Performability Analysis for Software-Intensive System Considering Variety of Tasks and Operation-Oriented Restoration

Koichi Tokuno and Shigeru Yamada
Graduate School of Engineering, Tottori University, Tottori-shi, Japan
(Received February 2009, accepted December 2009)

Abstract: In this paper, we discuss software performability evaluation considering the real-time property. The time-dependent behavior of the system alternating between up and down state is described by the Markov process. Then we incorporate the operation-oriented restoration scenario into the model, i.e., we consider the following two types of restoration: one is the restoration with debugging and the other is without debugging. Assuming that the software system can process the multiple tasks simultaneously, we also consider the variety of tasks in terms of the task arrival process and the processing time limit. We describe the arrival process of the tasks follows a nonhomogeneous Poisson process and treat the processing time limit as a random variable. We analyze the distribution of the number of tasks whose processes can be completed within the processing time limit with the infinite server queueing model. From the model, we derive several software performability measures considering the real-time property. Finally, we illustrate several numerical examples of the measures to investigate the impact of the variety of tasks and the operational restoration on the system performability evaluation.

Keywords: Infinite-server queueing model, operation-oriented restoration, performability, real-time property, software reliability growth, variety of tasks.

1. Introduction

Today service reliability engineering have a growing attention [21, 22]; this aims at the establishment of the evaluation methods for the quality of service created by the use of the artificial industrial products as well as the inherent quality of the products. Considering the software systems are just the industrial products to provide the services for the end users, especially in computer network systems, it is meaningful to discuss the performability evaluation methods for software systems oriented to the service reliability engineering. Furthermore, the body of knowledge areas of service computing has recently been summarized [24, 25].

The studies on performability evaluation methods for computing systems have much been discussed. For example, Beaudry [1], Meyer [7], Nakamura and Osaki [11], and Sols [18] have discussed the hardware-configuration-conscious performability evaluation. On the other hand, most of studies on software-oriented reliability evaluation have treated only the inherent and internal reliability characteristics such as the residual fault content, the mean time between software failures, and the software reliability function. These measures are mainly used for the project management in the software development process [6, 15, 23].

However, software-conscious approaches extended to performability evaluation have also increased recently. Okamura et al. [12] have derived several dependability measures such
as the steady-state availability, the probability of transaction loss, and the upper bound of mean response time on transaction for a transaction-based system with preventive maintenance named software rejuvenation. Eto and Dohi [2] have treated the software system consisting of one operating system and multiple applications and derived the optimal preventive maintenance schedule maximizing the steady-state performability. Schwefel and Antonios [17] have constructed the analytical performability model for the cluster system, which is one of the multi-server and distributed computing system architectures, with the single-server Markov modulated Poisson process. They have derived the mean queue length and the tail probability of the queue-length distribution as performability metrics. All the above studies have paid attention to the phenomenon of software aging [4, 14].

The above software-conscious approaches are discussed on the basis of performability measures in steady states and assume that the probabilistic or stochastic characteristic in system failure does not change even though the system is debugged or refreshed, i.e., the system returns to the initial condition in terms of the failure characteristic, neither better nor worse states. As to this point, the analytical framework in the above studies is basically similar to the hardware-conscious approach even though the previous authors above-mentioned say that their works are software-oriented. Traditional stochastic software reliability modeling often considers the dynamic reliability performance growth process. Musa [10] says that the above mention is one of main differences from the modeling for the hardware system.

In this paper, we discuss the operation-oriented performability evaluation for the software system; this is the different approach from [2, 12, 17]. In particular, we consider the real-time property; this is defined as the attribute that the system can complete the task within the stipulated response time limit [5, 9]. In discussion on the software performability evaluation, we have to reflect the operational environment and the external factors of the system into the modeling as much as possible. As to the operational restoration, debugging activities in system down are not always performed since the cause of the system down may be the software aging mentioned above, which is a sort of restoration procedure without debugging, or protracting an inoperable time may much affect the customers. Here we consider the two kinds of restoration; one includes debugging and the other does not include debugging. This is a different policy from the testing phase that the debugging is always performed and the improvement of software reliability performance is attempted whenever a software failure occurs. Furthermore, in terms of the external factors of the system, we treat the case where the task arrival process follows a nonhomogeneous Poisson process (NHPP) and consider that each of arriving tasks has a different processing time limit, which is treated as a random variable. The previous related works such as [3, 20] often assume that the task arrival process follows a homogeneous Poisson process and that any task has a same and constant processing time limit. The time-dependent behavior of the system alternating between up and down states is described by the Markov process. The stochastic behavior of the number of tasks whose processes can be complete within the processing time limit is modeled with the infinite-server queueing model [16].

The organization of the rest of the paper is shown as follows: Section 2 states the operation-oriented software availability model proposed by Tokuno and Yamada [19]. Section 3 defines the operating regulation of the system and analyzes the distribution of the number of tasks whose processes are complete within the processing time limit up to a given time point. Section 4 derives several software performability measures from the model. The measures are given as the functions of time and the number of debuggings.
Section 5 illustrates the numerical examples of the measures and examines the software performability analysis. Finally, Section 6 summarizes the conclusion of the paper.

2. Operation-Oriented Software Availability Model

2.1. Model Description

The following assumptions are made for operational software availability modeling:

AI-1. The software system breaks down and starts to be restored as soon as a software failure occurs, and the system cannot operate until the restoration action completes.

AI-2. When a software failure occurs, the restoration action with the debugging activity is performed with probability  \( p (0 < p < 1) \), on the other hand, without the debugging activity is performed with probability  \( q (= 1 - p) \).

AI-3. The debugging activity is perfect with the perfect debugging probability  \( a (0 < a < 1) \), on the other hand, imperfect with the probability  \( b (= 1 - a) \). If the debugging activity is perfect, one fault is corrected and removed from the system.

AI-4. When \( n \) faults have been corrected, the time to the next software failure-occurrence, \( U_n \), and the restoration time with the debugging activity, \( L_n \), follow the exponential distributions with means \( 1/\lambda_n \) and \( 1/\mu_n \), respectively. \( \lambda_n \) and \( \mu_n \) are non-increasing functions of \( n \).

AI-5. The restoration time without the debugging activity, \( L_n^2 \), follows the exponential distribution with mean \( 1/\eta \).

Let \( \{X(t), t \geq 0\} \) be the stochastic process representing the state of the software system at the time point \( t \) and its state space is defined as follows:

\[
W = (W_n : n = 0, 1, 2, \ldots) : \text{the system is operating and available,}
\]

\[
R^1 = (R^1_n : n = 0, 1, 2, \ldots) : \text{the system is inoperable and in process of restoration with the debugging activity,}
\]

\[
R^2 = (R^2_n : n = 0, 1, 2, \ldots) : \text{the system is inoperable and in process of restoration without the debugging activity,}
\]

where \( n \) denotes the cumulative number of corrected faults.

We refer to the treatment of the probabilistic characteristic of the restoration time in state \( R^2 \). There are various causes of system down and corresponding restoration scenarios in the operation phase. In the case where the cause of the system down is software aging, the restoration procedure called software rejuvenation may be performed [4, 14]. As another case, if we judge to prioritize to shorten inoperable time over to adapt the system to operational environment, the restoration including only data recovery and program reload, in other words, the restoration without debugging is often performed. In this case, the system returns to the state before software failure-occurrence, i.e., software reliability growth never occurs. We assume that it is probabilistic whether or not debugging activity is performed when the system is down and that the restoration without debugging completes randomly throughout the operation phase.

Figure 1 illustrates the state transition diagram of \( X(t) \). Let \( Q_{A,B}(t) \) \( (A, B \in \{W, R^1, R^2\}) \) denote the one-step transition probability that, after making a transition into state \( A \), the process \( X(t) \) next makes a transition into state \( B \) in an amount of time less than or equal to \( t \) [13]. From Figure 1, we have the following expressions of \( Q_{A,B}(t) \)'s:
2.2. Traditional Software Availability Measures

2.2.1. Distribution of Transition Time between State W

Let $S_{i,n} (i,n = 0,1,2,...; i \leq n; S_{i,n} = 0)$ be the random variable representing the transition time of $X(t)$ from state $W_i$ to state $W_n$ and $G_{i,n}(t)$ be a distribution function of $S_{i,n}$, respectively. Then, we obtain the following renewal equation of $G_{i,n}(t)$:

$$Q_{W_i, R_i} (t) = \Pr\{U_n \leq t, X(U_n) = R_i^1 | X(0) = W_n\} = p(1 - e^{-\lambda t}),$$

$$Q_{W_i, R_i} (t) = \Pr\{U_n \leq t, X(U_n) = R_i^2 | X(0) = W_n\} = q(1 - e^{-\lambda t}),$$

$$Q_{R_i^1, W_{i+1}} (t) = \Pr\{L_n^1 \leq t, X(L_n^1) = W_{i+1} | X(0) = R_i^1\} = a(1 - e^{-\mu t}),$$

$$Q_{R_i^2, W_n} (t) = \Pr\{L_n^2 \leq t | X(0) = R_i^2\} = b(1 - e^{-\eta t}),$$

$$Q_{R_i^2, W_n} (t) = \Pr\{L_n^2 \leq t | X(0) = R_i^2\} = 1 - e^{-\eta t}.$$
$$G_{i,n}(t) = Q_{W_i,R_i}^r * Q_{R_i,W_i}^r * G_{i+1,n}(t) + Q_{W_i,R_i}^r * Q_{R_i,W_i}^r * G_{i,n}(t)$$
$$+ Q_{W_i,R_i}^r * Q_{R_i,W_i}^r * G_{i,n}(t) \quad (i = 0, 1, 2, ..., n-1),$$

where * denotes a Stieltjes convolution and $G_{n,n}(t) = 1(t)$ (unit function) $(n = 0, 1, 2,...)$. We can solve Equation (6) with respect to $G_{i,n}(t)$ by applying the Laplace-Stieltjes transform [13]. The solution of Equation (6) is obtained as

$$G_{i,n}(t) = \Pr \{ S_{i,n} \leq t \}$$
$$= 1 - \sum_{m=1}^{n-1} \left[ A_{i,n}^1(m)e^{-\lambda_m t} + A_{i,n}^2(m)e^{-\mu_m t} + A_{i,n}^3(m)e^{-\eta_m t} \right]$$
$$(t \geq 0; i, n = 0, 1, 2,...; i \leq n)$$

$$A_{i,n}^1(m) = \prod_{j=i}^{n-1} \left( \frac{\prod_{j=m}^{n-1} (d_j^1 - d_m^2) \prod_{j=m}^{n-1} (d_j^2 - d_m^3)(d_j^3 - d_m^1)}{d_m^1 \prod_{j=i}^{n-1} (d_j^1 - d_m^2) \prod_{j=m}^{n-1} (d_j^2 - d_m^3)(d_j^3 - d_m^1)} \right)$$

$$A_{i,n}^2(m) = \prod_{j=i}^{n-1} \left( \frac{\prod_{j=m}^{n-1} (d_j^2 - d_m^2) \prod_{j=m}^{n-1} (d_j^3 - d_m^3)(d_j^1 - d_m^2)}{d_m^2 \prod_{j=i}^{n-1} (d_j^2 - d_m^2) \prod_{j=m}^{n-1} (d_j^3 - d_m^3)(d_j^1 - d_m^2)} \right)$$

$$A_{i,n}^3(m) = \prod_{j=i}^{n-1} \left( \frac{\prod_{j=m}^{n-1} (d_j^3 - d_m^3) \prod_{j=m}^{n-1} (d_j^1 - d_m^2)(d_j^2 - d_m^3)}{d_m^3 \prod_{j=i}^{n-1} (d_j^3 - d_m^3) \prod_{j=m}^{n-1} (d_j^1 - d_m^2)(d_j^2 - d_m^3)} \right)$$

$$(m = i, i + 1, i + 2, ..., n - 1)$$

where $-d_m^1, -d_m^2$ and $-d_m^3$ are the distinct roots of the following third order equation of $s$

$$s^3 + (\lambda_m + \mu_m + \eta)s^2 + [(1 - pb)\lambda_m \mu_m + \mu_m \eta + p\eta \lambda_m]s + p\lambda_m \mu_m \eta = 0.$$  \hspace{1cm} (8)

The form of Equation (7) is a hypoexponential distribution.

### 2.2.2. Operational State Occupancy Probability and Software Availability

Let $P_{A,B}(t) = \Pr \{ X(t) = B | X(0) = A \} (A, B) \in \{ W, R^1, R^2 \}$ be the state occupancy probability that the system is in state $B$ at the time point $t$ on the condition that the system was in state $A$ at time point $t = 0$. Then, we obtain the following renewal equation of $P_{W_i,W_i}(t)$:

$$P_{W_i,W_i}(t) = G_{i,n} * P_{W_i,W_i}(t),$$  \hspace{1cm} (9)

$$P_{W_i,W_i}(t) = e^{-\lambda_i t} + Q_{W_i,R_i}^a \cdot Q_{R_i,W_i}^a \cdot P_{W_i,W_i}(t) + Q_{W_i,R_i}^a \cdot Q_{R_i,W_i}^a \cdot P_{W_i,W_i}(t).$$  \hspace{1cm} (10)

Solving Equations (9) and (10), we obtain the operational state occupancy probability as
\[ P_{W_i, w_n}(t) = \Pr\{X(t) = W_i \mid X(0) = W_n\} = \frac{g_{i, n+1}(t)}{\mu_n} + \frac{g'_{i, n+1}(t)}{\mu_n}, \]

where \( g_{i, n}(t) = \frac{dG_{i, n}(t)}{dt} \) is the density function of \( S_{i, n} \) and \( g'_{i, n}(t) = \frac{d^2G_{i, n}(t)}{dt^2} \).

The instantaneous software availability and the average software availability are given by

\[ A(t, l) = \sum_{i=0}^{l} \binom{l}{i} b^{l-i} \sum_{n=i}^{\infty} \left[ \frac{g_{i, n+1}(t)}{\mu_n} + \frac{g'_{i, n+1}(t)}{\mu_n} \right], \]

\[ A_{av}(t, l) = \frac{1}{l} \sum_{i=0}^{l} \binom{l}{i} b^{l-i} \sum_{n=i}^{\infty} \left[ \frac{G_{i, n+1}(t)}{\mu_n} + \frac{g_{i, n+1}(t)}{\mu_n} \right], \]

respectively, where

\[ \binom{l}{i} = \frac{l!}{i!(l-i)!} \]

denotes the binomial coefficient. Equations (12) and (13) represent the probability that the software system is operable and available at the time point \( t \) and the expected proportion of system's operating time to the time interval \((0, t]\), given that the \( l\)-th debugging activity \((l = 0, 1, 2, \ldots)\) was complete at time point \( t = 0 \), respectively.

3. Model Description and Analysis for Task Processing

We make the following assumptions for system's task processing:

AI-1. The number of tasks the system can process simultaneously is sufficiently large.

AI-2. The process \( \{N(t), t \geq 0\} \) representing the number of tasks arriving at the system up to the time \( t \) follows the NHPP with the arrival rate \( \omega(t) \) and the mean value function \( \Omega(t) = \mathbb{E}[N(t)] = \int_0^t \omega(x)dx \).

AI-3. Each task has a processing time limit, \( T_r \), which follows a general distribution whose distribution function is denoted as \( F_r(t) = \Pr\{T_r \leq t\} \).

AI-4. The processing time of a task is distributed generally whose distribution function is denoted as \( F_r(t) = \Pr\{Y \leq t\} \). Each of processing times is independent.

AI-5. When the system causes a software failure in task processing or the processing times of tasks exceed their own processing time limits, the corresponding tasks are canceled.

Here we derive the distribution of the number of tasks whose processes are complete within the processing time limit. Figure 2 illustrates the configuration of the system's task processing we consider. Hereafter, we set the time origin \( t = 0 \) at the time point when the debugging activity is complete and \( t(= 0, 1, 2, \ldots) \) faults are corrected.

Let \( \{Z_i^1(t), t \geq 0\} \) be the stochastic process representing the cumulative number of tasks whose processes can be complete within the processing time limit out of the tasks arriving up to the time \( t \). By conditioning with \( \{N(t) = k\} \), we obtain the probability mass function of \( Z_i^1(t) \) as
\[
\Pr\{Z_i^1(t) = j\} = \sum_{k=0}^{\infty} \Pr\{Z_i^1(t) = j \mid N(t) = k\}e^{-\Omega(t)} \frac{[\Omega(t)]^k}{k!} \quad (j = 0, 1, 2, \ldots).
\] (14)

From Figure 2, the probabilities that the process of an arbitrary task is complete within the processing time limit is given by

\[
\beta_{W_n} = \Pr\{Y < U_n, Y < T_r \mid X(t) = W_n\}
= \int_0^\infty e^{-\lambda_n y} \overline{F}_r(y) dF_r(y),
\] (15)

respectively, where we denote \(\overline{F}(\cdot) \equiv 1 - F(\cdot)\). Furthermore, from the property of the NHPP, we should note that the arrival time of an arbitrary task out of ones arriving up to the time \(t\) is the random variable having the following probability density function:

\[
f(x) = \begin{cases} \frac{\omega(x)}{\Omega(t)} & (0 \leq x \leq t) \\ 0 & (x > t) \end{cases}
\] (16)

\[\]

Figure 2. Configuration of task processing.

Therefore, the probability that the process of an arbitrary task having arrived up to the time \(t\) is complete within the processing time limit is obtained as

\[
\gamma_i^1(t) = \left[\sum_{n=1}^{\infty} \Pr\{X(x) = W_n \mid X(0) = W_i\} \times \Pr\{Y < U_n, Y < T_r \mid X(x) = W_n\}\right] \cdot f(x) dx
= \frac{1}{\Omega(t)} \sum_{n=1}^{\infty} \beta_{W_n} \int_0^\infty \left[ \frac{g_{i, n+1}(t)}{pa\lambda_n} + \frac{g'_{i, n+1}(t)}{pa\lambda_n\mu_n} \right] \omega(x) dx,
\] (17)
from the infinite-server queueing theory [16]. Then from assumption AII-4,

\[
\Pr\{Z_i^1(t) = j | N(t) = k\} = \begin{cases} 
\binom{k}{j} [\gamma_i^1(t)]^j [1 - \gamma_i^1(t)]^{k-j} & (j = 0, 1, 2, \ldots, k) \\
0 & (j > k)
\end{cases}
\] (18)

Equation (18) means that, given that \{N(t) = k\}, the number of tasks whose processes can be complete within the processing time limit follows the binomial process with mean \(k\gamma_i^1(t)\). Accordingly, from Equation (14) the distribution of \(Z_i^1(t)\) is given by

\[
\Pr\{Z_i^1(t) = j\} = e^{-\Omega(t)} \frac{[\Omega(t)\gamma_i^1(t)]^j}{j!}.
\] (19)

Equation (19) means that \(\{Z_i^1(t), t \geq 0\}\) follows the NHPP with the mean value function \(\Omega(t)\gamma_i^1(t)\).

Letting \(\{Z_i^2(t), t \geq 0\}\) be the random variable representing the cumulative number of tasks canceled out of ones arriving up to the time \(t\), we can have a similar discussion on \(\{Z_i^2(t), t \geq 0\}\), i.e., the distribution of \(Z_i^2(t)\) is given by

\[
\Pr\{Z_i^2(t) = j\} = e^{-\Omega(t)\gamma_i^2(t)} \frac{[\Omega(t)\gamma_i^2(t)]^j}{j!},
\] (20)

Equation (20) means that \(\{Z_i^2(t), t \geq 0\}\) follows the NHPP with the mean value function \(\Omega(t)\gamma_i^2(t)\).

### 4. Derivation of Software Performability Measures

Based on the above analysis, we can obtain several measures for software performability evaluation considering the real-time property.

The expected number of tasks completable out of the tasks arriving up to the time is given by

\[
\Lambda_i^1(t) = E[Z_i^1(t)]
= \sum_{n=1}^{\infty} \beta_{W_n} \left[ \int \left( \frac{g_{i,n+1}(x)}{pa\lambda_n} + \frac{g'_{i,n+1}(x)}{pa\lambda_n\mu_n}\right) \omega(x) dx \right].
\] (21)

Furthermore, the instantaneous task completion ratio is obtained as

\[
\upsilon_i^1(t) = \frac{d\Lambda_i^1(t)}{d\tau} \int \omega(t)
= \sum_{n=1}^{\infty} \beta_{W_n} \left[ \frac{g_{i,n+1}(t)}{pa\lambda_n} + \frac{g'_{i,n+1}(t)}{pa\lambda_n\mu_n}\right].
\] (22)
which represents the ratio of the number of tasks completed within the processing time limit to one arriving at the system per unit time at the time point $t$. We should note that Equation (22) is no bearing on $\Omega(t)$, i.e., $\nu_i^2(t)$ is independent of the task arrival process. As to $\gamma_i^1(t)$ in Equation (17), we can give the following interpretations:

$$\gamma_i^1(t) = \frac{\mathbb{E}[Z_i^1(t)]}{\mathbb{E}[N(t)]}. \quad (23)$$

That is, $\gamma_i^1(t)$ is the cumulative task completion ratio up to the time $t$ which represents the expected proportion of the cumulative number of tasks completed within the processing time limit to one arriving at the system in the time interval $(0, t]$.

For the number of tasks canceled out of ones arriving up to the time $t$, we can have the similar discussion, i.e., the expected number of tasks canceled up to the time $t$, the instantaneous and the cumulative task incompletion ratios are given by

$$\Lambda_i^2(t) = \mathbb{E}[Z_i^2(t)]$$
$$\Lambda_i^2(t) = \Omega(t) - \sum_{n=i}^{\infty} \beta_{W_n} \int_0^t \left[ \frac{g_{i,n+1}(x)}{pa\lambda_n} + \frac{g'_{i,n+1}(x)}{pa\lambda_n\mu_n} \right] \omega(x)dx, \quad (24)$$

$$\nu_i^2(t) = \frac{d\Lambda_i^1(t)}{dt} \omega(t)$$
$$\nu_i^2(t) = 1 - \sum_{n=i}^{\infty} \beta_{W_n} \left[ \frac{g_{i,n+1}(t)}{pa\lambda_n} + \frac{g'_{i,n+1}(t)}{pa\lambda_n\mu_n} \right], \quad (25)$$

$$\gamma_i^2(t) = \frac{\mathbb{E}[Z_i^2(t)]}{\mathbb{E}[N(t)]}, \quad (26)$$

respectively.

We should note that it is too difficult to use Equations (21)-(26) practically since this model assumes the imperfect debugging environment and the initial condition $i (= 0, 1, 2, \ldots)$ appearing in the above equations, which represents the cumulative number of faults corrected at time point $t = 0$, cannot be observed immediately. However, the numbers of software failures or debugging activities can be easily observed. Furthermore, the cumulative number of faults corrected immediately after the completion of the $l$-th debugging activity, $C_l$, follows the binomial distribution whose probability mass function is given by

$$\text{Pr}\{C_l = i\} = \text{Pr}\{X(0) = W_i\}$$
$$= \binom{l}{i} a^i b^{l-i} \quad (i = 0, 1, 2, \ldots, l). \quad (27)$$

Accordingly, we can convert Equations (21)-(26) into the functions of the number of debuggings, $l$, i.e., we obtain

$$\Lambda^1(t, l) = \sum_{i=0}^{l} \binom{l}{i} a^i b^{l-i} \sum_{n=i}^{\infty} \beta_{W_n} \int_0^t \left[ \frac{g_{i,n+1}(x)}{pa\lambda_n} + \frac{g'_{i,n+1}(x)}{pa\lambda_n\mu_n} \right] \omega(x)dx, \quad (28)$$
respectively. Equations (28)-(33) represent the expected cumulative number of tasks completable, the instantaneous and the cumulative task completion ratios, the expected cumulative number of tasks canceled, and the instantaneous and the cumulative task incompletion ratios at the time point $t$, given that the $l$-th debugging is complete at time point $t = 0$, respectively.

5. Numerical Examples

We present several numerical examples on software performability analysis based on the above measures. We apply the model of Moranda [8] to the hazard rate and the restoration rate in the numerical examples, i.e.,

$$D_{c,D}^n = \binom{0,0}{1}, \quad E_{r,E}^n = \binom{0,0}{1},$$

respectively, and cite the estimates of $D_{c,E}$ and $E_{r}$ from [20], i.e., we use the following values:

$$a = 0.8$$

where we set $a = 0.8$ (see [20] for the detail of the parameter estimation).

For the distributions of the processing time of a task, $Y_{i,t}$, and the processing time limit, $Y_{i,t}$, we apply the gamma distribution whose density is denoted as

$$f_{\tilde{Q},\tilde{D}}(x) = \frac{x^{\tilde{Q}-1}e^{-x/\tilde{D}}}{\Gamma(\tilde{Q})\tilde{D}^{\tilde{Q}}}, \quad x > 0, \tilde{Q} > 0, \tilde{D} > 0.$$
respectively. Equations (28)-(33) represent the expected cumulative number of tasks completable, the instantaneous and the cumulative task completion ratios, the expected cumulative number of tasks canceled, and the instantaneous and the cumulative task incompletion ratios at the time point \( t \), given that the \( l \)-th debugging is complete at time point \( 0, t \) respectively.

5. Numerical Examples

We present several numerical examples on software performability analysis based on the above measures. We apply the model of Moranda [8] to the hazard rate and the restoration rate in the numerical examples, i.e., \( (0, 0.1) \) and \( (0, 0.1) \), respectively, and cite the estimates of \( D_c \) and \( E_r \) from [20], i.e., we use the following values:

\[
D_c = 0.246, \quad E_r = 0.940, \quad \ldots
\]

where we set \( a = 0.8 \) (see [20] for the detail of the parameter estimation).

For the distributions of the processing time of a task, \( \gamma(t) \), and the processing time limit, \( \gamma(t) \), we apply the gamma distribution whose density is denoted as

\[
1 \frac{1}{\Gamma(\alpha_y)} \frac{t^\alpha_y - 1}{\Gamma(\alpha_y)}
\]

where \( \alpha_y = 2.0, \quad \beta_y = 4.0 \times 10^2, \quad \alpha_r = 1.0 \times 10^3, \quad \eta = 2.0 \).

Figure 3 shows the time-dependent behaviors of the instantaneous task completion ratio, \( \nu^l(t, l) \), in Equation (29) and the instantaneous software availability, \( \nu^l(t, l) \), in Equation (12). This figure tells us that the new measure considering the real-time property gives more pessimistic evaluation than the traditional one \( A(t, l) \).

Figures 4 and 5 show \( \nu^l(t, l) \) for various numbers of debuggings, \( l \), and various values of the perfect debugging probability, \( a \), respectively. We can see that software performability also improves as the debugging progresses and that the higher debugging ability makes a greater contribution to the improvement of software performability as well.

Figure 4. \( \nu^l(t, l) \) for various numbers of debuggings, \( l(v_r = 0.1, \quad \alpha_y = 2.0, \quad \alpha_r = 4.0 \times 10^2, \quad \alpha_r = 1.0 \times 10^3, \quad \eta = 2.0) \).
Figure 5. Dependence of $v^1(t, l)$ on perfect debugging probability, $a(l = 0, \nu_y = \nu_r = 2.0, \alpha_y = 4.0 \times 10^2, \alpha_r = 1.0 \times 10^3, p = 0.8, \eta = 2.0)$.

Figure 6 shows the dependence of $v^1(t, l)$ on the value of $p$, representing the probability that the debugging activity is performed when the system is down. This figure indicates that software performability is evaluated lower in the early stage of the operation phase but more improves with the lapse of time as the value of $p$ increases. The larger value of $p$ gives the following two impacts: (I) inherent software reliability growth occurs earlier, on the other hand, (II) the unavailable (restoration) time tends to be longer since the mean time of restoration with debugging, $E[L^n_r] = 1/\mu_r$, is assumed to be the increasing function of $n$. As to the larger $p$, impact (II) appears in the early stage of the operation phase and then impact (I) becomes larger gradually with the lapse of time.

Figure 6. Dependence of $v^1(t, l)$ on $p(l = 0, \nu_y = \nu_r = 2.0, \alpha_y = 4.0 \times 10^2, \alpha_r = 1.0 \times 10^3, \eta = 2.0)$. 
Figure 7 shows the dependence of the instantaneous task incompletion ratio, \( \nu^2(t, l) \), in Equation (32) on the distributions of the processing time, \( F_Y(t) \), and the processing time limit, \( F_{T_r}(t) \), where we set the parameters \( \nu \) and \( \alpha \) as equalize the expectations of \( Y \) and \( T_r \) among the three cases in this figure, i.e.,

- case (i) \( \nu_Y = \nu_{T_r} = 1.0, \alpha_Y = \alpha_{Y_0} = 500.0, \alpha_{T_r} = \alpha_{T_{r_0}} = 200.0 \) (exponential distribution),
- case (ii) \( \nu_Y = \nu_{T_r} = 2.0, \alpha_Y = 2\alpha_{Y_0}, \alpha_{T_r} = 2\alpha_{T_{r_0}} \) (gamma distribution of order two),
- case (iii) \( \nu_Y = \nu_{T_r} = 4.0, \alpha_Y = 4\alpha_{Y_0}, \alpha_{T_r} = 4\alpha_{T_{r_0}} \) (gamma distribution of order four).

![Figure 7: Dependence of \( \nu^2(t, l) \) on distribution functions, \( F_Y(t) \) and \( F_{T_r}(t) \) (\( l = 0, p = 0.8, \eta = 2.0 \)).](image)

In all above cases, \( \text{E}[Y] = 1/500 \) and \( \text{E}[T_r] = 1/200 \). As to the difference of the characteristics among the three cases, the densities of \( Y \) and \( T_r \) in case (i) are monotonically decreasing functions, whereas those of in cases (ii) and (iii) are unimodal functions. This figure indicates that the performability evaluation becomes higher as the value of the shape parameter \( \nu \) becomes larger. As to the variance of the processing time, \( 1/(\alpha_{Y_0}^2) \) for case (i), \( 2/(2\alpha_{Y_0})^2 = 1/(2\alpha_{Y_0}^2) \) for case (ii), and \( 4/(4\alpha_{Y_0})^2 = 1/(4\alpha_{Y_0}^2) \) for case (iii), and it should be noted that \( 1/(\alpha_{Y_0}^2) > 1/(2\alpha_{Y_0}^2) > 1/(4\alpha_{Y_0}^2) \). The same holds for the variance of the processing time limit. We can see that the smaller dispersion-degrees of the processing time and the processing time limit rise the software performability evaluation.

For the mean value function of \( \{N(t), t \geq 0\} \), we apply the Weibull process, i.e., \( \text{E}[N(t)] = \Omega(t) = \xi t^\phi \ (t \geq 0; \xi > 0, \phi > 0) \). Figure 8 shows the dependence of the cumulative task completion ratio, \( \gamma^I(t, l) \), in Equation (30) on parameter \( \phi \); this reflects the intensity of the task arrival process. Especially in the case of \( \phi = 1.0 \), \( \{N(t), t \geq 0\} \) follows a homogeneous Poisson process and then \( \gamma^I(t, l) \) is given by

\[
\gamma^I(t, l) = \frac{1}{\xi} \sum_{i=0}^{l} \left( \sum_{n=i}^{\infty} \beta_{W_n} \left[ \frac{G_{i,n+1}(t)}{pa\lambda_n} + \frac{g_{i,n+1}(t)}{pa\lambda_n\mu_n} \right] \right). 
\] (35)
It is noted that $\gamma^l(t,l)$ in the case of $\phi=1.0$ is independent of the task arrival process. We can see that the software performability evaluation rises with the increasing $\phi$ after a certain lapse of time, i.e., the higher task arrival rate results in the higher performability evaluation although the opposite tendency appears in the short time interval immediately after the beginning of the system operation in this figure. In other words, the performability evaluation based on $\gamma^l(t,l)$ is generally susceptible to the task arrival process, whereas $\nu^l(t,l)$ is independent of the task arrival process from the form of Equation (29).

![Figure 8. Dependence of $\gamma^l(t,l)$ on $\phi$](image)

Figure 8. Dependence of $\gamma^l(t,l)$ on $\phi(l=0, \nu_y = \nu_{r_t} = 2.0, \alpha_r = 4.0 \times 10^2, \alpha_{r_t} = 1.0 \times 10^3, p = 0.8, \eta = 2.0, \xi = 1.0)$.

6. Concluding Remarks

In this paper, we have proposed the operation-oriented software performability model considering the dynamic software reliability growth process and the operation-oriented restoration scenarios. Assuming the operating regulation that the system can process the multiple tasks simultaneously, we have particularly incorporated the variety of the tasks in terms of its arrival process and processing time limit into the model, i.e., we have discussed the case where the cumulative number of the tasks arriving at the system up to a given time point follows the NHPP and the processing time limit of a task is treated as a random variable. We have analyzed the distribution of the number of tasks whose processes can be complete with the concept of the infinite-server queueing model. From the model, we have derived several software performability measures considering the real-time property. They have been given as the functions of time and the number of debuggings. We have also illustrated the several numerical examples of these measures to investigate the impacts of the inherent reliability growth, the operational restoration, the system's ability in task processing, and the task arrival characteristics on the software performability evaluation. The results obtained from the numerical examples in Section 5 are summarized as follows:

(i) The software performability measures proposed here give harsher evaluation than traditional availability ones.

(ii) The inherent software reliability growth also improves software performability.
(iii) Parameter \( p \) representing the probability that the debugging is performed in system down is a kind of subjective one. Setting the value of \( p \) accompanies the trade-off on software performability, i.e., we will choose either we intend to improve the future software performability though the performability in the early stage of operation is low, or we restrain the decrease in software performability in the early stage of operation and do not expect the future performability improvement very much instead.

(iv) The task arrival process affects software performability evaluation based on only the cumulative measures; the instantaneous measures are independent of the task arrival process.

(v) The external factors such as the processing time and the processing time limit which are the task characteristics also affect software performability evaluation. Software performability evaluation is higher as the variation in the task characteristics becomes smaller even in the case where the averages of the task characteristics display similar values.

Most of previous works such as [2, 12, 17] have conducted the performability evaluation of the software system only in steady states; this means that they have not described the inherent software reliability growth process although there originally exists a possibility of dynamic quality reliability growth of software systems. This paper has overcome the above issue and this model enables us to evaluate the real-time property as well; this knowledge is very meaningful.

We have presented the numerical examples in the situation where the parameters related to the task arrival process, \( \xi \) and \( \phi \), or the task processing time, \( \nu_I \) and \( \alpha_I \) \( (I \in \{Y, T_r\}) \) in Equation (34) are known empirically in advance since we have not obtained the details of the data concerned with the task arrival and processing. Reasonable estimation of these characteristics remains as a future work.

Acknowledgments

This work was supported in part by Grant-in-Aid for Scientific Research (C) of Japan Society for the Promotion of Science under Grant Nos. 20510136 and 22510150, and Takahashi Industrial and Economic Research Foundation.

References


Authors' Biographies:

**Koichi Tokuno** received the B.S.E. and M.S. degrees from Hiroshima University, Japan in 1990 and 1992, respectively, and the Ph.D. degree from Tottori University, Japan in 1999. From 1992 to 1994, he worked at the Japan Steel Works Ltd., Hiroshima Prefecture, Japan. From 1994 to 2001, he was an Assistant Professor at Tottori University, Japan. Since 2001, he has been working as an Associate Professor at the Graduate School of Engineering, Tottori University, Tottori-shi, Japan. His research area includes software reliability engineering, stochastic modeling, and system dependability measurement and assessment. He is a regular member of the Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan, the Operations Research Society of Japan, the Reliability Engineering Association of Japan and the Japan Society for Software Science and Technology. Dr. Tokuno is serving as a Regional Editor of the International Journal of Reliability and Applications.

**Shigeru Yamada** received the B.S.E., M.S., and Ph.D. degrees from Hiroshima University in 1975, 1977, and 1985, respectively. Since 1993, he has been working as a Professor at the Graduate School of Engineering, Tottori University, Tottori-shi, Japan. He has published numerous technical papers in the area of software reliability models, project management, reliability engineering, and quality control. He has authored several books entitled *Introduction to Software Management Model* (Kyoritsu Shuppan, 1993), *Software Reliability Models: Fundamentals and Applications* (JUSE, Tokyo, 1994), *Statistical Quality Control for TQM* (Corona Publishing, Tokyo, 1998), *Software Reliability: Model, Tools, Management* (The Society of Project Management, 2004), and *Quality-Oriented Software Management* (Morikita Shuppan, 2007). Dr. Yamada received the Best Author Award from the Information Processing Society of Japan in 1992, the TELECOM System Technology Award from the Telecommunications Advancement Foundation in 1993, the Paper Award from the Reliability Engineering Association of Japan in 1999, the International Leadership Award in Reliability Engineering Research from the ICQRIT/SREQOM in 2003, the Best Paper Award from the Society of Project Management in 2006, the Leadership Award from the ISSAT in 2007, and the Outstanding Paper Award at the IEEE-IEEM2008. He is a regular member of the Institute of Electronics, Information and Communication Engineers, the Information Processing Society of Japan, the Operations Research Society of Japan, the Japan SIAM, the Reliability Engineering Association of Japan, Japan Industrial Management Association, the Japanese Society for Quality Control, the Society of Project Management of Japan, and the IEEE.