Software Reliability Modeling Research and Its Brief History in My Career

山田　茂
鳥取大学大学院工学研究科社会経営工学講座

Shigeru YAMADA
Department of Social Management Engineering, Graduate School of Engineering
Tottori University, Tottori, 680-8552 Japan
E-mail: yamada@sse.tottori-u.ac.jp

Abstract: Software reliability is one of the most important characteristics of software quality. Its measurement and management technologies during the software life-cycle are essential to produce and maintain quality/reliable software systems. In this paper, we discuss software reliability modeling and its applications. As to software reliability modeling, hazard rate and NHPP models are investigated particularly for quantitative software reliability assessment. Further, imperfect debugging and software availability models are also discussed with reference to incorporating practical factors of dynamic software behavior. Finally, the history of my software reliability modeling research effort is briefly summarized in Appendix.

Key Words: Product quality/reliability assessment, Software reliability growth modeling, Hazard rate, Nonhomogeneous Poisson process, Imperfect debugging, Software availability, Markov process.

1. Introduction

A *software reliability model* (*SRM*) is a mathematical analysis model for the purpose of measuring and assessing software quality/reliability quantitatively. Many software reliability models have been proposed and applied to practical use because software reliability is considered to be a "*must-be quality*" characteristic of a software product. The software reliability models can be divided into two classes as shown in Figure 1. One treats the upper software development process, i.e., design and coding phases, and analyzes the reliability factors of the software products and processes, which is categorized in the class of static model. The other deals with testing and operation phases by describing a software failure-occurrence phenomenon or software fault-detection phenomenon, by applying the stochastic/statistics theories and can estimate and predict the software reliability, which is categorized in dynamic model.

In the former class, a *software complexity model* is well known and can measure the reliability by assessing the complexity based on the structural characteristics of products and the process features to produce the products. In the latter class, a *software reliability growth model* is especially well known. Further, this model is divided into three categories:

(1) Software failure-occurrence time model
The model which is based on the software failure-occurrence time or the software fault-detection time.

(2) Software fault-detection count model
The model which is based on the number of software failure-occurrence or the number of detected faults.

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

The software reliability growth models are utilized for assessing the degree of achievement of software quality, deciding the time to software release for operational use, and evaluating the maintenance cost for faults undetected during the testing phase. We discuss the software reliability growth models.

2. Software Reliability Growth Modeling

Generally, a mathematical model based on stochastic and statistical theories is useful to describe the software fault-detection phenomena or the software failure-occurrence phenomena and estimate the

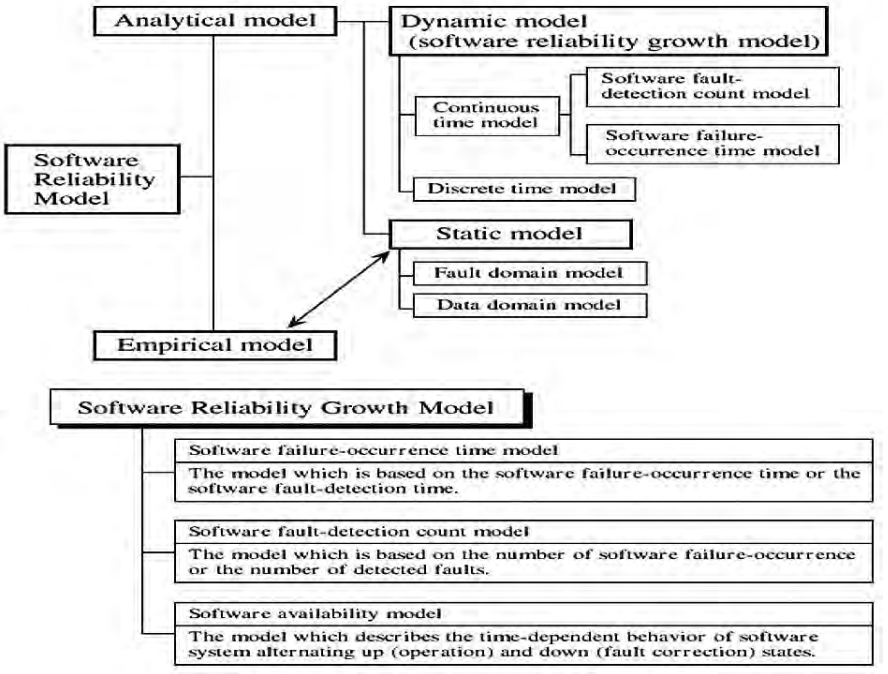

Fig. 1 Hierarchical classification of software reliability model.

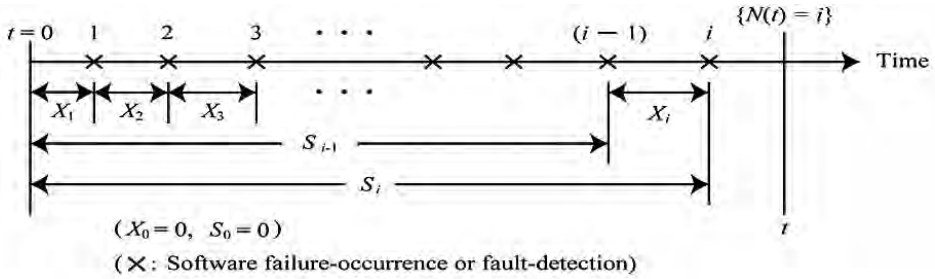

Fig. 2 The stochastic quantities related to a software fault-detection phenomenon or a software failure-occurrence phenomenon.

software reliability quantitatively. During the testing phase in the software development process, software faults are detected and removed with a lot of testing-effort expenditures. Then, the number of faults remaining in the software system decreases as the testing goes on. This means that the probability of software failure-occurrence is decreasing, so that the software reliability is increasing and the time interval between software failures becoming longer with the testing time.

A mathematical tool which describes software reliability aspect is a *software reliability growth model*.

Based on the plausible definitions, we can develop a software reliability growth model based on the assumptions used for the actual environment during the testing phase or the operation phase. Then, we can define the following random variables on the number of detected faults and the software failure-occurrence time (see Figure 2):

$N(t)$ = the cumulative number of software faults (or the cumulative number of observed software failures) detected up to time *t*,

S_i = the *i*-th software-failure occurrence time $(i=1,2,\cdots; S_0=0)$,

X_i = the time interval between (i-1)-st and i-th software failures $(i=1,2,\cdots; X_0=0)$.

Figure 2 shows the occurrence of event $\{N(t)=i\}$ since *i* faults have been detected up to time *t*. From these definitions, we have

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

Assuming that the hazard rate, i.e., the software failure rate, for $X_i (i=1,2,\cdots)$, $z_i(x)$, is proportional to the current number of residual faults remaining in the system, we have

$$z_i(x) = (N-i+1)\lambda(x) \quad (i=1,2,\cdots,N; x \geq 0, \lambda > 0), \tag{2}$$

where N is the initial fault content and $\lambda(x)$ the software failure rate per fault remaining in the system at time x. If we consider two special cases in Eq. (2) as

$$\lambda(x) = \phi \quad (\phi > 0), \tag{3}$$

$$\lambda(x) = \phi x^{m-1} \quad (\phi > 0, m > 0), \tag{4}$$

then two typical software hazard rate models, respectively called the Jelinski-Moranda model and the Wagoner model can be derived, where ϕ and m are constant parameters. Usually, it is difficult to assume that a software system is completely fault free or failure free. Then, we have a software hazard rate model called the Moranda model for the case of the infinite number of software failure occurrences as

$$z_i(x) = Dk^{i-1} \quad (i=1,2,\cdots; D>0; 0<k<1), \tag{5}$$

where D is the initial software hazard rate and k the decreasing ratio. Eq. (5) 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 thereafter. Based on the software hazard rate models above, we can derive several software reliability assessment measures. For example, the *software reliability function* for is given as

$$R_i(x) = \exp\left[-\int_0^x z_i(x)dx\right] \quad (i=1,2,\cdots). \tag{6}$$

Further, we also discuss NHPP models, which are modeled for random variable $N(t)$ as typical software reliability growth models. In the NHPP models, a *nonhomogeneous Poisson process* (*NHPP*) is assumed for the random variable $N(t)$, the distribution function of which is given by

$$\Pr\{N(t)=n\} = \frac{\{H(t)\}^n}{n!}\exp[-H(t)] \quad (n=1,2,\cdots),$$

$$H(t) \equiv E[N(t)] = \int_0^t h(x)dx, \tag{7}$$

where $\Pr[\cdot]$ and $E[\cdot]$ mean the probability and expectation, respectively. $H(t)$ in Eq. (7) is called a mean value function which indicates the expectation of $N(t)$, *i.e.*, the expected cumulative number of faults detected (or the expected cumulative number of software failures occurred) in the time interval $(0,t]$, and $h(t)$ in Eq. (7) called an *intensity function* which indicates the instantaneous fault-detection rate at time t.

From Eq. (7), various software reliability assessment measures can be derived. For examples, the expected number of faults remaining in the system at time t is given by

$$n(t) = a - H(t), \tag{8}$$

where $a \equiv H(\infty)$, *i.e.*, parameter a denotes the expected initial fault content in the software system. Given that the testing or the operation has been going up to time t, the probability that a software failure does not occur in the time interval $(t, t+x](x \geq 0)$ is given by conditional probability $\Pr\{X_i > x \mid S_{i-1} = t\}$ as

$$R(x \mid t) = \exp[H(t) - H(x+t)] \quad (t \geq 0, x \geq 0). \tag{9}$$

$R(x \mid t)$ in Eq. (9) is a so-called *software reliability*. Measures of *MTBF* (*mean time between software failures* or *fault-detections*) can be obtained follows:

$$MTBF_I(t) = \frac{1}{h(t)}, \tag{10}$$

$$MTBF_C(t) = \frac{t}{H(t)}. \tag{11}$$

MTBFs in Eqs. (10) and (11) are called *instantaneous and cumulative MTBFs*, respectively.

It is obvious that the lower the value of $n(t)$ in Eq. (8), the higher the value for specified x in Eq. (9), or the longer the value of *MTBFs* in Eqs. (10) and (11), the higher the achieved software reliability is. Then, analyzing actual test data with accepted NHPP models, these measures can be utilized to assess software reliability during the testing or operation phase, where statistical inferences, *i.e.*, parameter estimation and goodness-of-fit test, are usually performed by a method of maximum likelihood.

Table 1 A summary of NHPP models.

NHPP model	Mean value function $H(t)$	Intensity function $h(t)$	Environment
Exponential software reliability growth model	$m(t) = a(1 - e^{-bt})$ $(a > 0, b > 0)$	$h_m(t) = a(1 - e^{-bt})$	A software failure-occurrence phenomenon with a constant fault-detection rate at an arbitrary time is described.
Modified exponential software reliability growth model	$m_p(t) = a\sum_{i=1}^{2} p_i(1 - e^{-b_i t})$ $(a > 0, 0 < b_2 < b_1 < 1,$ $\sum_{i=1}^{2} p_i = 1, 0 < p_i < 1)$	$h_p(t) = a\sum_{i=1}^{2} p_i b_i e^{-b_i t}$	A difficulty of software fault-detection during the testing is considered. (b_1 is the fault-detection rate for easily detectable faults: b_2 is the fault-detection rate for hardly detectable faults)
Delayed S-shaped software reliability growth model	$M(t) = a[1 - (1 + bt)e^{-bt}]$ $(a > 0, b > 0)$	$h_M(t) = ab^2 te^{-bt}$	A software fault-detection process is described by two successive phenomena, i.e., failure-detection process and fault-isolation process.
Inflection S-shaped software reliability growth model	$I(t) = \frac{a(1-e^{-bt})}{1+ce^{-bt}}$ $(a > 0, b > 0, c > 0)$	$h_I(t) = \frac{ab(1+c)e^{-bt}}{(1+ce^{-bt})^2}$	A software failure-occurrence phenomenon with mutual dependency of detected faults is described.
Testing-effort-dependent software reliability growth model	$T(t) = a[1 - e^{-rW(t)}]$ $W(t) = \alpha(1 - e^{-\beta t^m})$ $(a > 0, 0 < r < 1, \alpha > 0,$ $\beta > 0, m > 0)$	$h_T(t) = ar\alpha$ $\cdot\beta m t^{m-1} e^{-rW(t)}$	The time-dependent behavior of the amount of testing effort and the cumulative number of detected faults is considered.
Testing-domain-dependent software reliability growth model	$D(t) = a[1 - \frac{1}{v-b}(ve^{-bt} - be^{-vt})]$ $(v \neq b)$	$h_D(t) = \frac{avb}{v-b}$ $(e^{-bt} - e^{-vt})$	The testing domain, which is the set of software functions influenced by executed test cases, is considered.
Logarithmic Poisson execution time model	$\mu(t) = \frac{1}{\theta}\ln(\lambda_0\theta t + 1)$ $(\lambda_0 > 0, \theta > 0)$	$\lambda(t) = \frac{\lambda_0}{\lambda_0\theta t + 1}$	When the testing or operation time is measured on the basis of the number of CPU hours, and exponentially decreasing software failure rate is considered with respect to the cumulative number of software failures.

a = the expected number of initial fault content in the software system.
b, b_i, r = the parameter representing the fault-detection rate.
c = the parameter representing the inflection factor of test personnel.
p_i = the fault content ratio of Type i fault ($i = 1, 2$).
α, β, m = the parameters which determine the testing-effort function $W(t)$.
v = the testing-domain growth rate.
λ_0 = the initial software failure rate.
θ = the reduction rate of software failure rate.

To assess the software reliability actually, it is necessary to specify the mean value function $H(t)$ in Eq. (7). Many NHPP models considering the various testing or operation environments for software reliability assessment have been proposed in the last decade. Typical NHPP models are summarized in Table 1. As discussed above, a software reliability growth is described as the relationship between the elapsed testing or operation time and the cumulative number of detected faults and can be shown as the reliability growth curve mathematically. Among the NHPP models in Table 1, exponential and modified exponential software reliability growth models are appropriate when the observed reliability growth curve shows an exponential curve. Similarly, delayed S-shaped and inflection S-shaped software reliability growth models are appropriate when the reliability growth curve is S-shaped.

In addition, as for computer makers or software houses in Japan, *logistic curve* and *Gompertz curve models* have often been used as software quality assessment models, on the assumption that software fault-detection phenomena can be shown by S-shaped reliability growth curves. In these deterministic models, the cumulative number of faults detected up to testing t is formulated by the following growth equations:

$$L(t) = \frac{k}{1 + me^{-\alpha t}} \quad (m > 0, \alpha > 0, k > 0). \tag{12}$$

$$G(t) = ka^{(b^t)} \quad (0 < a < 1, 0 < b < 1, k > 0). \tag{13}$$

In Eqs. (12) and (13), assuming that a convergence value of each curve ($L(\infty)$ or $G(\infty)$), i.e., parameter *k*, represents the initial fault content in the software system, it can be estimated by a regression analysis.

3. IMPERFECT DEBUGGING MODELING

Most software reliability growth models proposed so far are based on the assumption of perfect debugging, i.e., that all faults detected during the testing and operation phases are corrected and removed perfectly. However, debugging actions in real testing and operation environments are not always performed perfectly. For example, typing errors invalidate the fault-correction activity or fault-removal is not carried out precisely due to incorrect analysis of test results. We therefore have an interest in developing a software reliability growth model which assumes an *imperfect debugging* environment. Such an imperfect debugging model is expected to estimate reliability assessment measures more accurately.

A. Imperfect debugging model with perfect correction rate

To model an imperfect debugging environment, the following assumptions are made:

(1) Each fault which causes a software failures is corrected perfectly with probability $p(0 \le p \le 1)$. It is not corrected with probability $q(=1-p)$. We call p the perfect debugging rate or the perfect correction rate.
(2) The hazard rate is given by Eq. (5) and decrease geometrically each time a detected fault is corrected (see Figure 3).
(3) The probability that two or more software failures occur simultaneously is negligible.
(4) No new faults are introduced during the debugging. At most one fault is removed when it is corrected, and the correction time is not considered.

Let $X(t)$ be a random variable representing the cumulative number of faults corrected up to the testing time *t*. Then, $X(t)$ forms a *Markov process*. That is, from assumption (1), when *i* faults have been corrected by arbitrary testing time *t*,

$$X(t) = \begin{cases} i, & \text{with probability } q, \\ i+1, & \text{with probability } p, \end{cases} \tag{14}$$

(see Fig. 4). Then, the one-step transition probability for the Markov process that after making a transition

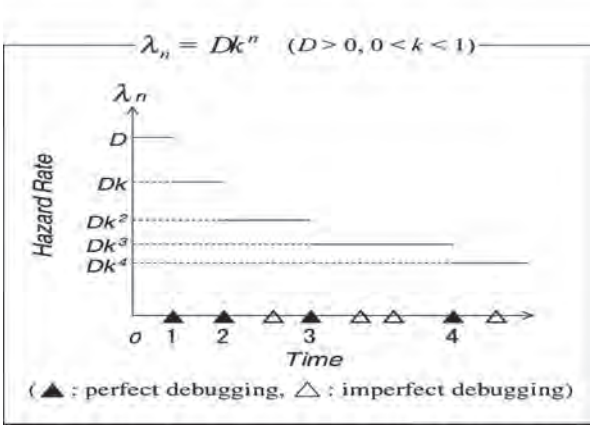

Fig. 3 Behavior of hazard rate.

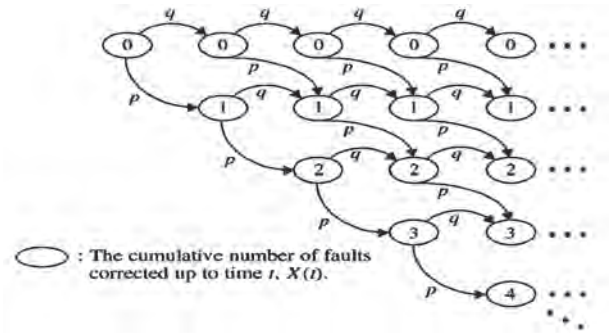

Fig. 4 A diagrammatic representation of transitions between states of $X(t)$.

into state *i*, the process $\{X(t), t \ge 0\}$ makes a transition into state *j* by time *t* is given by

$$Q_{ij}(t) = p_{ij}(1 - \exp[-Dk^i t]), \tag{15}$$

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

$$p_{ij} = \begin{cases} q, & (i = j) \\ p, & (i, j = 0,1,2,\cdots) \\ 0, & (elsewhere). \end{cases} \tag{16}$$

Eq. (15) represents the probability that if *i* faults have been corrected at time zero, *j* faults are corrected by time *t* after the next software failure occurs. Therefore, based on Markov analysis by using the assumptions and stochastic quantities above, we have the software reliability function and the mean time between software failures for $X_i(i = 1,2,\cdots)$ as

$$R_i(x) = \sum_{s=0}^{i-1} \binom{i-1}{s} p^s q^{i-1-s} \exp[-Dk^s x], \tag{17}$$

$$E[X_i]=\int_0^\infty R_i(x)dx=\frac{\left(\frac{p}{k}+q\right)^{i-1}}{D}. \tag{18}$$

And if the initial fault content in the system, N, is specified the expected cumulative number of faults debugged imperfectly up to time t is given by

$$M(t)=\frac{q}{p}\sum_{n=1}^{N}\sum_{i=0}^{n-1}A_{i,n}(1-\exp[-pDk^i t]), \tag{19}$$

where $A_{i,n}$ is

$$\left.\begin{aligned} &A_{0,1}\equiv 1\\ &A_{i,n}=\frac{k^{(1/2)n(n-1)i}}{\prod_{\substack{j=0\\ j\neq i}}^{n-1}(k^j-k^i)}\\ &(n=2,3,\cdots;\ i=0,1,2,\cdots,n-1)\end{aligned}\right\}. \tag{20}$$

B. Imperfect debugging model for introduced faults

Besides the imperfect debugging factor above in fault-correction activities, we consider the possibility of introducing new faults in the debugging process. It is assumed that the following two kinds of software failures exist in the dynamic environment, *i.e.*, the testing or user operation phase:

(F1) software failures caused by faults originally latent in the software system prior to the testing (which are called inherent faults),

(F2) software failures caused by faults introduced during the software operation owing to imperfect debugging.

In addition, it is assumed that one software failure is caused by one fault and that it is impossible to discriminate whether the fault that caused the software failure that has occurred is F1 or F2. As to the software failure-occurrence rate due to F1, the inherent faults are detected with the progress of the operation time. In order to consider two kinds of time dependencies on the decreases of F1, let $a_i(t)(i=1,2)$ denote the software failure-occurrence rate for F1. On the other hand, the software failure-occurrence rate due to F2 is denoted as constant $\lambda(\lambda>0)$, since we assume that F2 occurs randomly throughout the operation. When we consider the software failure-occurrence rate at operation time t is given by

$$h_i(t)=\lambda+a_i(t) \qquad (i=1,2). \tag{21}$$

From Eq. (21), the expected cumulative number of software failures in the time interval $(0,t]$ (or the expected cumulative number of detected faults) is given by

$$\left.\begin{aligned} &H_i(t)=\lambda t+A_i(t),\\ &A_i(t)=\int_0^t a_i(x)dx \quad (i=1,2)\end{aligned}\right\}. \tag{22}$$

Then, we have two imperfect debugging models based on an NHPP, where $h_i(t)$ in Eq. (21) and $H_i(t)$ in Eq. (22) are used as the intensity functions and the mean value functions (i=1,2) for an NHPP, respectively. Especially, exponential and delayed S-shaped software reliability growth models are assumed for describing software failure-occurrence phenomena attributable to the inherent faults (see Table 1).

4. SOFTWARE AVAILABILITY MODELING

Recently, software performance measures such as the possible utilization factors have begun to be interesting for metrics as well as the hardware products. That is, 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 specification, at a specified time point. Several stochastic models have been proposed so far for software availability measurement and assessment. One group has proposed a software availability model considering a reliability growth process, taking account of the cumulative number of corrected faults. Others have constructed software availability models describing the uncertainty of fault removal. Still others have incorporated the increasing difficulty of fault removal.

The actual operational environment needs to be more clearly reflected in software availability modeling, since software availability is a customer-oriented metrics. In existing models the development of a plausible model is described, which assumes that there exist two types of software failure occurring during the operation phase. Furthermore, an operational software availability model is built up from the viewpoint of restoration scenarios.

The above models have employed Markov processes for describing the stochastic time-dependent behaviors of the systems which alternate between the up state, operating regularly, and the restoration state (down state) when a system is inoperable. Several stochastic metrics for software availability measurement in dynamic environment are derived from the respective models.

We discuss a fundamental software availability model below.

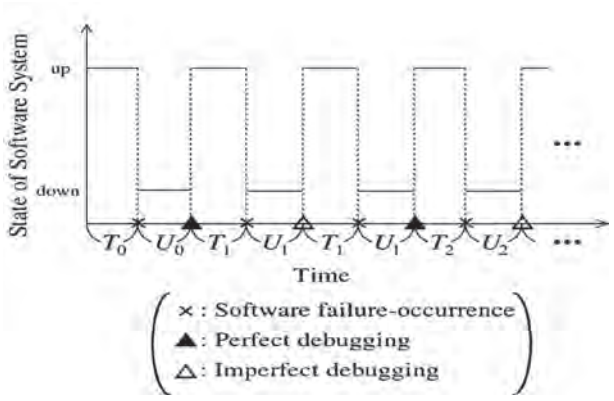

Fig. 5 Sample behavior of the software system alternating between up and down state.

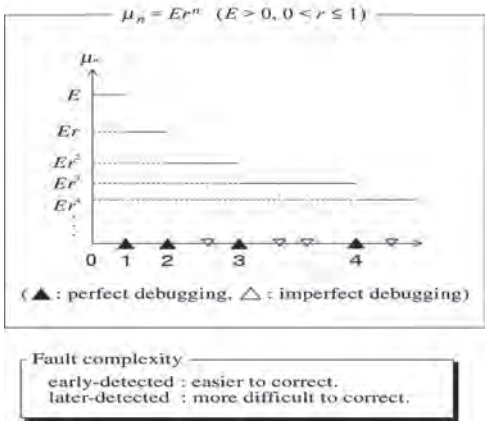

Fig. 6 Behavior of restoration rate.

A. Model description

The following assumptions are made for software availability modeling:

(1) The software system is unavailable and starts to be restored as soon as a software failure occurs, and the system cannot operate until the restoration action is complete (see Figure 5).

(2) The restoration action implies debugging activity, which is performed perfectly with probability $a\,(0<a\le 1)$ 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.

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

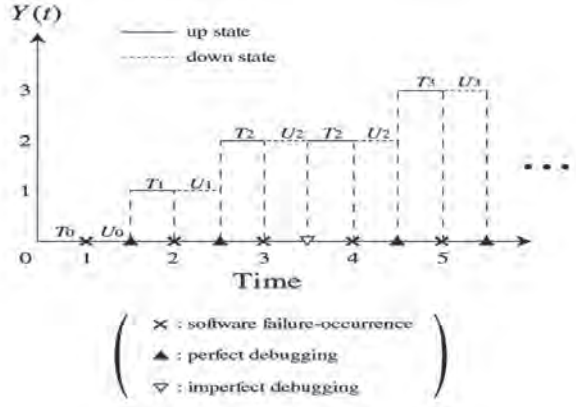

Fig. 7 A sample realization of $Y(t)$.

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

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

From assumption (2), when the restoration action has been completed in $\{X(t)=R_n\}$,

$$X(t)=\begin{cases} W_n, & \textit{with probability } b, \\ W_{n+1}, & \textit{with probability } a, \end{cases} \tag{23}$$

We use the Moranda model in Eq. (5) to describe the software failure-occurrence phenomenon, *i.e.*, when n faults have been corrected, the software hazard rate λ_n (see Figure 3) is given by

$$\lambda_n = Dk^n \qquad (n=0,1,2,\cdots;\ D>0,\ 0<k<1) \tag{24}$$

The expression of Eq. (24) 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.

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 complexity, there are cases where the faults detected during the early stage of the testing or operation phase have low complexity and are easy to

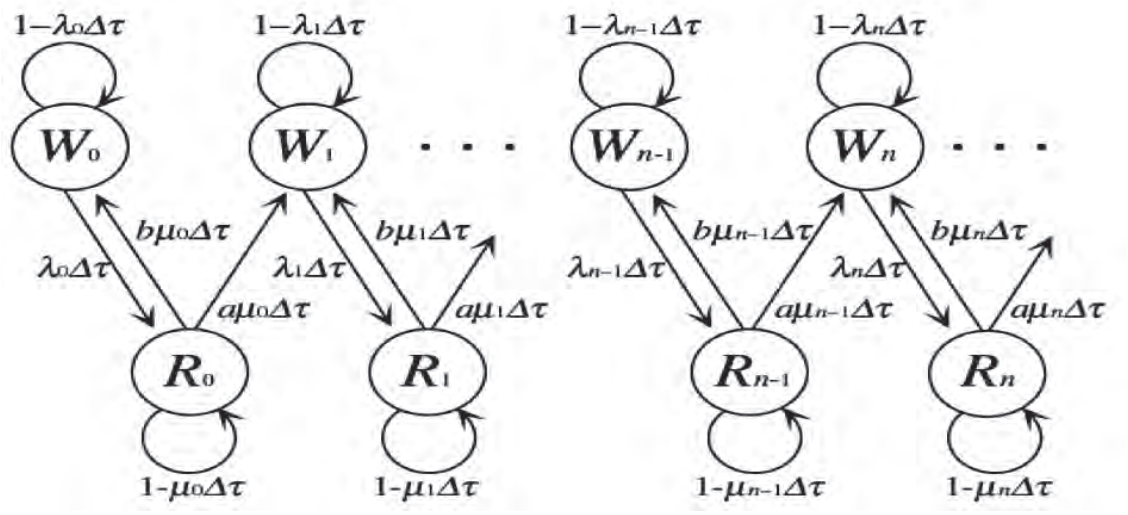

Fig. 8 A state transition diagram for software availability modeling.

correct/remove, and as the testing is in progress, detected faults have higher complexity and are more difficult to correct/remove. In the above case, it is appropriate that the mean restoration time becomes longer with the increasing number of corrected faults. Accordingly, we express μ_n as follows (see Figure 6):

$$\mu_n = Er^n \quad (n = 0,1,2,\cdots;\ E > 0,\ 0 < r < 1) \tag{25}$$

where E and r are the initial restoration rate and the decreasing ratio of the restoration rate, respectively. In Eq. (25) the case of r=1, *i.e.*, $\mu_n = E$, means that the complexity of each fault is random.

Let T_n and U_n $(n = 0,1,2,\cdots)$ be the random variables representing the next software failure occurrence and the next restoration time intervals when n faults have been corrected, in other words the sojourn times in states W_n and R_n, respectively. Furthermore, let $Y(t)$ be the random variable representing the cumulative number of faults corrected up to time t. The sample behavior of $Y(t)$ is illustrated in Figure 7. It is noted that the cumulative number of corrected faults is not always coincident with that of software failures or restoration actions. The sample state transition diagram of $X(t)$ is illustrated in Figure 8.

B. Software availability measures

We can obtain the state occupancy probabilities that the system is in state W_n and R_n at time point t as

$$P_{W_n}(t) \equiv \Pr\{X(t) = W_n\} = \frac{g_{n+1}(t)}{a\lambda_n} + \frac{g'_{n+1}(t)}{a\lambda_n\mu_n} \quad (n = 0,1,2,\cdots), \tag{26}$$

$$P_{R_n}(t) \equiv \Pr\{X(t) = R_n\} = \frac{g_{n+1}(t)}{a\mu_n}, \tag{27}$$

respectively, where $g_n(t)$ is the probability density function of random variable S_n, which denotes the first passage time to state W_n, and $g'_n(t) \equiv dg_n(t)/dt$. $g_n(t)$ and $g'_n(t)$ can be given analytically.

The following equation holds for arbitrary time t:

$$\sum_{n=0}^{\infty}[P_{W_n}(t) + P_{R_n}(t)] = 1. \tag{28}$$

The *instantaneous availability* is defined as

$$A(t) \equiv \sum_{n=0}^{\infty} P_{W_n}(t), \tag{29}$$

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

$$A_{av}(t) \equiv \frac{1}{t}\int_0^t A(x)dx, \tag{30}$$

which represents the ratio of system's operating time to the time interval . Using Eqs. (26) and (27), we can express Eqs. (29) and (30) as

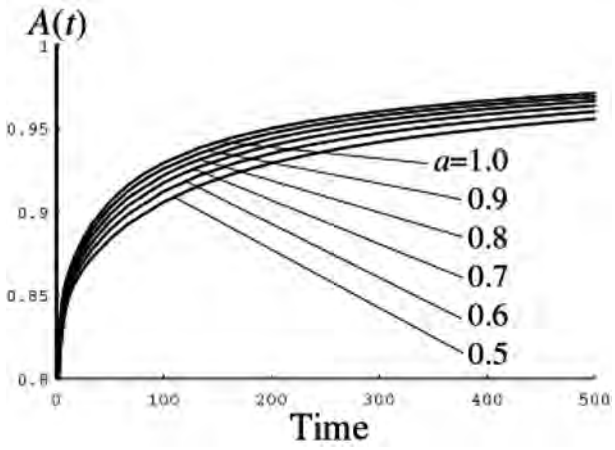

Fig. 9 Dependence of perfect debugging rate a on $A(t)$.

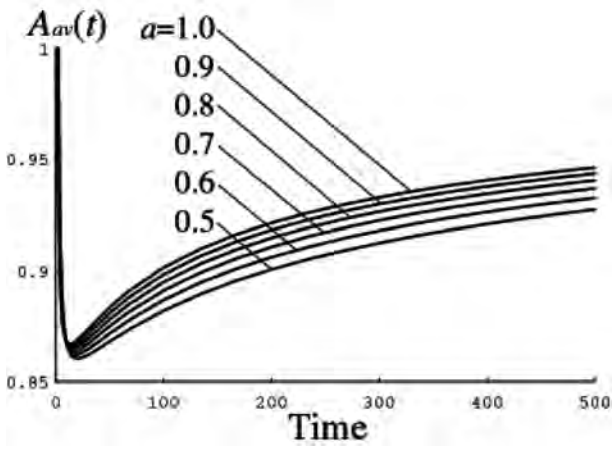

Fig. 10 Dependence of perfect debugging rate a on $A_{av}(t)$.

$$A(t)=\sum_{n=0}^{\infty}\left[\frac{g_{n+1}(t)}{a\lambda_n}+\frac{g'_{n+1}(t)}{a\lambda_n\mu_n}\right]=1-\sum_{n=0}^{\infty}\frac{g_{n+1}(t)}{a\mu_n}, \quad (31)$$

$$A_{av}(t)=\frac{1}{t}\sum_{n=0}^{\infty}\left[\frac{G_{n+1}(t)}{a\lambda_n}+\frac{g_{n+1}(t)}{a\lambda_n\mu_n}\right]=1-\frac{1}{t}\sum_{n=0}^{\infty}\frac{G_{n+1}(t)}{a\mu_n}, \quad (32)$$

respectively, where $G_n(t)$ is the distribution function of S_n.

Figures 9 and 10 show numerical illustrations of $A(t)$ and $A_{av}(t)$ in Eqs. (31) and (32), respectively.

REFERENCES

[1] S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*, Springer International Publishing, Switzerland, 2016.
[2] S. Yamada, *Software Reliability Modeling: Fundamentals and Applications*, Springer-Verlag, Tokyo/Heidelberg, 2014.
[3] S. Yamada, *Elements of Software Reliability: Modeling Approach* (in Japanese), Kyoritsu Publishing, Tokyo, 2011.
[4] S. Yamada, *Fundamentals and Applications of Software Engineering* (in Japanese), Surikogaku-sha, Tokyo, 2013
[5] S. Yamada and T. Fukushima, *Quality-Oriented Software Management* (in Japanese), Morikita Publishing, Tokyo, 2007.
[6] S. Yamada, *Software Reliability Modeling* (in Japanese), JUSE Press, Tokyo, 1994.
[7] S. Yamada and M. Takahashi, *Introduction to Software Management Models* (in Japanese), Kyoritsu Publishing, Tokyo, 1993.

APPENDIX

Professor Shigeru Yamada's Research Effort for Software Reliability Modeling

1975: BSE degree in Industrial Engineering received from Hiroshima University
1977: MS degree in Industrial Engineering received from Hiroshima University
1977–1980: Denso Co. (Quality Assurance Division)
1983: Ph.D. degree in System Engineering received from Hiroshima University
1983—1988: Assistant Professor/Senior Lecturer at Okayama University of Science

・November 1985: "Method of Software Reliability Assessment"*, Soft Research Center, Tokyo
* extended version from Ph.D. Thesis: "A Study on Software Reliability Growth Models Based on Nonhomogeneous Poisson Process and Its Statistical Inferences"

1988—1993: Associate Professor at the Department of Engineering, Hiroshima University

・May 1989: "Software Reliability Assessment Technologies, HBJ Publishing, Tokyo
・May 1990: "Software Reliability: Theory and Practical Application", Soft Research Center, Tokyo (Co-authored with Dr. H. Ohtera)
・May 1992: Best Author Award from Information Processing Society of Japan*
* S. Yamada, "Principles and Current Views in Software Quality Assessment: Quantitative Software Quality Assessment Method Based on Software Reliability Growth Models," IPSJ Magazine, Vol. 32, No. 11, pp. 1189–1202, 1991.
・March 1993: Telecom System Technology Award from Telecommunications Advancement Foundation*
* Shigeru Yamada: "Software Quality/Reliability Measurement and Assessment: Sofware Reliability Growth Models and Data Analysis," Journal of Information Processing, Vol. 14, No. 3, pp. 254–266, 1991.
・March 1993: "Introduction to Software Management Model," Kyoritsu Publishing, Tokyo (Co-authored with Dr. M. Takahashi)

©Shigeru Yamada

Professor Shigeru Yamada's Research Effort for Software Reliability Modeling

1993-present : Professor at the Dept. of Social Management Engineering, Graduate School of Engineering, Tottori University

・November 1994: "Software Reliability Model-Fundamentals and Applications-," JUSE Publishing, Tokyo

・November1998 : "Statistical Quality Control for TQM," Corona Publishing, Tokyo (Co-authored with Dr. M. Kimura and Dr. M. Takahashi)

・February 1999 - present : Member of Deming Prize Committee

・May 1999 : Takagi Prize 1998 (Best Paper Award) from Reliability Engineering Association of Japan*
* "Performance Evaluation Modeling for Fault-Tolerant Software Systems with Processing Time Limit," Journal of Reliability Engineering Society of Japan, Vol. 20, No1.7, pp. 422-432, 1998. (Co-authored with Ms. M. Yamamoto and Dr. M. Kimura)

・December 2003 : International Leadership Award in Reliability Engg. Research, (ICQRIT and SREQOM), India.
・March 2004 : Research Achievement Award 2003 from President of Tottori University

・November 2004 : "Software Reliability: Model, Tool, Management," Society of Project Management, Narashino (Co-authored with Dr. T. Fujiwara)

・March2006 : Best Paper Award from Society of Project Management*
* "The Proposal and the Trial for Software Quality Management System in the Global Era" Journal of the Society of Project Management, Vol. 7, No. 2, 21-26, 2001. (Co-authored with Dr. K. Yasuda, A. Ooishi, and N. Yoshida)

・March 2007 : "Quality-Oriented Software Management~Project Management for Highly Quality Software Development-," Morikita Publishing, Tokyo (Co-authored with Dr. T. Fukushima)

・March 2007 : Fellowship Recognition from Operations Research Society of Japan

・August 2007 : Leadership Award in Appreciation for Outstanding Contribution to ISSAT International Conference on Reliability and Quality in Design from ISSAT, U.S.A.

・December 2009 : International Leadership and Pioneering Research Award in Software Reliability Engineering, SREQOM, India

・March 2010 : Research Achievement Award on "Software Reliability Engineering" from President of Tottori University

・November 2010 : Exceptional International Leadership and Contribution Award in Software Reliability, ICRITO' 2010, India

・August 2011 : "Elements of Software Reliability - Modeling Approach -," Kyoritsu Publishing, Tokyo

・February 2012 : "Project Management," Kyoritsu Publishing, Tokyo (Co-authored with Dr. K. Esaki, Mr. H. Takane, and Dr. M. Takahasi)

・June 2012 : 2011 Best Paper Award from IEEE Reliability Society Japan Chapter*
* "Codesign-Oriented Performability Modeling for Hardware-Software Systems," IEEE Transactions on Reliability, Vol. 60, No. 1, pp.171-179, 2011. (Co-authored with Dr. K. Tokuno)

・July 2013 : "Fundamentals and Applications for Software Engineering," Surikogaku-sha, Tokyo (Co-authored with Dr. Y. Tamura)

・Marh 2014 : "Software Reliability Modeling: Fundamentals and Applications", Springer-Verlag (Springer Briefs in Statistics), Tokyo/Heidelberg

・May 2014 : The Korean Reliability Society' s Certificate of Appointment "Honorary Canon", KORAS, Korea

・August 2014 : Leadership Award for Outstanding Contributions to ISSAT International Conference on Reliability and Quality in Design from ISSAT, U.S.A.

・November 2014 : Project Management Service Award from Society of Project Management

・February 2015 : The Title of "Honorary Professor" Recognition from Amity University, India

・April 2016 : "OSS Reliability Measurement and Assessment", Springer International Publishing (Springer Series in Reliability Engineering), Switzerland (Co-authored with Dr. Y. Tamura)

・March 2017 : Contribution Award for Promoting OR from Operations Research Society of Japan

・August 2017 : Research Award from ISSAT (International Society of Science and Applied Technologies), Piscataway, New Jersey, U.S.A.

(Academic Publication Effort: Authored Books:13, Book Chapters: 62, Refereed Journal Papers: 291, Refereed Conference/Symposium Papers: 342, Survey Papers: 25, Keynote Address: 11, Invited Talk: 46 since March 1976 up to 2017)

eru Yamada