \tag{10.35}$$

$$Q_{R_n,W_{n+1}}(\tau) = a(1 - e^{-\mu_n\tau}), \tag{10.36}$$

$$Q_{R_n,W_n}(\tau) = b(1 - e^{-\mu_n\tau}). \tag{10.37}$$

The sample state transition diagram of $X(t)$ is illustrated in Fig. 10.2.

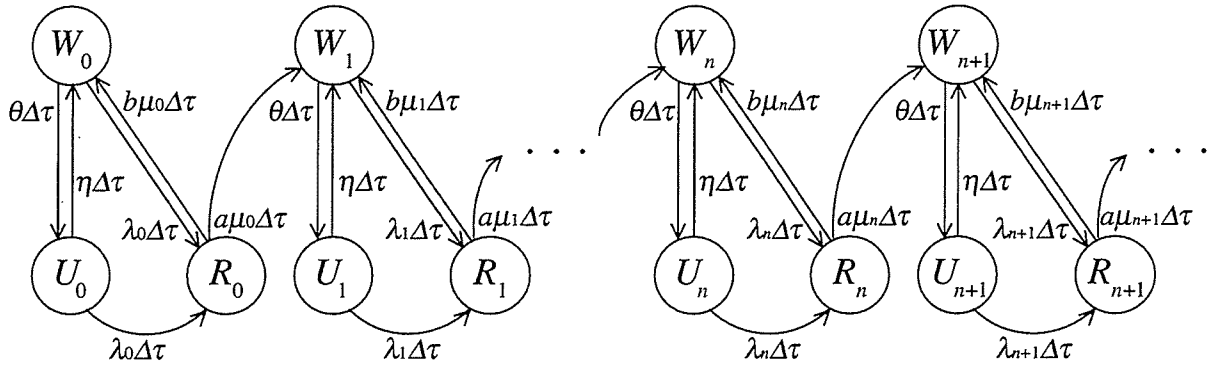

Fig. 10.2. A diagrammatic representation of state transitions between $X(t)$'s for the availability-intensive safety assessment model.

10.3.2 Software Availability/Safety Analysis

This model can be made analyses similar to those of the model discussed in Chapter 7 by changing the definition of the state space of $X(t)$ from state U_n of this section to state L_n of Chapter 7.

Distribution of the First Passage Time to the Specified Number of Corrected Faults

Recall that S_n and $G_n(t)$ ($n = 1,\ 2,\ \ldots;\ S_0 \equiv 0$) denote the random variable representing the time spent in correcting n faults and the distribution function of S_n, respectively. Then,

we have the distribution function of S_n as

$$\begin{aligned} G_n(t) &\equiv \Pr\{S_n \le t\} \\ &= 1 - \sum_{i=0}^{n-1} (A_{n,i}^1 e^{-x_i t} + A_{n,i}^2 e^{-y_i t}) \\ &\quad (n = 1,\ 2,\ \ldots;\ G_0(t) \equiv \mathbf{1}(t)), \end{aligned} \tag{10.38}$$

where

$$\left.\begin{matrix} x_i \\ y_i \end{matrix}\right\} = \frac{1}{2}\left[(\lambda_i + \mu_i) \pm \sqrt{(\lambda_i + \mu_i)^2 - 4a\lambda_i\mu_i}\right]$$
(double signs in same order). (10.39)

and constant coefficients $A_{n,i}^1$ and $A_{n,i}^2$ are given by

$$A_{n,i}^1 = \frac{\prod_{j=0}^{n-1} x_j y_j}{x_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1} (x_j - x_i) \prod_{j=0}^{n-1} (y_j - x_i)}$$
$$(i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{10.40}$$

$$A_{n,i}^2 = \frac{\prod_{j=0}^{n-1} x_j y_j}{y_i \prod_{\substack{j=0 \\ j\neq i}}^{n-1} (y_j - y_i) \prod_{j=0}^{n-1} (x_j - y_i)}$$
$$(i = 0,\ 1,\ 2,\ \ldots,\ n-1), \tag{10.41}$$

respectively. It is noted that (10.38) has no bearing on parameters θ and η and that

$$\sum_{i=0}^{n-1} (A_{n,i}^1 + A_{n,i}^2) = 1 \quad (n = 1,\ 2,\ \ldots). \tag{10.42}$$

Furthermore, the mean and the variance of S_n are given by

$$\mathrm{E}[S_n] = \sum_{i=0}^{n-1} \left(\frac{1}{x_i} + \frac{1}{y_i}\right), \tag{10.43}$$

$$\mathrm{Var}[S_n] = \sum_{i=0}^{n-1} \left(\frac{1}{{x_i}^2} + \frac{1}{{y_i}^2}\right), \tag{10.44}$$

respectively.

State Occupancy Probability

The state occupancy probabilities that $X(t)$ is in the respective states are derived analytically (see Section 7.3.2).

Initially, the probability that $X(t)$ is in state W_n at time point t is given by

$$\begin{aligned} P_{W_n}(t) &\equiv \Pr\{X(t) = W_n\} \\ &= B_n^0 e^{-(\lambda_n+\theta+\eta)t} + \sum_{i=0}^{n}(B_{n,i}^1 e^{-x_i t} + B_{n,i}^2 e^{-y_i t}) \\ &\quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{10.45}$$

where constant coefficients B_n^0, $B_{n,i}^1$, and $B_{n,i}^2$ are given by

$$\begin{aligned} B_n^0 &= \frac{-\theta(\mu_n - \lambda_n - \theta - \eta)\prod_{j=0}^{n-1} x_j y_j}{\prod_{j=0}^{n}(x_j - \lambda_n - \theta - \eta)(y_j - \lambda_n - \theta - \eta)} \\ &\quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{10.46}$$

$$\begin{aligned} B_{n,i}^1 &= \frac{(\lambda_n + \eta - x_i)(\mu_n - x_i)\prod_{j=0}^{n-1} x_j y_j}{(\lambda_n + \theta + \eta - x_i)\prod_{\substack{j=0 \\ j\neq i}}^{n}(x_j - x_i)\prod_{j=0}^{n}(y_j - x_i)} \\ &\quad (i = 0,\ 1,\ 2,\ \ldots,\ n), \end{aligned} \tag{10.47}$$

$$\begin{aligned} B_{n,i}^2 &= \frac{(\lambda_n + \eta - y_i)(\mu_n - y_i)\prod_{j=0}^{n-1} x_j y_j}{(\lambda_n + \theta + \eta - y_i)\prod_{\substack{j=0 \\ j\neq i}}^{n}(y_j - y_i)\prod_{j=0}^{n}(x_j - y_i)} \\ &\quad (i = 0,\ 1,\ 2,\ \ldots,\ n), \end{aligned} \tag{10.48}$$

respectively. It is noted that

$$\left.\begin{aligned} &B_0^0 + B_{0,0}^1 + B_{0,0}^2 = 1 \\ &B_n^0 + \sum_{i=0}^{n}\left(B_{n,i}^1 + B_{n,i}^2\right) = 0 \quad (n = 1,\ 2,\ \ldots) \end{aligned}\right\}. \tag{10.49}$$

Next, the probability that $X(t)$ is in state R_n at time point t is given by

$$\begin{aligned} P_{R_n}(t) &\equiv \Pr\{X(t) = R_n\} \\ &= \frac{1}{a\mu_n} g_{n+1}(t) \quad (n = 0,\ 1,\ 2,\ \ldots), \end{aligned} \tag{10.50}$$

where $g_n(t)$ denotes the probability density function of S_n, i.e. $g_n(t) \equiv dG_n(t)/dt$.

Finally, the probability that $X(t)$ is in state U_n at time point t is given by

$$\begin{aligned} P_{U_n}(t) &\equiv \Pr\{X(t) = U_n\} \\ &= G_n(t) - G_{n+1}(t) - P_{W_n}(t) - P_{R_n}(t) \\ &\quad (n = 0,\ 1,\ 2,\ \ldots). \end{aligned} \tag{10.51}$$

Software Safety and Availability

The following identical equation holds for arbitrary time t:

$$\sum_{n=0}^{\infty} [P_{W_n}(t) + P_{U_n}(t) + P_{R_n}(t)] = 1. \tag{10.52}$$

In this section, the software safety is defined as

$$S_2(t) \equiv \sum_{n=0}^{\infty} [P_{W_n}(t) + P_{R_n}(t)], \tag{10.53}$$

which represents the probability that the software system does not fall into any unsafe states at time point t.

The instantaneous software availability is defined as

$$A(t) \equiv \sum_{n=0}^{\infty} P_{W_n}(t), \tag{10.54}$$

which represents the probability that the software system is operating safely at time point t. Furthermore, the average software availability is defined as

$$A_{av}(t) \equiv \frac{1}{t} \int_0^t A(x)dx, \tag{10.55}$$

which represents the ratio of the amount of time when the system is operating safely to the time interval $(0,\ t]$. Equations (10.54) and (10.55) are the measures considering both safety and availability.

10.4 Numerical Examples

Using two software safety assessment models discussed above, we show numerical illustrations for software safety and reliability measurement.

The software safety metrics, $S_1(t)$ in (10.24) for various values of θ are shown in Fig. 10.3, where $D = 0.1$, $k = 0.8$, $a = 0.9$, and $\eta = 0.1$. Figure 10.3 indicates that the software safety becomes larger as θ decreases.

$S_1(t)$'s are shown in Fig. 10.4 for various values of k, where $D = 0.1$, $a = 0.9$, $\theta = 0.01$, and $\eta = 0.1$. $S_1(t)$ converges to $\eta/(\theta + \eta)$, which denotes the steady probability that the system is operating safely in the case where software failure-occurrences are not considered. Figure 10.4 indicates that the software safety converges earlier with decreasing k. Smaller k means that software reliability growth occurs more rapidly. Since this model assumes that the system is not unsafe in causing a software failure, the software safety becomes larger with increasing k, which means the high frequency of software failure-occurrences.

Figure 10.5 represents an example of the time-dependent behaviors of state occupancy probabilities, $P_{W_n}(t)$, $P_{R_n}(t)$, and $P_{U_n}(t)$ where $a = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.01$, and $\eta = 0.1$.

Figure 10.6 shows the dependence of a on the software safety, $S_2(t)$ in (10.53) where $D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.01$, and $\eta = 0.1$. This figure indicates that the software safety decreases with increasing a. This reasoning is the same as in the case of Fig. 10.4 since larger a means that software reliability growth occurs more rapidly.

Figures 10.7 and 10.8 represent the instantaneous software availability, $A(t)$ in (10.54) and the average software availability, $A_{av}(t)$ in (10.55) for various values of a, respectively, where $D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.01$, and $\eta = 0.1$. $A(t)$ and $A_{av}(t)$ drop rapidly immediately after operation and improve gradually with the lapse of time. These figures also tell us that a system has higher availability with increasing a.

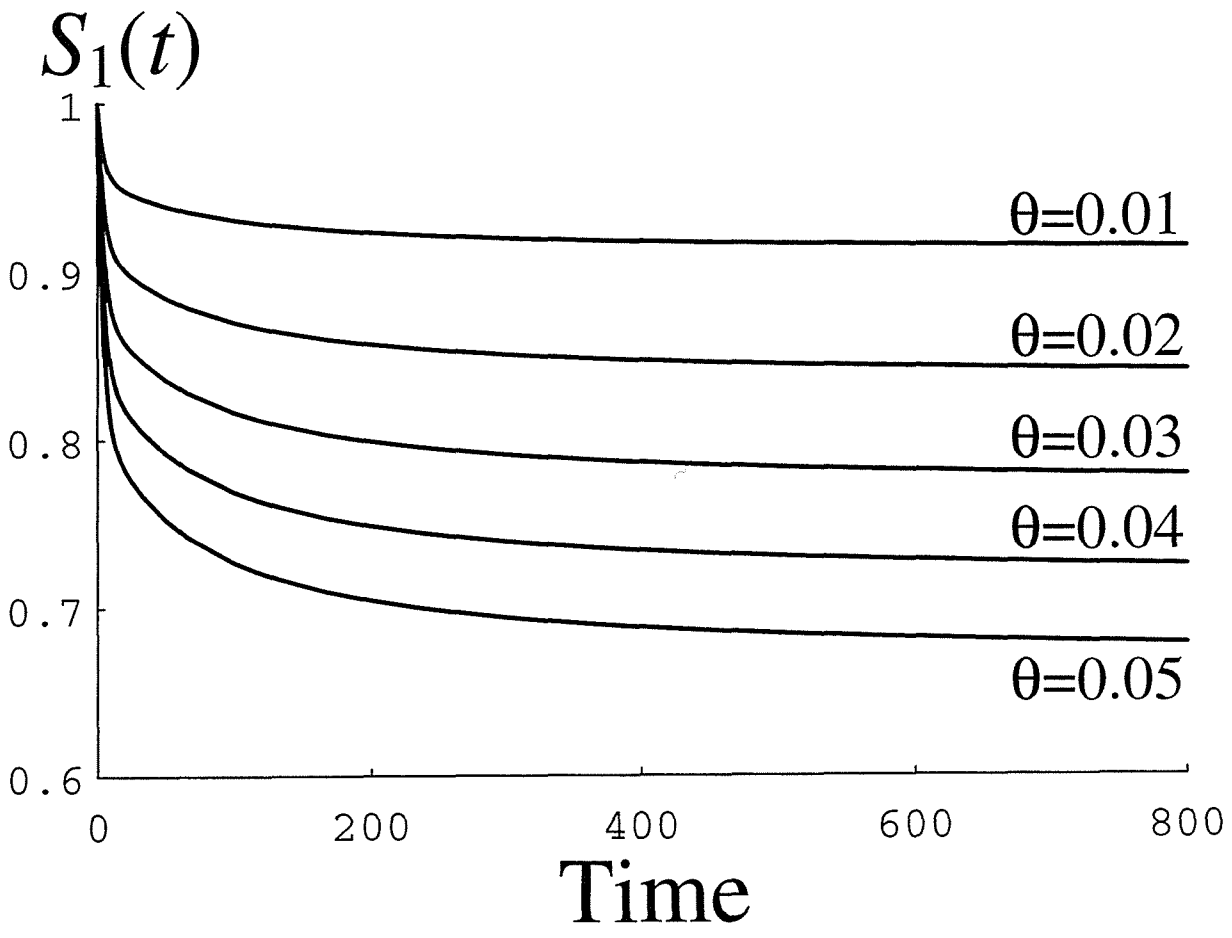

Fig. 10.3. Dependence of θ on $S_1(t)$ ($D = 0.1$, $k = 0.8$, $a = 0.9$, $\eta = 0.1$).

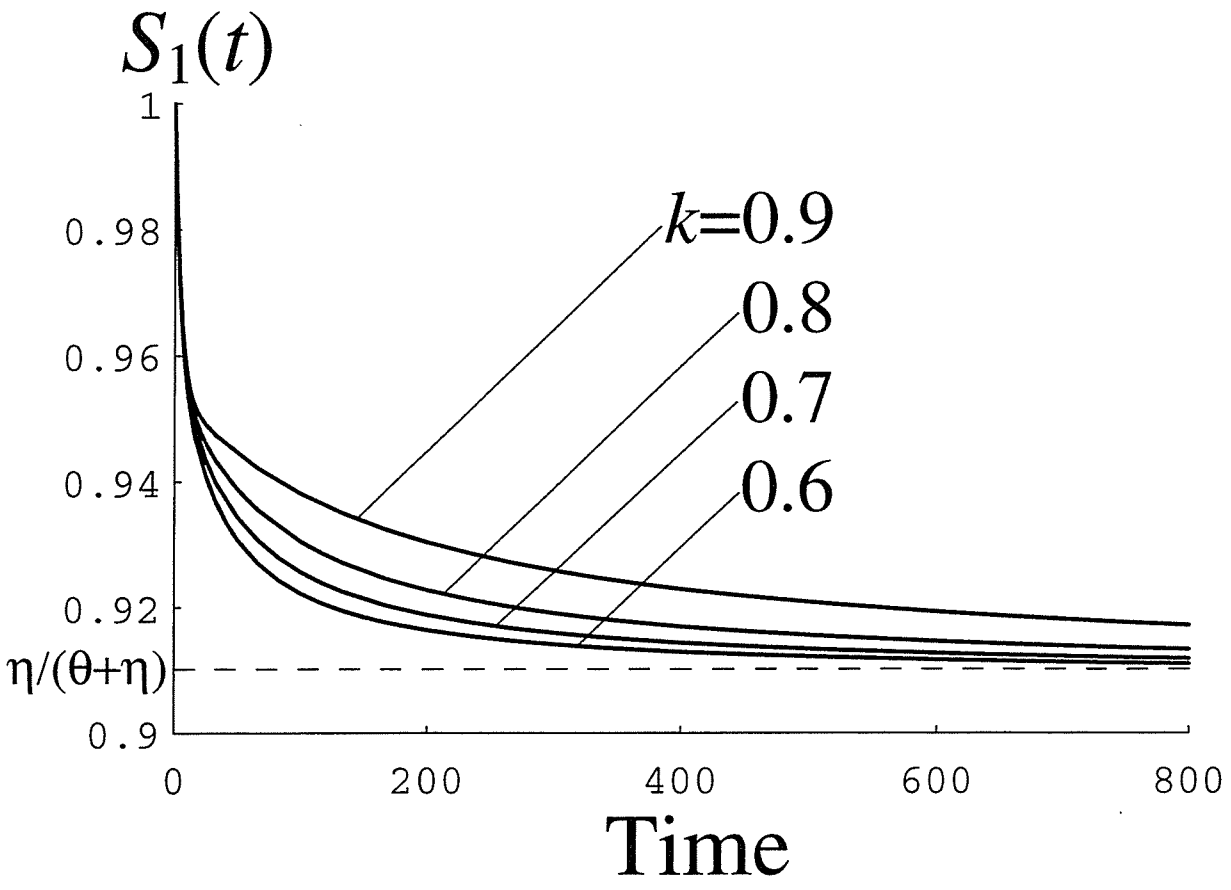

Fig. 10.4. Dependence of k on $S_1(t)$ ($D = 0.1$, $a = 0.9$, $\theta = 0.01$, $\eta = 0.1$).

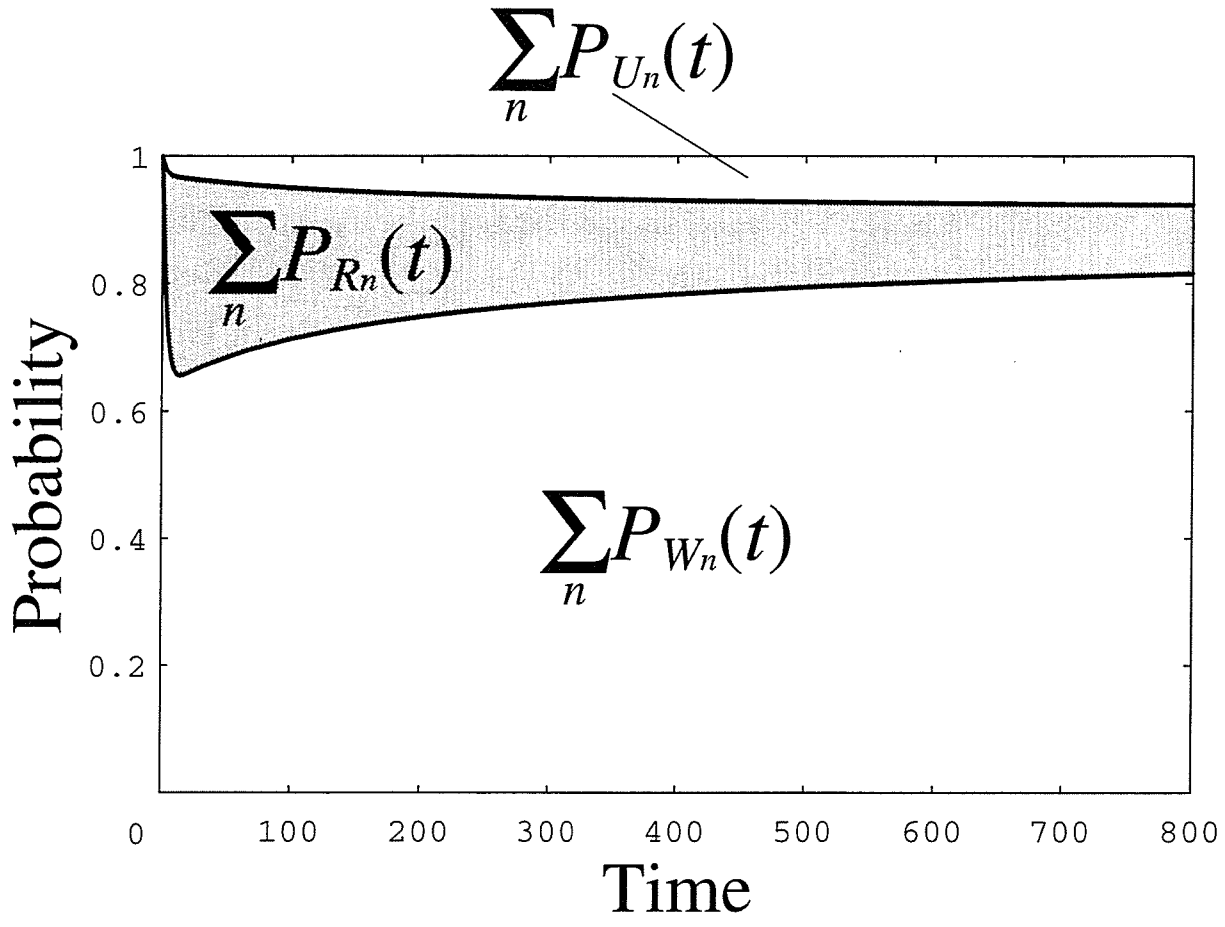

Fig. 10.5. Behaviors of state occupancy probabilities ($a = 0.9$, $D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.01$, $\eta = 0.1$).

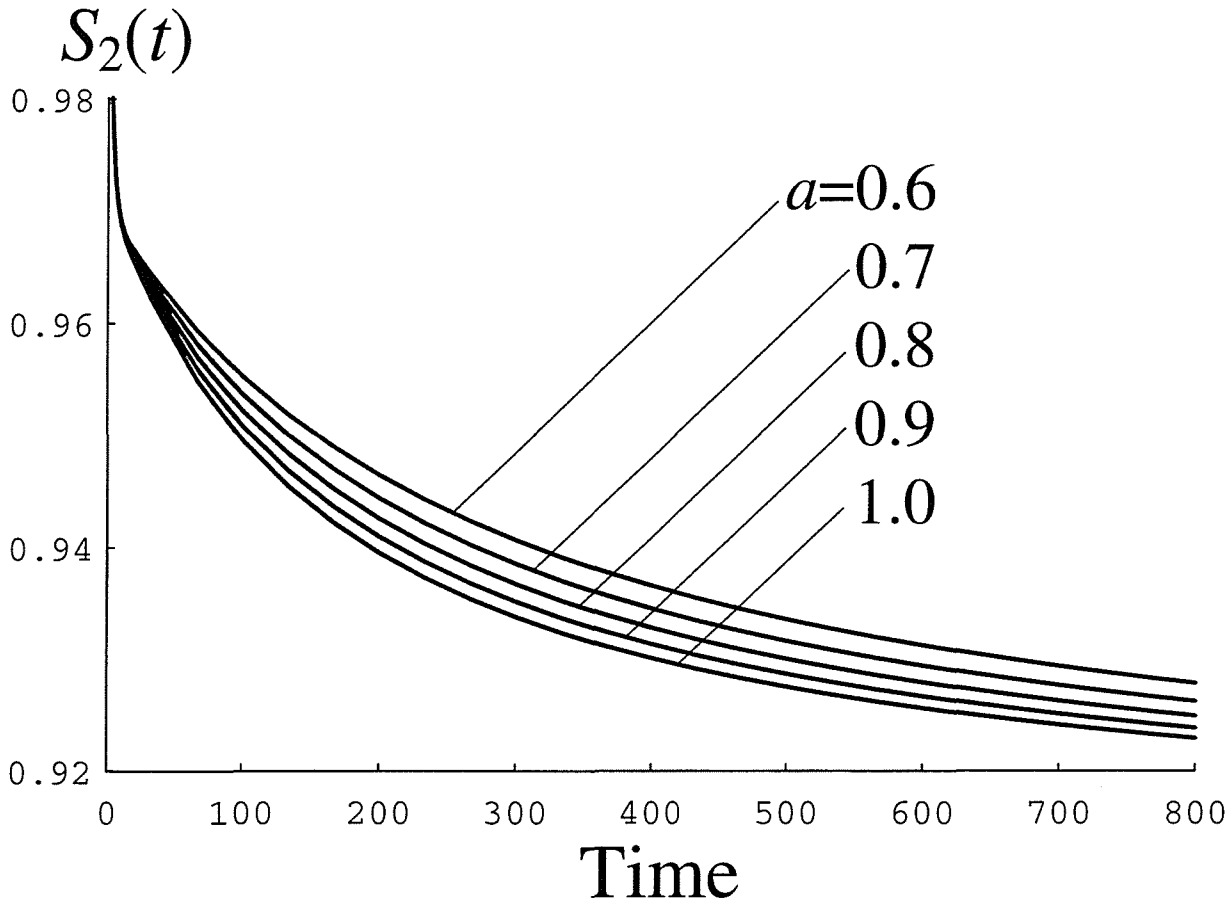

Fig. 10.6. Dependence of a on $S_2(t)$ ($D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.01$, $\eta = 0.1$).

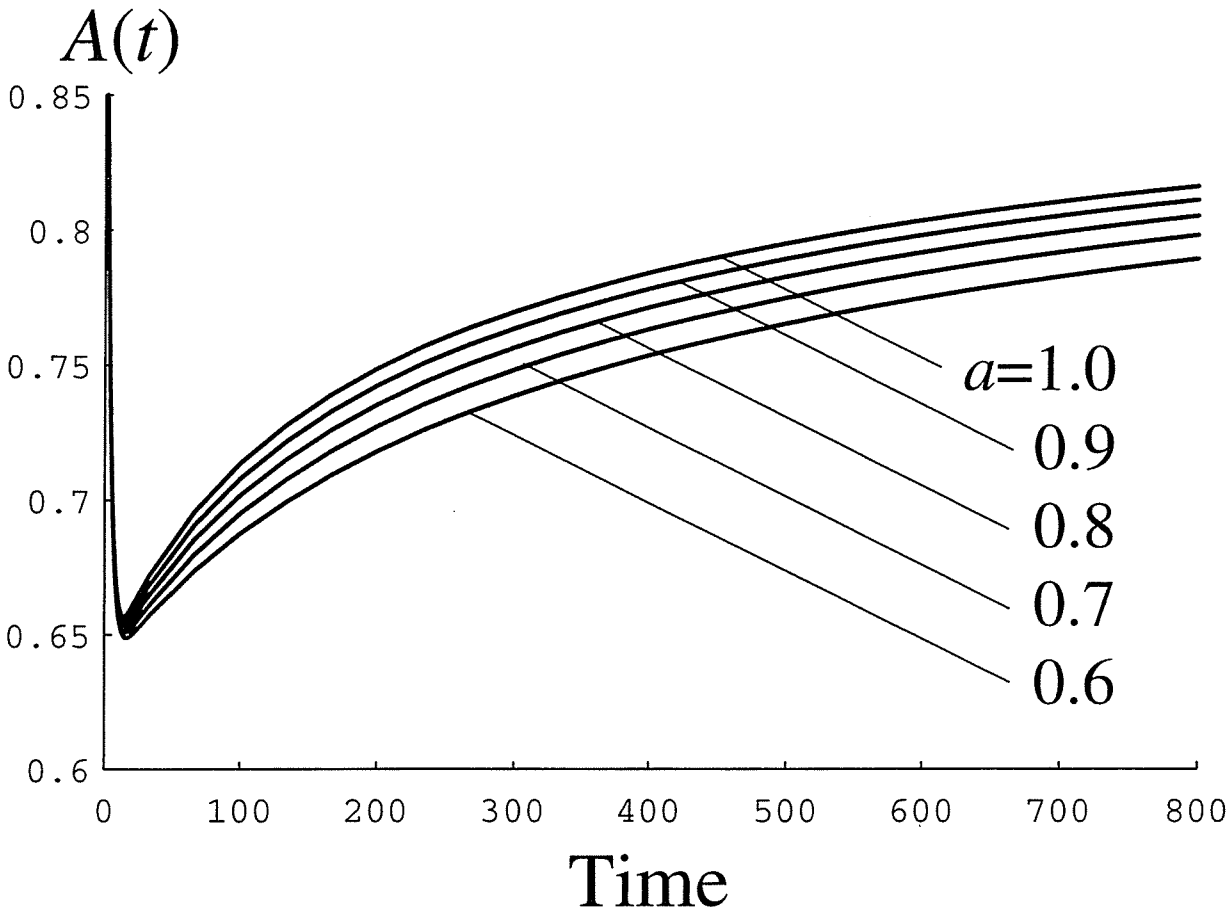

Fig. 10.7. Dependence of a on $A(t)$ ($D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.01$, $\eta = 0.1$).

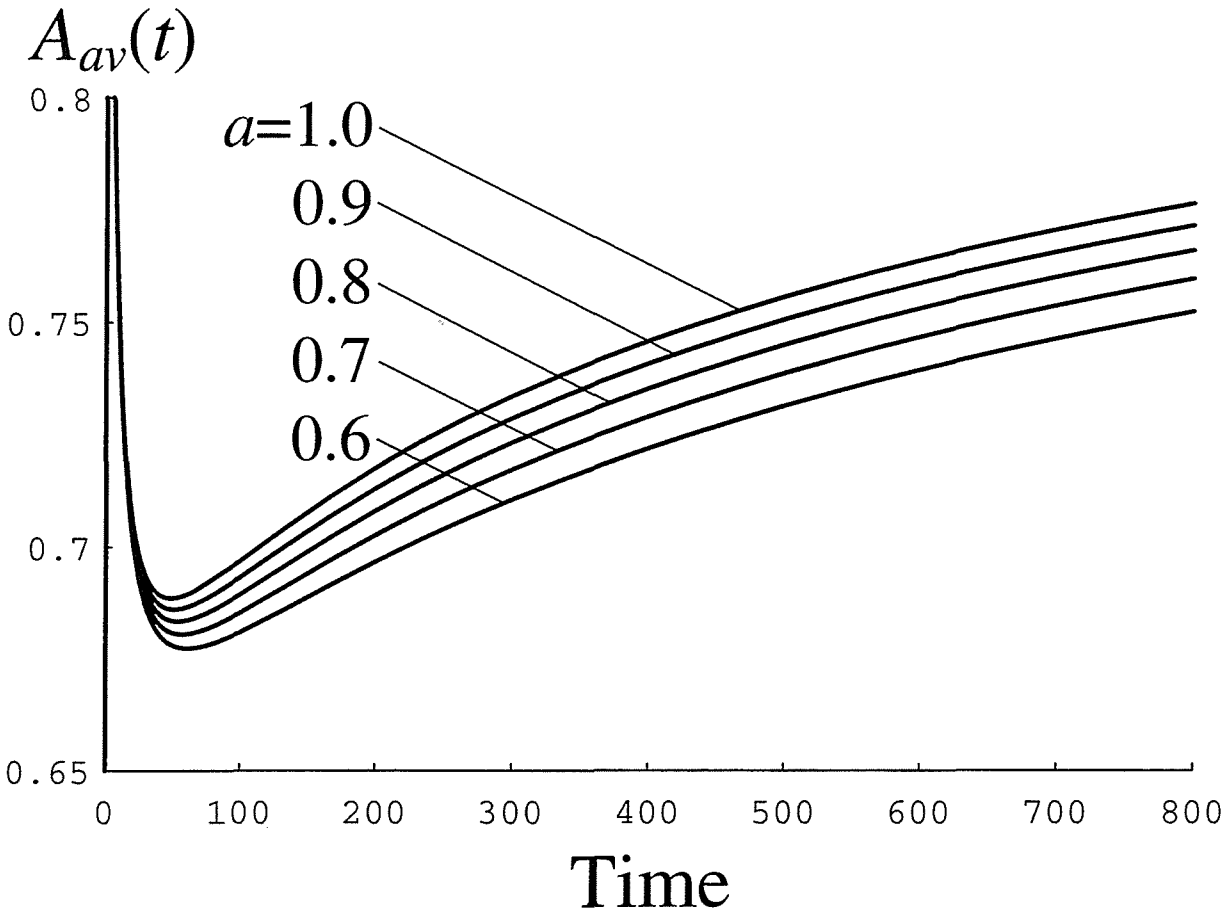

Fig. 10.8. Dependence of a on $A_{av}(t)$ ($D = 0.1$, $k = 0.8$, $E = 0.2$, $r = 0.9$, $\theta = 0.01$, $\eta = 0.1$).

10.5 Concluding Remarks

In this chapter, we have proposed two software safety assessment models: the software reliability assessment model with safety, and the availability-intensive safety assessment model. These have considered the random occurrences of hazardous conditions in system operation. The stochastic behaviors of the software system in dynamic environments have been described by Markov processes, involving the software reliability growth process. Several software safety/reliability assessment measures have been derived from these two models and the numerical examples of software safety/reliability measurement have been illustrated. These models can provide quantitative measures of software safety, which have scarcely been proposed so far. In particular, it is very meaningful that this work suggests

to enable quantitative assessment of simultaneous software safety and software availability, which are the customer-oriented quality characteristics.

Part IV

CLOSING

Chapter 11

Conclusion

This dissertation has provided several Markovian models for software reliability, availability, and safety measurement and assessment. The main contributions obtained and the future studies remaining in the respective parts are summarized as follows:

Part I: Software Reliability Modeling

Chapter 2 has given a software reliability model considering the imperfect debugging environment where faults detected are not always corrected and removed certainly. Optimal software release problems based on this model have also been discussed by introducing the total software cost and the reliability requirement. Chapter 3 has reconstructed the extended imperfect debugging model based on Chapter 2 by assuming that there exist the faults regenerated during the testing phase. Estimation of unknown parameters has been discussed and the application examples to the actual testing data have been presented in this chapter. Several quantitative measures useful for software reliability assessment have been derived from these models.

Estimation of cost parameters is not still established in optimal software release problems. Furthermore, the hazard rate for F2, θ, and the perfect debugging rate, p, have been prespecified in Chapter 3. An experimental method for setting θ and p is as follows: By analyzing the testing data observed from a similar software-production project experienced before, p can be set to the ratio of the number of faults corrected perfectly to the total number of software failures observed during testing time interval $(0, T]$, which is denoted as M_0. Moreover, the ratio of the number of F2 to M_0, which is denoted as r_0, is found. Since a lot of efforts are needed in analyzing a testing data, p and r_0 are also applied to similar new projects. If M software failures are observed during testing time interval $(0, T]$ of a new project, then we can determine that $\theta = (r_0 M)/T$. However, it is necessary to analyze testing data in detail in order to validate the assumption of two types of software

failures. Verification of the validity of this model remains a future problem.

Part II: Software Availability Modeling

The second part has discussed several stochastic modeling for software availability measurement and assessment and derived quantitative performance measures considering several operational environments. Chapter 4 has given the basic software availability model with only up and down state, which has described the software reliability-growth process as well. Chapters 5 and 6 have constructed the extended software availability models reflecting the user operational environment, which have been based on the model in Chapter 4. The existence of software failures due to operational uses deviating from the specification has explicitly been considered in Chapter 5 and the operational restoration policy without debugging in Chapter 6. Chapter 7 has developed the software availability model with computation by assuming that a system can have two different levels in user operation: full and degenerated performance level. This model has provided a performance measure considering both of reliability and computation. Such measures have seldom been proposed for software systems. Chapter 8 has presented availability modeling for a software-intensive computer system, i.e. a hardware-software system.

The unknown model parameters must be estimated based on the actual data in order to assess software availability with these models. For example, the value of p in Chapter 6 can be set to the ratio of the number of restoration actions with debugging to the total number of restoration actions by analyzing the field restoration-data observed from a similar software product developed before. We will also apply this value to similar new software products. But it is difficult to estimate model parameters immediately by using the testing or field data observed from new software products themselves. In particular, it is necessary to equip the data-collection procedures for measuring the actual restoration times during the operation phase. The future study is to establish the practical estimation method of unknown parameters. In Chapter 7 it seems that we can expand our model by considering multilevel performance degeneration. However, it may be too difficult to analyze the extended model by a Markov process.

Part III: Software Safety Modeling

The third part has dealt with stochastic modeling for software safety measurement based on the software reliability/availability models discussed in the preceding two parts. Chapter 9 has presented software safety modeling by assuming that software failure-occurrences may induce unsafe states. Chapter 10 has developed two software safety assessment models considering that a system in operation may induce unsafe states. The software safety metrics defined as the probability that a system does not fall into any unsafe states at a specified time point have been derived from those software safety models. It is very meaningful to have provided quantitative safety measures for software systems.

Generally, it is difficult to estimate the parameters related to safety; for example, p_1 and γ in Chapter 9 and θ and η in Chapter 10. We may be able to apply qualitative methods for software safety analysis such as FTA and FMEA in order to estimate such parameters. Reasonable estimation of those model parameters remains a future problem. Moreover, the models in Chapter 10 deal with software safety and reliability factors independently, i.e. θ and η have no concern with the cumulative number of corrected faults. However, there may be cases where these two factors have a strong relation. It is an interesting problem to construct models correlating safety factors with software reliability growth.

Most results derived in this dissertation have been obtained as closed forms. Thus, we can evaluate performance measures easily, specifying each model parameters. On the other hand, in general, various restrictions are more imposed on the development of models in mathematical modeling than simulation modeling. For example, it is assumed that the restoration times are distributed exponentially over this dissertation. However, it is not said that this assumption is realistic. If restoration times are assumed to be distributed generally, computation of performance measures seems to be much complicated. It is an interesting problem to compare analytical modeling with simulation one. Moreover, establishment of application of our models to practical problems concerned in the software project management remains as advanced research works.

References

[1] A. A. Abdel-Ghaly, P. Y. Chan and B. Littlewood, "Evaluation of competing software reliability predictions", *IEEE Trans. Software Engineering*, Vol. SE-12, No. 9, pp. 950–967, September 1986.

[2] A. Avižienis, "The N-version approach to fault-tolerant software", *IEEE Trans. Software Engineering*, Vol. SE-11, No. 12, pp. 1491–1501, December 1985.

[3] M. D. Beaudry, "Performance-related reliability measures for computing systems", *IEEE Trans. Computers*, Vol. C-27, No. 6, pp. 540–547, 1978.

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

[5] M. A. Friedman and J. M. Voas, *Software Assessment: Reliability, Safety, Testability*, John Wiley & Sons, New York, 1995.

[6] M. A. Friedman, P. Y. Tran and P. L. Goddard, *Reliability of Software Intensive Systems*, Notes Data Corporation, New Jersey, 1994.

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

[8] O. Gaudoin, C. Lavergne and J.-L. Soler, "A generalized geometric de-eutrophication software-reliability model", *IEEE Trans. Reliability*, Vol. 43, No. 4, pp. 536–541, December 1994.

[9] A. L. Goel, "Software reliability models: Assumptions, limitations, and applicability", *IEEE Trans. Software Engineering*, Vol. SE-11, No. 12, pp. 1411–1423, December 1985.

[10] A. L. Goel and K. Okumoto, "An imperfect debugging model for reliability and other quantitative measures of software systems", Technical Report Vol. 78-1, Department of Industrial Engineering and Operations Research, Syracuse University, New York, 1978.

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

[12] A. L. Goel and J. Soenjoto, "Models for hardware-software system operational-performance evaluation", *IEEE Trans. Reliability*, Vol. R-30, No. 3, pp. 232–239, August 1981.

[13] Z. Jelinski and P. B. Moranda, "Software reliability research", *Statistical Computer Performance Evaluation*, W. Freiberger, ed., pp. 465–484, Academic Press, New York, 1972.

[14] A. Kanno, *Software Quality Control* (in Japanese), JUSE, Tokyo, 1986.

[15] S. J. Keene, Jr., "Assuring software safety", *Proc. Annu. Reliability and Maintainability Symp.*, pp. 274–279, 1992.

[16] J. H. Kim, Y. H. Kim and C. J. Park, "A modified Markov model for the estimation of computer software performance", *Operations Research Letters*, Vol. 1, No. 6, pp. 253–257, December 1982.

[17] H. S. Koch and P. Kubat, "Optimal release time of computer software", *IEEE Trans. Software Engineering*, Vol. SE-9, No. 3, pp. 323–327, May 1983.

[18] W. Kremer, "Birth-death and bug counting", *IEEE Trans. Reliability*, Vol. R-32, No. 1, pp. 37–47, April 1983.

[19] J.-C. Laprie and K. Kanoun, "X-ware reliability and availability modeling", *IEEE Trans. Software Engineering*, Vol. 18, No. 2, pp. 130–147, February 1992.

[20] J.-C. Laprie, J. Arlat, C. Béounes and K. Kanoun, "Definition and analysis of hardware- and software-fault-tolerant architectures", *IEEE Computer Magazine*, Vol. 23, No. 7, pp. 39–51, July 1990.

[21] J.-C. Laprie, K. Kanoun, C. Béounes and M. Kaâniche, "The KAT (Knowledge-Action-Transformation) approach to the modeling and evaluation of reliability and availability growth", *IEEE Trans. Software Engineering*, Vol. 17, No. 4, pp. 370–382, April 1991.

[22] N. G. Leveson, "Software safety: Why, what, and how", *ACM Computing Surveys*, Vol. 18, No. 2, pp. 125–163, June 1986.

[23] N. G. Leveson, *Safeware: System Safety and Computers*, Addison-Wesley, New York, 1995.

[24] B. Littlewood and L. Strigini, "The risks of software", *Scientific American*, Vol. 267, No. 5, pp. 62–75, November 1992.

[25] M. R. Lyu, ed., *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[26] Y. K. Malaiya and P. K Srimani, eds., *Software Reliability Models: Theoretical Developments, Evaluation and Applications*, IEEE Computer Society Press, Los Alamitos, CA, 1991.

[27] J. J. Marciniak, ed., *Encyclopedia of Software Engineering* (Vols. 1 and 2), John Wiley & Sons, New York, 1994.

[28] P. Mellor, "Software reliability modeling: The state of the art", *Information and Software Technology*, Vol. 29, No. 2, pp. 81–98, March 1987.

[29] J. F. Meyer, "On evaluating the performability of degradable computing systems", *IEEE Trans. Computers*, Vol. C-29, No. 8, pp. 720–731, 1980.

[30] G. De Micheli, "A survey of problems and methods for computer-aided hardware/software co-design", *J. Information Processing Society of Japan*, Vol. 36, No. 7, pp. 605–613, July 1995.

[31] P. B. Moranda, "Event-altered rate models for general reliability analysis", *IEEE Trans. Reliability*, Vol. R-28, No. 5, pp. 376–381, December 1979.

[32] J. D. Musa and W. W. Everett, "Software-reliability engineering: Technology for the 1990s", *IEEE Computer Magazine*, Vol. 7, No. 6, pp. 36–43, November 1990.

[33] J. D. Musa and K. Okumoto, "A logarithmic Poisson execution time model for software reliability measurement", *Proc. 7th IEEE Int. Conf. Software Engineering*, pp. 230–238, 1984.

[34] J. D. Musa and K. Okumoto, "Application of basic and logarithmic Poisson execution time models in software reliability measurement", *Software System Design Method* (NATO ASI Series, Vol. F22), J. K. Skwirzynski, ed., pp. 275–298, Springer-Verlag, Berlin, 1986.

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

[36] Y. Nakagawa and I. Takenaka, "Error complexity model for software reliability estimation" (in Japanese), *Trans. IEICE D-I*, Vol. J74-D-I, No. 6, pp. 379–386, June 1991.

[37] M. Nakamura and S. Osaki, "Performance/reliability evaluation for multi-processor systems with computational demands", *Int. J. Systems Science*, Vol. 15, No. 1, pp. 95–105, January 1984.

[38] M. Ohba and X. Chou, "Does imperfect debugging affect software reliability growth?", *Proc. 11th IEEE Int. Conf. Software Engineering*, pp. 237–244, 1989.

[39] K. Okumoto and A. L. Goel, "Availability and other performance measures for system under imperfect maintenance", *Proc. COMPSAC '78*, pp. 66–71, 1978.

[40] K. Okumoto and A. L. Goel, "Optimum release time for software system based on reliability and cost criteria", *J. Systems and Software*, Vol. 1, No. 4, pp. 315–318, 1980.

[41] S. Osaki, *Stochastic System Reliability Modeling*, World Scientific, Singapore, 1985.

[42] S. Osaki, *Applied Stochastic System Modeling*, Springer-Verlag, Heidelberg, 1992.

[43] H. Pham, ed., *Software Reliability and Testing*, IEEE Computer Society Press, Los Alamitos, CA, 1995.

[44] C. V. Ramamoorthy and F. B. Bastani, "Software reliability status and perspectives", *IEEE Trans. Software Engineering*, Vol. SE-8, No. 4, pp. 345–371, July 1982.

[45] B. Randell, "System structure for software fault-tolerance", *IEEE Trans. Software Engineering*, Vol. SE-1, Vol. 2, pp. 220–232, June 1975.

[46] P. Rook, ed., *Software Reliability Handbook*, Elsevier Applied Science, London, 1990.

[47] S. M. Ross, *Stochastic Processes, Second Edition*, John Wiley & Sons, New York, 1996.

[48] J. G. Shanthikumar, "A state- and time-dependent error occurrence-rate software reliability model with imperfect debugging", *Proc. National Computer Conf.*, pp. 311–315, 1981.

[49] J. G. Shanthikumar, "Software reliability models: A review", *Microelectronics and Reliability*, Vol. 23, No. 5, pp. 903–943, 1983.

[50] M. L. Shooman, *Software Engineering: Design, Reliability, and Management*, McGraw-Hill, New York, 1983.

[51] A. Sols, "System degraded availability", *Reliability Engineering and System Safety*, Vol. 56, No. 1, pp. 91–94, 1997.

[52] U. Sumita and J. G. Shanthikumar, "A software reliability model with multiple-error introduction & removal", *IEEE Trans. Reliability*, Vol. R-35, No. 4, pp. 459–462, October 1986.

[53] W. A. Thompson, Jr., *Point Process Models with Applications to Safety and Reliability*, Chapman and Hall, New York, 1988,

[54] M. Xie, *Software Reliability Modelling*, World Scientific, Singapore, 1991.

[55] S. Yamada, *Software Reliability Assessment Technology* (in Japanese), HBJ Japan, Tokyo, 1989.

[56] S. Yamada, "Software quality/reliability measurement and assessment: Software reliability growth models and data analysis", *J. Information Processing*, Vol. 14, No. 3, pp. 254–266, 1991.

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

[58] S. Yamada, "Software reliability/safety assessment" (in Japanese), *J. Japan Society for Safety Engineering*, Vol. 33, No. 6, pp. 432–441, December 1994.

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

[60] S. Yamada and S. Osaki, "Optimal software release policies with simultaneous cost and reliability requirements", *European J. Operational Research*, Vol. 31, pp. 46–51, 1987.

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

[62] S. Yamada, T. Ichimori and H. Masuyama, "Optimal release problems on software failure time-measuring reliability models" (in Japanese), *Trans. IEICE A*, Vol. J73-A, No. 6, pp. 1117–1122, June 1990.

[63] S. Yamada, M. Kimura and M. Takahashi, *Statistical Quality Control for TQM — Applying SQC Methods in a Wide Variety of Both General Industrial Products and Software Products* (in Japanese), Corona Publishing, Tokyo, 1998.

[64] S. Yamada, T. Yamane and S. Osaki, "Software reliability growth models with error debugging rate" (in Japanese), *Trans. IPS Japan*, Vol. 27, No. 1, pp. 64–71, January 1986.

Publication List of the Author

1. Koichi Tokuno, Shigeru Yamada and Shunji Osaki, "A Markovian imperfect debugging model for software reliability measurement", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E75-A, No. 11, pp. 1590–1596, November 1992.

2. Shigeru Yamada, Koichi Tokuno and Shunji Osaki, "Imperfect debugging models with fault introduction rate for software reliability assessment", International Journal of Systems Science, Vol. 23, No. 12, pp. 2241–2252, December 1992.

3. Shigeru Yamada, Koichi Tokuno and Shunji Osaki, "Software reliability measurement in imperfect debugging environment and its application", Reliability Engineering and System Safety, Vol. 40, No. 2, pp. 139–147, May 1993.

4. Koichi Tokuno and Shigeru Yamada, "A Markovian software availability modeling and measurement" (in Japanese), in the Proceedings of the 15th Software Reliability Symposium, pp. 86–92, Osaka, Japan, December 1994.

5. Koichi Tokuno and Shigeru Yamada, "A Markovian software availability measurement with a geometrically decreasing failure-occurrence rate", IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E78-A, No. 6, pp. 737–741, June 1995.

6. Koichi Tokuno and Shigeru Yamada, "A Markovian modeling for software availability measurement" (in Japanese), in Foundation of Software Engineering II (Kazuhito Ohmaki ed.), pp. 159–164, Kindai-Kagakusha, January 1996.

7. Koichi Tokuno and Shigeru Yamada, "A software availability model with imperfect debugging environment" (in Japanese), in the Proceedings of the 16th Software Reliability Symposium, pp. 72–77, Kyoto, Japan, February 1996.

8. Koichi Tokuno and Shigeru Yamada, "A software reliability growth model with two types of software failure-occurrences" (in Japanese), in the Proceedings of Software Symposium '96, pp. 16–24, Hiroshima, Japan, June 1996.

9. Koichi Tokuno and Shigeru Yamada, "Imperfect debugging modelling with two types of failure-occurrence rates for software reliability measurement", in the Proceedings of the Second Australia-Japan Workshop on Stochastic Models in Engineering, Technology and Management (Richard J. Wilson, D. N. Pra Murthy, and Shunji Osaki eds.), pp. 621–630, Gold Coast, Australia, July 1996.

10. Keiko Takata, Koichi Tokuno and Shigeru Yamada, "Markovian software availability modeling with a geometrically decreasing hazard rate", in the Proceedings of the Third China-Japan International Symposium on Industrial Management (ISIM '96) (Feng Yuncheng and Hirokazu Osaki eds.), pp. 544–549, Nanjing, China, October 1996.

11. Koichi Tokuno and Shigeru Yamada, "Markovian software availability modeling for performance evaluation", in Stochastic Modelling in Innovative Manufacturing: Proceedings, Cambridge, U.K., July 21-22, 1995 (Lecture Notes in Economics and Mathematical Systems 445) (Anthony H. Christer, Shunji Osaki, and Lyn C. Thomas eds.), pp. 246–256, Springer-Verlag, Berlin, 1997.

12. Koichi Tokuno and Shigeru Yamada, "Software availability model with a decreasing fault-correction rate" (in Japanese), Reliability (the Journal of Reliability Engineering Association of Japan), Vol. 19, No. 1, pp. 3–12, January 1997.

13. Koichi Tokuno and Shigeru Yamada, "Markovian software availability modeling with two types of software failures for operational use", in the Proceedings of the Third ISSAT International Conference on Reliability and Quality in Design (RQD '97) (Hoang Pham ed.), pp. 97–101, Anaheim, California, U.S.A., March 1997.

14. Koichi Tokuno and Shigeru Yamada, "A Markovian modeling for software availability measurement" (in Japanese), Computer Software (the Journal of Japan Society for Software Science and Technology), Vol. 14, No. 2, pp. 38–44, March 1997.

15. Koichi Tokuno and Shigeru Yamada, "Markovian availability modelling for computer-based system with software failure-occurrence", in the Proceedings of the Second International Conference on Quality and Reliability (ICQR '97) (Albert H. C. Tsang and C. Y. Tang eds.), Vol. 1, pp. 397–403, Hong Kong, September 1997.

16. Koichi Tokuno and Shigeru Yamada, "Markovian availability measurement and assessment for hardware-software system", International Journal of Reliability, Quality and Safety Engineering, Vol. 4, No 3, pp. 257–268, September 1997.

17. Koichi Tokuno and Shigeru Yamada, "Operational availability measurement with two types of software failures" (in Japanese), in Foundation of Software Engineering IV (Yoshiaki Fukazawa and Mikio Aoyama eds.), pp. 95–98, Kindai-Kagakusha, December 1997.

18. Shigeru Yamada, Koichi Tokuno and Yu Kasano, "Quantitative assessment models for software safety/reliability" (in Japanese), the Transactions of IEICE A, Vol. J80-A, No. 12, pp. 2127–2137, December 1997, and Electronics and Communications in Japan, Part 2, Vol. 81, No. 5, pp. 33–43, 1998.

19. Koichi Tokuno and Shigeru Yamada, "A Markovian software availability model for operational use" (in Japanese), Computer Software (the Journal of Japan Society for Software Science and Technology), Vol. 15, No. 3, pp. 17–24, May 1998.

20. Koichi Tokuno and Shigeru Yamada, "Markovian software availability modeling with degenerated performance", in the Proceedings of the European Conference on Safety and Reliability (ESREL '98) (Stian Lydersen, Geir K. Hansen and Helge A. Sandtorv eds.), Vol. 1, pp. 425–431, Trondheim, Norway, June 1998.

21. Koichi Tokuno and Shigeru Yamada, "Markovian software safety/reliability measurement with imperfect debugging", in the Proceedings of the Fourth ISSAT International Conference on Reliability and Quality in Design (RQD '98) (Hoang Pham and Ming-Wei Lu eds.), pp. 56–60, Seattle, Washington, U.S.A., August 1998.

22. Takashi Miki, Koichi Tokuno and Shigeru Yamada, "Imperfect debugging models with introduced software faults and their comparisons", in the Proceedings of the

Fourth China-Japan International Symposium on Industrial Management (ISIM '98) (Feng Yuncheng and Hirokazu Osaki eds.), pp. 278–283, Dalian, China, October 1998.

23. Koichi Tokuno and Shigeru Yamada, "Operational software availability measurement with two kinds of restoration actions", Journal of Quality in Maintenance Engineering, Vol. 4, No. 4, pp. 273–283, October 1998.

24. Koichi Tokuno and Shigeru Yamada, "Software availability modeling with two restoration actions for operational use" (in Japanese), in the Proceedings of the 18th Symposium on Quality Control of Software Production, pp. 193–200, Tokyo, Japan, November 1998.

25. Shigeru Yamada, Koichi Tokuno and Kei Inoue, "Optimal release problems with software reliability/safety based on cost criteria" (in Japanese), the Transactions of IEICE A, Vol. J82-A, No. 1, pp. 64–72, January 1999.

26. Koichi Tokuno and Shigeru Yamada, "An imperfect debugging model with two types of hazard rates for software reliability measurement and assessment", Mathematical and Computer Modelling, to be published in 1999.

27. Koichi Tokuno and Shigeru Yamada, "Markovian reliability modeling for software safety/availability measurement", in Recent Advances in Reliability and Quality Engineering (Hoang Pham ed.), World Scientific Publishers, to be published in 1999.

28. Koichi Tokuno and Shigeru Yamada, "Stochastic software safety/reliability measurement and its application", Annals of Software Engineering, to be published.

END