A Software Cost Model with Reliability Constraint under Two Operational Scenarios

Satoru UKIMOTO and Tadashi DOHI

Department of Information Engineering, Graduate School of Engineering
Hiroshima University, 1–4–1 Kagamiyama, Higashi-Hiroshima, 739–8527 Japan
dohi@rel.hiroshima-u.ac.jp

Abstract

In this paper we extend the reliability constrained cost minimization (RCCM) model by Helander et al. (1998) from two view points: time non-homogeneous property on software failure-occurrence process and gap between testing and operational phases of software product. The expected cost minimization with reliability constraint is formulated as a non-linear minimization problem under alternative scenario on the operation. We develop an effective optimization algorithm based on the Kuhn-Tucker conditions and provide an illustrative example on how to design a component-based software.

Keywords: software cost model, effort allocation, reliability constraint, operational phase, non-homogeneous Poisson process, non-linear optimization.

1 Introduction

As recent software development process is much more complex, the effective software cost management is of great importance for the project manager in a competitive market. Since software testing cost is usually rather expensive in the total development cost, the software testing-resource allocation is one of the most significant issues in the software project management. The search-based software engineering (SBSE) aims at optimizing the software development process and is becoming much popular to realize the economic planning in software development. Taking account of both software economics and reliability, the software development project should be managed quantitatively with any sophisticated optimization technique. Nevertheless, in the actual software development, it is quite rare to apply such an SBSE approach.

Ohtera and Yamada [13] first considered a simple software reliability model dependent on the testing effort, and formulated a testing-resource allocation problem. The basic idea comes from the classical reliability allocation problems for component-based systems. Since then, many authors have formulated several kinds of software resource allocation problems from various points of view. Zaheidi and Ashrafi [19] used AHP (Analytic Hierarchy Process) to solve a software reliability allocation model and determined reliability goals at the planning and design stages of the software project. Berman and Ashrafi [2] extended the original works in [1], [13] and gave non-linear programming algorithms for more complex resource-allocation problems with constraints. Leung [6, 7, 8] treated static software cost
models in time, and discussed different optimization problems with various objective functions such as worst case failure probability, software development cost, worst case utility, etc. Hou et al. [4] considered a different testing-resource allocation problem based on the hypergeometric distribution software reliability model.

Wadekar and Gokhale [16] and Lyu et al. [9] also formulated the similar optimization problems for the software resource allocation and the software reliability allocation. Helander et al. [3] developed two problems; reliability-constrained cost-minimization (RCCM) and budget-constrained reliability-maximization (BCRM), under a software development scenario. Though their approach is quite similar to the classical non-linear programming approaches in the earlier works, it gives the detailed procedure with reality in applying the software resource allocation problem to the real design problem for a component-based software system. Ngo-The and Ruh [12] formulated a somewhat different problem for the software release planning by allocating the software development resources and gave an interesting case study. Recently, Pietrantuono et al. [14] used an architecture-based software reliability model and considered a reliability and testing time allocation problem. They also gave an empirical study for a program developed in the European space agency. In this way, considerable attentions have been received for the software resource allocation problems.

In this paper, we focus on the software testing-resource allocation with operational profile. Musa et al. [10, 11] recommended to use the operational profile in estimating the user’s operational circumstance and the operational software reliability, where the profile consists of the usage frequency and the execution probability of software components. Recently, Ukimoto et al. [15] considered a testing-resource allocation problem for a component-based software system taking account of the operational profile. In the past literature, what dealt with the operational profile explicitly for the optimization formulations were only Leung [7], Helander et al. [3] and Ukimoto et al. [15]. In other words, almost all related work did not take an effect of the dynamic environmental change from testing to operational phases into consideration.

Here we re-consider a simple optimization problem by Helander et al. [3], where the software failure intensity is controlled by minimizing the relevant expected cost with a reliability constraint. However, it should be worth noting in [3] that the failure intensity for each software component is controlled and that the time-homogeneous scenario is assumed to describe the operational phase. More precisely, Helander et al. [3] implicitly assume that the software debugging process does not change even in the operational phase. In this paper we introduce alternative scenario on the operational phase, and propose a model to bridge between the software testing phase and operational phase. Under this scenario, we formulate the different reliability constrained cost minimization problem from [3], and give the optimal design policy on the allowable number of faults in each software component. A numerical example is presented to show how to derive the optimal design policy. Finally, the paper is concluded with some remarks and future study.

2 RCCM Model

We introduce RCCM (Reliability Constrained Cost Minimization) model by Helander et al. [3]. Suppose that software consists of $n$ components whose failure intensity functions are given by $\lambda_i$ $(i = 1, 2, \ldots, n)$. More precisely, let $\{N_i(t), t \geq 0\}$ be the cumulative
number of software failures occurred by time \( t \) in the \( i \)-th component and be a stochastic (non-decreasing) point process in continuous time. It is said that \( N_i(t) \) is a homogeneous Poisson process (HPP) if the following conditions hold:

- \( N_i(0) = 0 \),
- \( \{N_i(t), t \geq 0\} \) has independent and stationary increments,
- \( \Pr\{N_i(t + \Delta t) - N_i(t) = 2\} = o(\Delta t) \),
- \( \Pr\{N_i(t + \Delta t) - N_i(t) = 1\} = \lambda_i \Delta t + o(\Delta t) \),

where the intensity of an HPP, \( \lambda_i \), denotes the instantaneous software failure-occurrence rate per each failure for \( i \)-th component, and \( o(\Delta t) \) is the higher term of the infinitesimal time \( \Delta t \). Then the probability mass function (p.m.f.) of the HPP is given by

\[
\Pr\{N_i(t) = x\} = \frac{\lambda_i^x}{x!} \exp[-\lambda_i t],
\]

where the mean value functions, \( E[N_i(t)] = \lambda_i t \) and \( E[\sum_{i=1}^n N_i(t)] = \sum_{i=1}^n \lambda_i t \), denote the expected cumulative numbers of software failures occurred in the \( i \)-th component and the whole software system by time \( t \), respectively.

Let \( TC(\lambda_1, \lambda_2, \ldots, \lambda_n; \tau) \) and \( R(\lambda_1, \lambda_2, \ldots, \lambda_n; \tau) \) be the expected total operational cost and the quantitative software reliability, respectively, as functions of software intensity \( \lambda_i \ (i = 1, 2, \ldots, n) \), where \( \tau \) (> 0) denotes the operational period of software after the release. Suppose that the expected total operational cost is given by the sum of individual expected operational cost, \( C_i(\lambda_i) \), for each software component \( i \ (= 1, 2, \ldots, n) \):

\[
TC(\lambda_1, \lambda_2, \ldots, \lambda_n; \tau) = \sum_{i=1}^n C_i(\lambda_i; \tau). \tag{2}
\]

From the property of expected operational cost for each component \( i \), we assume that the function \( C_i(\lambda_i; \tau) \) is monotonically decreasing and convex in \( \lambda_i \) without any loss of generality. Helander et al. [3] assume the following three kinds of cost functions; (i) linear function (Linear), (ii) logarithmic exponential (LogExp) function and (iii) inverse power (InvPow) function:

- **Linear**: \( C_i(\lambda_i; \tau) = -\alpha_i \lambda_i \tau + \beta_i \),
- **LogExp**: \( C_i(\lambda_i; \tau) = \beta_i \ln(1 - \exp[-\lambda_i \tau]) \),
- **InvPow**: \( C_i(\lambda_i; \tau) = \frac{\beta_i}{(\lambda_i \tau - \delta_i)\gamma_i} \),

for \( i = 1, 2, \ldots, n \), where \( \alpha_i \ (> 0) \), \( \beta_i \ (> 0) \) and \( \delta_i \ (> 0) \) are constant cost parameters. Then the RCCM with a fixed \( \tau \) is formulated as the following minimization problem with respect to the software intensity:

\[
\min_{\lambda_i; i = 1, 2, \ldots, n} \quad TC(\lambda_1, \lambda_2, \ldots, \lambda_n; \tau),
\]

s.t. \( R(\lambda_1, \lambda_2, \ldots, \lambda_n; \tau) \geq \rho \),

\( \lambda_i \geq 0 \) for \( i = 1, \ldots, n \), \tag{3}

where \( \rho \) in Eq.(3) is the minimum requirement of quantitative software reliability level and \( R(\lambda_1, \lambda_2, \ldots, \lambda_n; \tau) \) means the probability that the software does not fail during the operational period \( \tau \).
It is common to see that the cumulative number of software failures caused by software faults is estimated by the operational profile \([10],[11]\) inferred in the design phase. Let \(\phi_i\) be the frequency to use the component \(i\) in the operational phase with \(\sum_{i=1}^{n} \phi_i = 1\). Similar to Helander et al. \([3]\) and Ukimoto et al. \([15]\), let \(\mu_{ij}\) and \(p_j\) denote the ratio of the \(j\)-th software operation \((j = 1, 2, \ldots, m)\), processed by component \(i\) and the probability that the \(j\)-th software operation is executed, respectively, where \(\sum_{j=1}^{m} p_j = 1\) and \(\sum_{i=1}^{n} \sum_{j=1}^{m} \mu_{ij} = 1\).

More formally, define

\[\mathbf{p} = (p_1, \ p_2, \ \ldots, \ p_m)\]  

\[\mathbf{\mu} = \begin{bmatrix} \mu_{11}, & \mu_{12}, & \cdots, & \mu_{1m} \\ \mu_{21}, & \mu_{22}, & \cdots, & \mu_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ \mu_{n1}, & \mu_{n2}, & \cdots, & \mu_{nm} \end{bmatrix}\]

Then the frequency to use the component \(i\) is given by

\[\phi_i = \mathbf{p} \mathbf{\mu} = \sum_{j=1}^{m} \mu_{ij} p_j,\]

so that the quantitative software reliability is derived by

\[R(\lambda_1, \lambda_2, \ldots, \lambda_n; \tau) = \exp\left\{-\sum_{i=1}^{n} \phi_i \lambda_i \tau\right\}\]

from the HPP assumption. By solving the minimization problem in Eq.\((3)\), the optimal software intensity for respective components, \(\lambda_i^* (i = 1, 2, \ldots, n)\), can be obtained so as to satisfy the reliability constraint.

For the minimization problem in Eq.\((3)\) with inequality constraints, it is straightforward to get necessary condition of optimality (Kuhn-Tucker condition) and to characterize the optimal software intensity \(\lambda_i^*\). Let \(u\) and \(k\) be the Lagrange multiplier and the slack variable, respectively. Then the Kuhn-Tucker conditions for three cost functions are given by

**Linear:**

\[-\alpha_i + u \phi_i \tau = 0,\]

\[\ln(\rho) + \sum_{i=1}^{n} \phi_i \lambda_i \tau - k = 0.\]

**LogExp:**

\[-\beta_i \frac{1}{1 - \exp(-\lambda_i)} + u \phi_i \tau = 0,\]

\[\ln(\rho) + \sum_{i=1}^{n} \phi_i \lambda_i \tau - k = 0.\]
Table 1. Optimally allocation policies.

<table>
<thead>
<tr>
<th>Cost</th>
<th>HPP</th>
<th>NHPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear</td>
<td>$\lambda_i^* = \begin{cases} -\ln(\rho)/\phi_k \tau &amp; (i = k) \ 0 &amp; (i \neq k) \end{cases}$</td>
<td>$a_i^* = \begin{cases} -\frac{\ln(\rho)}{F(\tau)} &amp; (i = k) \ 0 &amp; (i \neq k) \end{cases}$</td>
</tr>
<tr>
<td>LogExp</td>
<td>$\lambda_i^* = \text{ln} \left[ 1 + \frac{\beta_i}{u \rho \tau} \right]$</td>
<td>$a_i^* = \text{ln} \left[ 1 + \frac{\beta_i}{u \rho F(\tau)} \right]$</td>
</tr>
<tr>
<td>InvPow</td>
<td>$u^* = \frac{\alpha_i}{\rho} \left( \frac{\tau^{\alpha_i + 1} \sum_{i=1}^{n} \phi_i \lambda_i \tau - k}{\tau - \ln(\rho) - \sum_{i=1}^{n} \phi_i \lambda_i \tau - k} \right)^{\alpha_i + 1}$</td>
<td>$u^* = \frac{\alpha_i}{\rho} \left( \frac{\sum_{i=1}^{n} F(\tau)^{\alpha_i + 1} \beta_i^{\alpha_i + 1}}{\tau - \ln(\rho) - \sum_{i=1}^{n} F(\tau) \delta_i} \right)^{\alpha_i + 1}$</td>
</tr>
</tbody>
</table>

InvPow:

\[-\alpha_i \beta_i \left( \lambda_i - \delta_i \right)^{\alpha_i + 1} + u \phi_i \tau = 0, \quad (12)\]

\[\ln(\rho) + \sum_{i=1}^{n} \phi_i \lambda_i \tau - k = 0. \quad (13)\]

Based on these results, we summarize the optimal software intensity $\lambda_i^*$ for three models in Table 1, where $u^*$ is the optimal Lagrange multiplier which is given as a unique solution of nonlinear equations.

3 NHPP-based Modeling

In the original RCCM, it is assumed that the software failure occurrence process for each component $i$ is described by an HPP with software intensity $\lambda_i$ and that the inter-failure time distribution is exponentially distributed with rate $\lambda_i$. In a fashion dissimilar to the operational phase, it is well known that the software testing process is described by any time non-stationary process. The common experiences suggest that the software fault-detection process in testing is represented by a non-homogeneous Poisson process (HHPP). Suppose that the software failures are caused by software faults. Let $\{N_i(t), t \geq 0\}$ be the cumulative number of software faults detected by time $t$ for $i$-th component. It is said that $N_i(t)$ is an NHPP if the following conditions hold:

- $N_i(0) = 0,$
- $\{N_i(t), t \geq 0\}$ has independent increments,
- $\text{Pr}\{N_i(t + \Delta t) - N_i(t) \geq 2\} = o(\Delta t),$ 
- $\text{Pr}\{N_i(t + \Delta t) - N_i(t) = 1\} = \lambda_i(t; \theta_4) \Delta t + o(\Delta t),$

where $\lambda_i(t; \theta_4)$ is the software intensity function of the $i$-th component. In the above definition, $\theta = (\theta_1, \theta_2, \ldots, \theta_n)$ is the model parameters (vector) included in each intensity function. Then the p.m.f. of the total cumulative number of software faults detected in
testing is given by

\[ \Pr\{N(t) = x\} = \Pr\{\sum_{i=1}^{n} N(t) = x\} = \frac{\Lambda(t; \theta)^x}{x!} \exp[-\Lambda(t; \theta)], \]  \quad (14)  

\[ \Lambda(t; \theta) = \sum_{i=1}^{n} \Lambda_i(t; \theta_i) = \sum_{i=1}^{n} \int_{0}^{t} \lambda_i(x; \theta_i)dx, \]  \quad (15)  

where the functions \( \Lambda(t; \theta) = E[N(t)] \) and \( \Lambda_i(t; \theta_i) = E[N_i(t)] \) are called the mean value functions. For the two-parameters case with \( \theta_i = (a_i, b_i) \) \( (i = 1, 2, \ldots, n) \), the mean value functions have the product forms of \( \Lambda_i(t; \theta_i) = a_iF_i(t; b_i) \), where \( F_i(t; b_i) \) are the cumulative distribution functions (c.d.f.) of independent and identically distributed fault-detection times (see [11]) in the \( i \)-th component, and \( a = \sum_{i=1}^{n} a_i \) is the expected initial number of software faults remaining before testing, \( i.e., \lim_{t \to \infty} \Lambda(t; \theta) = a \) and \( \lim_{t \to \infty} \Lambda_i(t; \theta_i) = \lim_{t \to \infty} F_i(t; b_i) = a_i \).

Yamada and Osaki [17] define the software debug rate in each component:

\[ d_i(t) = \frac{d\Lambda_i(t; a_i, b_i)/dt}{\Lambda_i(\infty; a_i, b_i) - \Lambda_i(t; a_i, b_i)} = \frac{f_i(t; b_i)}{a_i\{F_i(\infty; b_i) - F_i(t; b_i)\}} = \frac{f_i(t; b_i)}{\{1 - F_i(t; b_i)\}}, \]  \quad (16)  

where the function \( f_i(t; b_i) = dF_i(t; b_i)/dt \) is the probability density function (p.d.f.) of \( F_i(t; b_i) \), so that \( d_i(t) \) is equivalent to the failure rate of the c.d.f. \( F_i(t; b_i) \) and is independent of \( a_i \). It should be noted that the NHPP-based model describes the debugging process in the software testing. On the other hand, the RCCM focuses on the operational phase after the release of software, but does not link to the testing phase.

### 4 Extended RCCM Model

Kimura et al. [5], and Yang and Xie [18] independently consider a scenario to change from software testing phase to operational phase, and assume that the software intensity function \( \lambda(t; \theta) = \sum_{i=1}^{n} \lambda_i(t; \theta_i) \) becomes a constant just after releasing the software. For a given release time \( t_0 \), the software intensity is give as \( \lambda_i(t_0; \theta_i) = \lambda_i \). In other words, the RCCM model [3] implicitly assumes the above scenario in the operational phase and controls the software intensity, \( \lambda_i \) \( (i = 1, 2, \ldots, n) \), in each component. We call this scenario that the software intensity becomes a constant after releasing the software Scenario 1.

However, the above scenario seems to be rather questionable, because the resulting fault counting process is an HPP with mean value function \( \lambda_i(t_0; \theta_i)t \) and is linearly increasing in \( t \), regardless of the expected number of remaining faults before testing in the component \( i \), say, \( a_i \). This is an inconsistent scenario to bridge the software testing phase and the software operational phase, even though the debugging activity terminates at time \( t_0 \). In this paper we propose alternative scenario, Scenario 2, where the software debug rate \( d_i(t) \) in Eq.(16), but not the software intensity function \( \lambda_i(t) \), becomes a constant after the release, say, \( d_i(t_0) \). This implies that the software failure-occurrence time distribution \( F_i(t; b_i) \) after the release time \( t_0 \) \( (t \leq t_0) \) in the \( i \)-th component is an exponential distribution with rate \( d_i(t_0) \), say \( b_i = d_i(t_0) \) and \( F_i(t; b_i) = a_i\{1 - \exp(-d_i(t_0)t)\} \) iff the software debug rate is constant.
Figure 1. Two scenarios in software intensity function.

for \( t \geq t_0 \). Then, we have

\[
\lambda_i(t; b_i) = \begin{cases} 
  a_i f_i(t; b_i) & (0 \leq t < t_0) \\
  a_i f_i(t_0; d_i(t_0)) \exp(-d_i(t_0)[t-t_0]) & (t_0 \leq t),
\end{cases}
\]

\[
\Lambda_i(t; \theta_i) = \begin{cases} 
  a_i F_i(t; b_i) & (0 \leq t < t_0) \\
  a_i F_i(t; d_i(t_0)) + a_i \{1 - F_i(t_0; d_i(t_0))\} \{1 - e^{-d_i(t_0)[t-t_0]}\} & (t_0 \leq t).
\end{cases}
\]

Figure 1 illustrates the concept of two scenarios in this paper. It seems that Scenario 1 with constant intensity level in the operational phase leads to the linear increase of cumulative number of failures and does not reflect the testing effort spent in the testing. On the other hand, Scenario 2 implies that the software intensity changes to a specific exponential form with rate \( d_i(t_0) \) but results the consistent behavior of the cumulative number of failures.

Based on Scenario 2, we re-formulate RCCM model. Instead of the software intensity \( \lambda_i \) in the original RCCM model, we regard the expected initial number of faults in each component \( a_i \) as the decision variable. Note that \( a_i \) is a target variable in the software testing, so that the software testing manager will set the objective number of faults detected in each software component. For given \( d_i(t_0) = d_i, \ (i = 1, 2, \ldots, n) \), we formulate:

\[
\min_{a_i; i=1,2,\ldots,n} TC(a_1, a_2, \ldots, a_i; \tau)
\]
\[ R(a_1, a_2, \ldots, a_n; \tau) = \exp \left\{ -\sum_{i=1}^{n} a_i(1 - \exp[-d_i(t_0)\phi_i \tau]) \right\} \] 

(21)

Then, we derive the Kuhn-Tucker conditions for three cost functions by

**Linear:**

\[-\alpha + u(1 - \exp[-b_i\phi_i \tau]) = 0,\]

\[\ln(\rho) + \sum_{i=1}^{n} a_i(1 - \exp[-b_i\phi_i \tau]) - k = 0.\]

(22)

(23)

**LogExp:**

\[-\frac{\beta_i}{1 - \exp[-\lambda_i]} + u(1 - \exp[-b_i\phi_i \tau]) = 0,\]

\[\ln(\rho) + \sum_{i=1}^{n} a_i(1 - \exp[-b_i\phi_i \tau]) - k = 0.\]

(24)

(25)

**InvPow:**

\[-\frac{\alpha_i\beta_i}{(\lambda_i - \delta_i)^{\alpha_i+1}} + u(1 - \exp[-b_i\phi_i \tau]) = 0,\]

\[\ln(\rho) + \sum_{i=1}^{n} a_i(1 - \exp[-b_i\phi_i \tau]) - k = 0.\]

(26)

(27)

In Table 1, we also summarize the optimal number of software faults \(a_i^*\) \((i = 1, 2, \ldots, n)\) for three cost functions, where \(F(\tau) = 1 - \exp[-d_i(t_0)\phi_i \tau]\).

### 5 An Illustrative Example

We give an illustrative example on how to optimize the RCCM models for a component-based software system. Suppose a software system with independent 5 components. For the illustrative purpose, it is assumed that only 5 operations are executed on the system and that the operation \(j\) \((= 1, 2, 3, 4, 5)\) is selected and executed with probability \(p_j\). In each execution, the process runs on some of components, where the processing time of the operation \(j\) on the \(i\)-th component is given by \(\mu_{ij}\). In this example, we set:

\[ p = (0.25, 0.30, 0.10, 0.15, 0.25) \]

and

\[ \mu = \begin{bmatrix} 0.60, & 0.10, & 0.00, & 0.20, & 0.10 \\ 0.20, & 0.10, & 0.10, & 0.50, & 0.10 \\ 0.20, & 0.10, & 0.50, & 0.10, & 0.10 \\ 0.00, & 0.10, & 0.30, & 0.00, & 0.60 \\ 0.20, & 0.10, & 0.10, & 0.40, & 0.20 \end{bmatrix} \]
where $\sum_{j=1}^{5} p_j = 1$ and $\sum_{i=1}^{5} \sum_{j=1}^{5} \mu_{ij} = 1$. The, we have

$$\phi_i = (0.25, 0.10, 0.15, 0.30, 0.20).$$

(28)

In the planning of software execution, we set the minimum requirement of software reliability function by $\rho = 0.99$. To avoid the trivial cases, we consider two cost models; LogExp and InvPow in Table 1, where $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5) = (305, 238, 249, 332, 291)$ in LogExp and $(\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5) = (0.25, 0.30, 0.26, 0.34, 0.29)$, $(\beta_1, \beta_2, \beta_3, \beta_4, \beta_5) = (410, 320, 430, 293, 352)$, $\delta_1 = \delta_2 = \delta_3 = \delta_4 = \delta_5 = 0$. The other parameters are given by $t_0 = 0.50$ [CPU time], $\tau = 1.00$ [CPU time].

Under Scenario 2, we calculate the optimal policy $a^*_i (i = 1, 2, \ldots, 5)$, where the software debug rate for each component is given by $(d_1, d_2, d_3, d_4, d_5) = (0.001, 0.002, 0.003, 0.004, 0.005)$. For LogExp and InvPow in Table 1, we obtain

$$ (a^*_1, a^*_2, a^*_3, a^*_4, a^*_5) = (4.36, 4.34, 3.59, 2.92, 2.97), $$(29)

$$ (a^*_1, a^*_2, a^*_3, a^*_4, a^*_5) = (12.91, 13.21, 8.51, 3.30, 4.06), $$

(30)

respectively, for given $(b_1, b_2, b_3, b_4, b_5) = (0.001, 0.002, 0.003, 0.004, 0.005)$. From this result, it is seen that the software component with relatively larger debug rate involves a smaller number of initial faults content. Looking at the component 4 and the component 5, since the usage rate for the component 4 is greater than that for the component 5, it is seen that $a^*_4$ should be smaller than $a^*_5$.

6 Concluding Remarks

In this paper, we have extended the RCCM model by introducing the NHPP-based modeling with alternative scenario on the operational phase. We have formulated a nonlinear optimization problem with constraints and provided the optimal solutions for three kinds of cost functions. Though the original RCCM model was an optimization problem to control the software intensity function for each software component, the decision variable in our model was the expected target number of software faults. So, it implies that the software project manager sets up the target number of software faults detected in the testing phase. In a fashion similar to our model, it would be possible to control the fault detection rate, say, $b_i (i = 1, 2, \ldots, n)$, directly, for fixed $a_i$. Also, similar to Helander et al. [3], we will also consider the budget-constrained reliability-maximization (BCRM) model in the same framework.

Acknowledgments

This research was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research (C), Grant No. 23517101 (2011–2013).

References


**Authors**

Satoru Ukimoto received the B.Sc. (Engineering) from Hiroshima University, Japan, in 2011. In 2011, he joined Hiroshima City as a Technical staff. His research interest when he was an undergraduate student of Hiroshima University was Software Project Management.

Tadashi Dohi received the B.Sc. (Engineering), M.Sc. (Engineering), and Ph.D. (Engineering) from Hiroshima University, Japan, in 1989, 1991, and 1995, respectively. In 1992, he joined the Department of Industrial and Systems Engineering, Hiroshima University, Japan, as an Assistant Professor. Now he is a Full Professor in the Department of Information Engineering, Faculty of Engineering, Hiroshima University, Japan, since 2002. In 1992, and 2000, he was a Visiting Research Scholar at University of British Columbia, Canada, and Duke University, USA, respectively, on leave of absence from Hiroshima University. His research areas include Software Reliability Engineering, Dependable Computing, and Performance Evaluation. Dr. Dohi has published over four hundred peer-reviewed papers in the above research topics. He is a Regular Member of ORSJ, JSIAM, IEICE, REAJ, and IEEE. He also served as General Co-chair of AIWARM 04-08, WoSAR 08-10, MENS’10, APARM’10-12, ISSRE’11, ATC’12.