Markovian Analysis for Software Reliability/Availability Measurement Considering Continuous Use

Takashi Karaki, Koichi Tokuno, and Shigeru Yamada

Department of Social Systems Engineering
Faculty of Engineering, Tottori University
4-101, Koyama Minami, Tottori-shi 680-8552, Japan
E-mail: {96t7018, toku, yamada}@sse.tottori-u.ac.jp

Abstract

Most of software availability models so far have paid attention to whether or not the software system is available only at a given time point. This paper discusses software availability measurement considering continuous use. In particular, the following two measures are derived: the interval software reliability defined as the probability that the system is continuously available for a given time period, and the conditional mean available time defined as the mean operable time measured from a time point at which the system is operating. The stochastic behavior of the system alternating between up and down states is described by a Markov process. Then, the software reliability growth process and the upward tendency of difficulty in debugging are also incorporated in the model. Finally, we show numerical examples of software availability analysis.

1 Introduction

To assess the quality and performance of a software system quantitatively in the present-day society which the life and computer system stick fast becomes important. A mathematical software reliability model is often called a software reliability growth model (SRGM)[1,2,3]; this describes a stochastic behavior of a software fault-detection process or a software failure-occurrence phenomenon during the testing phase of software development process and the operation phase.

Recently, performance and quality of the software systems begin to be evaluated from customer’s viewpoints. However, most of SRGMs so far provide quantitative software reliability measures for developers such as the expected residual fault content and the mean time between software failures. For example, customers take interests in the information on possible utilization, not the number of faults remaining in the system. Software availability is one of the customer-oriented attributes. This is defined as the attribute that the software systems are in available states when the customers want to use them, according to the specification, under the specified environment. Currently, research on stochastic software availability modeling has been carried out[4,5].

In this paper, we discuss the stochastic measures for continuous use. Existing software availability models often provide quantitative measures such as the instantaneous and the average software availabilities. These measures pay attention to whether or not the system is available only at a specified time point. In particular, the following two measures are derived: the interval software reliability defined as the probability that the system is continuously available for a given time period, and the conditional mean available time defined as the mean operable time measured from a time point at which the system is operating.

In software availability modeling, we also describe the imperfect debugging environment where the debugging activities corresponding to software failure-occurrences are not always performed for certain. In particular, we generalize the availability model from the viewpoint of the perfect debugging rate; this is the probability that a fault is corrected perfectly in debugging. Here the perfect debugging rate is assumed to decrease with the increasing number of corrected faults. This reflects the actual dynamic environment where the difficulty of fault correction increases as the debugging progresses[6]. The stochastic behavior of the system alternating between up and down states is described by a Markov process[7]. Finally, numerical examples of software availability analysis are illustrated.

2 Model Description

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 cannot operate until the restoration action is complete.
A2. The restoration action implies the debugging activity; this is performed perfectly with probability \( a_n \) \((0 < a_n \leq 1)\) and imperfectly with probability \( b_n \) \((1 - a_n)\). We call \( a_n \) the perfect debugging rate. One fault is corrected and removed from the software system when a debugging activity is perfect.

A3. When \( n \) faults have been corrected, the time to the next software failure-occurrence, \( T_n \), and the restoration time, \( U_n \), follow exponential distributions with means \( 1/\lambda_n \) and \( 1/\mu_n \), respectively. \( \lambda_n \) and \( \mu_n \) are decreasing functions of \( n \).

A4. The probability that two or more software failures occur simultaneously is negligible.

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

From assumption A2, if the restoration action is complete in \( \{ X(t) = R_n \} \), then

\[
X(t) = \begin{cases} \ W_n & (\text{with probability } b_n) \\ W_{n+1} & (\text{with probability } a_n). \end{cases}
\]  

(1)

In general, the faults detected later tend to have higher complexity[6]. That is, the certainty of debugging becomes smaller with increase in the number of corrected faults. For instance, we may describe the perfect debugging rate \( a_n \) as

\[
a_n = vw^n + \alpha \\
(n = 0, 1, 2, \ldots; 0 < v, w, \alpha, v + \alpha \leq 1),
\]

(2)

where \( v + \alpha, w, \) and \( \alpha \) mean the initial perfect debugging rate, the decreasing ratio of the perfect debugging rate, and the stationary perfect debugging rate, respectively. Equation (2) reflects that the debugging activities become more difficult with the progress of debugging. A sample behavior of \( a_n \) is illustrated in Fig. 1.

We use the model derived by Moranda[8] to describe the software failure-occurrence phenomenon, i.e., when \( n \) faults have been corrected, the hazard rate \( \lambda_n \) is given by

\[
\lambda_n = Dk^n \\
(n = 0, 1, 2, \ldots; D > 0, 0 < k < 1),
\]

(3)

where \( D \) and \( k \) are the initial hazard rate and the decreasing ratio of the hazard rate, respectively. The expression of (3) comes from the viewpoint 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) describes a software failure-occurrence phenomenon where earlier perfect debuggings have larger impact on software reliability growth than later ones[1,4].

Subsequently, we describe the time-dependent behavior of the restoration action. The restoration action includes not only the data recovery and the program reload but also the debugging activities for manifested faults. From the similar viewpoint of the fault complexity mentioned in the description of the perfect debugging rate, we consider the operational environment where the later restoration actions tend to be longer. Then, the following is feasible for the restoration rate \( \mu_n \):

\[
\mu_n = Ev^n \\
(n = 0, 1, 2, \ldots; E > 0, 0 < r \leq 1),
\]

(4)

where \( E \) and \( r \) are the initial restoration rate and the decreasing ratio of the restoration rate, respectively. In the case of \( r = 1 \), i.e., \( \mu_n = E \) means that the restoration time of each fault is random[9].

Let \( Q_{A,B}(t) (A, B \in \{ W_n, R_n; n = 0, 1, 2, \ldots \} \) 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 \( \tau \). The expressions for \( Q_{A,B}(t) \)'s are given as follows:

\[
Q_{W_n, W_{n+1}}(\tau) = a_n(1 - e^{-\lambda_n \tau}), \\
Q_{R_n, W_{n+1}}(\tau) = b_n(1 - e^{-\lambda_n \tau}), \\
Q_{R_n, R_{n+1}}(\tau) = \mu_n(1 - e^{-\lambda_n \tau}).
\]

(5)

(6)

(7)

Figure 2 illustrates the sample state transition diagram of \( X(t) \).
Fig. 2: A state transition diagram of $X(t)$.

3 Derivation of Software Availability Measures

3.1 Distribution of the First Passage Time to Up States

Let $S_{i,n}$ ($i < n$) and $G_{i,n}(t)$ be the random variable representing the transition time between states $W_i$ and $W_n$, and the distribution function of $S_{i,n}$, respectively, and we denote $S_n \equiv S_{0,n}$. Then, we obtain the following renewal equation with respect to $G_{i,n}(t)$:

$$G_{i,n}(t) = Q_{W_i} * Q_{W_n} * G_{i+1,n}(t) + Q_{W_i} * Q_{W_n} * G_{i,n}(t)$$

$$i = 0, 1, 2, \ldots, n - 1,$$  \hspace{1cm} (8)

where $*$ denotes a Stieltjes convolution and $G_{n,n}(t) = 1(t)$ (step function, $n = 0, 1, 2, \ldots$).

Solving (8) recursively by applying the Laplace-Stieltjes transforms, we can obtain the distribution function of the random variable $S_n$ representing the time to be spent in correcting $n$ faults as

$$G_n(t) = \Pr[S_n \leq t]$$

$$= 1 - \sum_{i=0}^{n-1} (A_{n,i}^1 e^{-x_i t} + A_{n,i}^2 e^{-y_i t})$$

$$=(n = 1, 2, \ldots; \quad G_0(t) \equiv 1),$$  \hspace{1cm} (9)

where $x_i$ and $y_i$ satisfy the following relations:

$$x_i + y_i = \lambda_i + \mu_i$$  \hspace{1cm} (10)

$$x_i y_i = a_i \lambda_i \mu_i$$  \hspace{1cm} (11)

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

$$A_{n,i}^1 = \prod_{j=0}^{n-1} a_j \lambda_j \mu_j$$

$$x_i \prod_{j=0}^{n-1} (x_j - x_i) \prod_{j=0}^{n-1} (y_j - y_i)$$

$$(i = 0, 1, \ldots, n - 1),$$  \hspace{1cm} (12)

respectively. It is noted that

$$\sum_{i=0}^{n-1} (A_{n,i}^1 + A_{n,i}^2) = 1 \quad (n \geq 1).$$  \hspace{1cm} (13)

Furthermore, the expectation and the variance of $S_n$ are given by

$$\mathbb{E}[S_n] = \sum_{i=0}^{n-1} \left( \frac{1}{x_i} + \frac{1}{y_i} \right),$$  \hspace{1cm} (15)

$$\text{Var}[S_n] = \sum_{i=0}^{n-1} \left( \frac{1}{x_i^2} + \frac{1}{y_i^2} \right),$$  \hspace{1cm} (16)

respectively.

3.2 State Occupancy Probability

We denote $P_{A,B} \equiv \Pr\{X(t) = B|X(t) = A\}$ ($A, B \in (W_i, R_i)$), and $P_{W_n}(t) \equiv P_{W_n,W_n}(t)$ and $P_{R_n}(t) \equiv P_{W_n,R_n}(t)$, respectively.

Then we obtain the following renewal equations of $P_{W_n}(t)$ and $P_{R_n}(t)$:

$$P_{W_n}(t) = G_n * P_{W_n,W_n}(t),$$  \hspace{1cm} (17)

$$P_{W_n,W_n}(t) = e^{-\lambda_n t} + Q_{W_n} * Q_{W_n,W_n} * P_{W_n,W_n}(t),$$  \hspace{1cm} (18)

$$P_{R_n}(t) = G_n * P_{R_n,R_n}(t),$$  \hspace{1cm} (19)

$$P_{R_n,R_n}(t) = e^{-\mu_n t} + Q_{R_n} * Q_{R_n,R_n} * P_{R_n,R_n}(t).$$  \hspace{1cm} (20)

Solving the above equations, we can obtain the state occupancy probabilities that the system is in states $W_n$ and $R_n$ at time point $t$ as

$$P_{W_n}(t) = \Pr\{X(t) = W_n\}$$

$$= \frac{1}{a_n \lambda_n} g_n(t) + \frac{1}{a_n \lambda_n} g'_n(t),$$  \hspace{1cm} (21)

$$P_{R_n}(t) = \Pr\{X(t) = R_n\}$$

$$= \frac{1}{a_n \mu_n} g_n(t),$$  \hspace{1cm} (22)

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

- 151 -
3.3 Software Availability

The following equation holds for arbitrary time point \( t \):

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

(23)

Furthermore, we introduce the following binary random variable \( I(t) \) representing the state of the system at time point \( t \):

\[
I(t) = \begin{cases} 
1 & \text{(up state)} \\
0 & \text{(down state)}.
\end{cases}
\]

(24)

Then the instantaneous software availability is defined as

\[
A(t) \equiv \Pr\{I(t) = 1\} = \sum_{n=0}^{\infty} P_{W_n}(t),
\]

(25)

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

\[
A_{av}(t) \equiv \frac{1}{t} \int_{0}^{t} A(x)dx,
\]

(26)

which represents the average proportion of system's operating time to time-interval \((0,t]\). Using (21)-(23), we can express (25) and (26) as

\[
A(t) = \sum_{n=0}^{\infty} \frac{1}{a_n \lambda_n} g_{n+1}(t) + \frac{1}{a_n \lambda_n \mu_n} g_{n+1}(t)
\]

\[= 1 - \sum_{n=0}^{\infty} \frac{1}{a_n \lambda_n} g_{n+1}(t), \]

(27)

\[
A_{av}(t) = \frac{1}{t} \sum_{n=0}^{\infty} \frac{1}{a_n \lambda_n} G_{n+1}(t) + \frac{1}{a_n \lambda_n \mu_n} G_{n+1}(t)
\]

\[= 1 - \frac{1}{t} \sum_{n=0}^{\infty} \frac{1}{a_n \lambda_n} G_{n+1}(t), \]

(28)

respectively.

3.4 Interval Software Reliability and Conditional Mean Available Time

Let \( Z_t \) be the random variable representing the software failure-occurrence time measured from arbitrary time point \( t \) at which the system is operating. Then, the interval software reliability is defined as

\[
R_I(t,x) = \Pr\{I(t) = 1, Z_t > x\} = \sum_{n=0}^{\infty} \Pr\{X(t) = W_n, T_n > x\},
\]

(29)

which represents the probability that system is operable at time point \( t \) and will continue to be available for time-interval \((t,t+x]\). Furthermore, the conditional mean available time is defined as

\[
MAT(t) = E[Z_t | I(t) = 1] = \int_{0}^{\infty} R_I(t,x)dx \Pr\{I(t) = 1\},
\]

(30)

which represents the expected available time on the condition that the system is operating at time point \( t \) (see Fig.3). Using the results obtained in Sects. 3.2 and 3.3, we can express (29) and (30) as

\[
R_I(t,x) = \sum_{n=0}^{\infty} P_{W_n}(t)e^{-\lambda_n x},
\]

(31)

\[
MAT(t) = \frac{\sum_{n=0}^{\infty} P_{W_n}(t) / \lambda_n}{\sum_{n=0}^{\infty} P_{W_n}(t)},
\]

(32)

respectively.

![Fig. 3: A sample behavior of \( I(t) \) and random variable \( Z_t \).](image)

4 Numerical Examples

We present several numerical examples of software availability measures derived above.

Figure 4 shows the interval software reliability, \( R_I(t,x) \) in (31) for various values of time interval \( x \). This figure indicates that it is difficult to use the system continuously in the initial operation and that the probability that the system is continuously available increases with the lapse of time point \( t \).
Figure 5 shows the dependence of the decreasing ratio of the perfect debugging rate, \( w \) on \( R(t, x) \). This figure tells us that higher debugging ability provides the more continuously-available system with time.

Furthermore, Fig. 6 shows the dependence of \( w \) on the conditional mean available time, \( MAT(t) \) in (32). This figure displays that we can lengthen the continuously-available time, given that the system is operating at that time.

\[
R(x, t)
\]

\[
R(t, 30)
\]

Fig. 5: Dependence of the decreasing ratio of the perfect debugging rate \( w \) on \( R(t, x) \) \((x = 30, v = 0.7, \alpha = 0.2, D = 0.1, k = 0.8, E = 0.5, r = 0.9)\).

Fig. 6: Dependence of the decreasing ratio of the perfect debugging rate \( w \) on \( MAT(t) \) \((v = 0.7, \alpha = 0.2, D = 0.1, k = 0.8, E = 0.5, r = 0.9)\).

5 Concluding Remarks

In this paper, the method of software availability measurement for continuous use has been discussed on the basis of the existing Markovian software availability model. The model has considered that the perfect debugging activities become more difficult as the fault removal work progresses. From the model, the following two quantitative measures have been derived: the interval software reliability and the conditional mean available time. Numerical examples of these measures have also been presented.

Acknowledgments

This work was supported in part by a Research Grant from the TAF of Japan and a Grant-in-Aid for Scientific Research from the Ministry of Education, Science, Sports, and Culture of Japan under Grant. No. 12680442.

References


