Hindawi
Journal of Optimization
Volume 2018, Article ID 6948656, 7 pages
https://doi.org/10.1155/2018/6948656

Research Article

Optimizing Testing-Resource Allocation Using Architecture-Based Software Reliability Model

Hiroyuki Okamura and Tadashi Dohi

Department of Information Engineering, Graduate School of Engineering, Hiroshima University, Japan

Correspondence should be addressed to Hiroyuki Okamura; okamu@hiroshima-u.ac.jp

Received 1 June 2018; Accepted 9 September 2018; Published 27 September 2018

Guest Editor: Bestoun S. Ahmed

In the management of software testing, testing-recourse allocation is one of the most important problems due to the tradeoff between development cost and reliability of released software. This paper presents the model-based approach to design the testing-resource allocation. In particular, we employ the architecture-based software reliability model with operational profile to estimate the quantitative software reliability in operation phase and formulate the multiobjective optimization problems with respect to cost, testing effort, and software reliability. In numerical experiment, we investigate the difference of the presented optimization problem from the existing testing-resource allocation model.

1. Introduction

Software testing is one of the most important phases to develop the highly reliable software products in software developments. In software testing, many developers, often called testers, try to find software bugs through the execution of test cases. As the number of test cases executed in software testing increases, the reliability of software product also increases by removing software bugs introduced in design and implement phases. However, it requires much efforts to increase the number of test cases executed in software testing. Thus from both cost and reliability points of view, it is important to make a plan for the allocation of testing resources such as the number of testers before software testing.

For this purpose, several papers have tried to solve the testing-resource problem with the probabilistic models. Ohtera and Yamada [1] 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 (e.g., see [2]). Zaheidi and Ashrafi [3] 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. Ashrafi and Berman [4], Berman and Ashrafi [5], Yamada et al. [6], and Nishiwaki et al. [7] extended the original works in [1, 8] and gave nonlinear programming algorithms for more complex resource allocation problems with constraints. Leung [9–11] discussed different optimization problems with various objective functions such as worst case failure probability, software development cost, and worst case utility. Hou et al. [12] considered a different testing-resource allocation problem based on the hypergeometric distribution software reliability model. Jha et al. [13], Wadekar and Gokhale [14], Lyu et al. [15], and Yang and Xie [16] also formulated various optimization problems for the software resource allocation and the software reliability allocation. Helander et al. [17] developed two problems: reliability-constrained cost minimization and budget-constrained reliability maximization under a software development scenario. Though their approach is quite similar to the classical nonlinear programming in the earlier works, it gives the detailed procedure with reality in applying the software resource allocation problem to the real problem. Ngo-The and Ruh [18] 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. [19] 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 study, we focus on the testing-resource allocation with operational profile. The operational profile is a quantitative representation of how the system will be used in user environment [20]. In fact, there are several representations for the operational profile. Ukimoto et al. [21] considered the software testing-resource allocation where the operational profile is time fraction of execution for modules, and they regarded the operational environment as the testing environment with different time scale. This idea is based on the accelerated life testing model where the testing environment is assumed to be accelerated from the operational environment with only the elapsed time [21]. However, since software testing is the environment to detect software bugs, the testing environment might not be the time-accelerated environment of operational environment. Thus in the paper, we consider another representation of operational profile by using architecture-based software reliability model.

The architecture-based software reliability model is based on the architecture of the targeted software. In general, software system consists of a number of modules and executes modules according to a programmed logic, namely, the currently executed module changes with the passage of time in operational phase, which is called the execution path. If an execution path does not include any faulty module, the software never fails. That is, the software failure essentially depends on the software architecture and its execution paths. This is the basic concept of architecture-based software reliability model. Littlewood [22, 23] developed the earliest architecture-based software reliability models in operational phase. In his models, the execution path in operational phase is generated by a continuous-time Markov chain (CTMC) and a semi-Markov process. Laprie [24] also provided the similar model to the Littlewood [23] in a different way. Cheung [25] modeled the execution path by a discrete-time Markov chain (DTMC). Ledoux and Rubino [26] and Ledoux [27] extend the original Littlewood models to represent failover operation. Goseva-Popstojanova et al. [28, 29] established a theoretical relationship among different architecture-based software reliability models and compared them through an empirical case study. Singh et al. [30] provided an approach with UML to analyze the component systems which consist of software modules. In this paper, we use the architecture-based software reliability model to estimate the software reliability in operational phase. Compared to Ukimoto's method [21], our approach provides more accurate estimation of the software reliability in operational phase.

The rest of this paper is organized as follows. In Section 2, we first describe the models for testing cost and efforts in testing environment based on software reliability growth models (SRGM). After that, the architecture-based software reliability model is also introduced to formulate the quantitative software reliability measure. In particular, we assume two different situations for the system usage. In Section 3, we formulate the software testing-resource allocation problems: reliability-constrained cost minimization and budget-constrained reliability maximization. Section 4 is devoted to the numerical illustration of our models. In Section 4, we compare the optimal solutions of resource allocation by Ukimoto et al.'s model and our model and discuss the effect of representation of operational profiles on the testing-resource allocation. Finally, in Section 5, we conclude this paper with some remarks.

2. Model Description

2.1. Cost Model in Testing Environment. First we describe the testing cost model in testing environment, which is essentially the same as Ukimoto et al. [21]. The system consists of n components. The software testing starts at time $t = 0$, and the system should be released at time $t = t_r$. Let $N_i(t)$ be the cumulative number of detected faults of component i before testing time t. Consider the following model assumptions:

(i) There are a finite number of faults in each component before testing.

(ii) The fault detection rate for a component is proportional to the amount of testing efforts for the component.

Let a_i and $W_i(t)$ be the expected number of faults before testing and the amount of testing efforts for component i at testing time t. Then the probability mass function (p.m.f.) of the cumulative number of faults is given by

$$P\left(N_i(t) = k\right) = \frac{H_i(t)^k}{k!} \exp\left(-H_i(t)\right), \tag{1}$$

$$H_i(t) = a_i\left(1 - \exp\left(-r_i W_i(t)\right)\right), \tag{2}$$

where r_i is a fault detection rate per testing effort. The above equations are essentially same as the nonhomogeneous Poisson process (NHPP) based software reliability growth model (SRGM). By applying the testing effort function $W_i(t)$, we can represent a variety of fault detection processes. For instance, (cumulative) Rayleigh curve is typically used to a testing effort function. For the same of simplicity, this paper assumes the following linear testing effort function for all the modules:

$$W_i(t) = \omega_i t + \beta_i, \tag{3}$$

where ω_i is the testing effort per unit testing time and β_i is a fixed effort for component i.

Define the cost structure in testing environment:

(i) c_1: fixing cost of a software fault detected in testing phase.

(ii) c_2: testing cost per software testing effort.

Then the expected total cost for component i in software testing is given by

$$C_i\left(t_r\right) = c_1 H_i\left(t_r\right) + c_2 W_i\left(t_r\right). \tag{4}$$

Thus the total cost for software testing becomes

$$C\left(t_r\right) = \sum_{i=1}^{n} C_i\left(t_r\right). \tag{5}$$

Also the total amount of testing efforts for testing is given by

$$W(t_r) = \sum_{i=1}^{n} W_i(t_r). \tag{6}$$

2.2. Reliability Model in Operational Environment. Ukimoto et al. [21] assumed the expected cumulative number of faults at time t, that is, the time after the release:

$$H_i(t) = H_i(t_r + \phi_i(t - t_r)), \quad t \geq t_r. \tag{7}$$

This equation can be rewritten by

$$H_i(t) = H_i(t_r) + (H_i(t_r + \phi_i(t - t_r)) - H_i(t_r)). \tag{8}$$

This implies that the expected number of detected faults in operational phase is accelerated/decelerated by a parameter ϕ from the one in testing environment, because $H_i(t_r + \Delta t) - H_i(t_r)$ means the expected number of faults detected in $[t_r, t_r + \Delta t]$. In [21], the parameter ϕ_i is given by time fraction of execution time of component i in operational phase. Also, they assumed that the number of detected faults after the releases causes maintenance costs to fix the faults. However, in general, the operational environment is quite different from the testing environment. Moreover, from the user perspective, the reliability of software product is more significant than maintenance costs. Thus in this paper, we use the quantitative software reliability in operational phase derived from architecture-based software reliability model.

The architecture-based software reliability model represents a sequence of component executions in operational phase [29]. In most of architecture-based software reliability models, the execution sequence is defined by a discrete or continuous-time Markov chain. In this paper, we focus on the continuous-time Markov chain (CTMC) based model.

The CTMC is a stochastic process with discrete state space on the continuous-time domain. In general, CTMC process $\{M(t); t \geq 0\}$ is characterized by its infinitesimal generator. The infinitesimal generator is a square matrix whose dimension is same as the dimension of state space. The nondiagonal entries of the infinitesimal generator are transition rates between respective states, and diagonal entries represent the exit rates from corresponding states. Let $\boldsymbol{Q}$ be an infinitesimal generator of CTMC process $M(t)$. The probability row vector of $\boldsymbol{\pi}(t) = [P(M(t) = i)]_i$ is given by

$$\frac{d}{dt}\boldsymbol{\pi}(t) = \boldsymbol{\pi}(t)\boldsymbol{Q}. \tag{9}$$

By using the matrix exponential, the probability vector is also given by

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0)\exp(\boldsymbol{Q}t). \tag{10}$$

In particular, we consider two cases: (i) execution of the system has an end; i.e., the system is an application such as command-line application; (ii) execution is continued; i.e., the system courteously provides a service such as server application. For convenience, the first and second cases are discrete and continuous cases, respectively.

(i) Discrete Case. Let $p_{i,j}$ be a transition probability to the execution of component j after finishing the execution of component i. Also $p_{i,S}$ is a probability that the execution is finished after component i. Furthermore, we assume each execution time of component i following an exponential distribution with rate λ_i. Then the sequence of component executions can be described by an absorbing CTMC with infinitesimal generator

$$\boldsymbol{D} = \left(\begin{array}{cccc|c} -\lambda_1 & p_{1,2}\lambda_1 & \cdots & p_{1,n}\lambda_1 & p_{1,S}\lambda_1 \\ p_{2,1}\lambda_2 & -\lambda_2 & \cdots & p_{2,n}\lambda_2 & p_{2,S}\lambda_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ p_{n,1}\lambda_n & p_{n,2}\lambda_n & \cdots & -\lambda_n & p_{n,S}\lambda_n \\ \hline 0 & 0 & \cdots & 0 & 0 \end{array}\right) = \left(\begin{array}{c|c} \boldsymbol{T} & \boldsymbol{\xi} \\ \hline \boldsymbol{0} & 0 \end{array}\right) \tag{11}$$

where $\sum_{j=1}^{n} p_{i,j} + p_{i,S} = 1$ for $i = 1, \ldots, n$, $\boldsymbol{T}$ is a n-by-n matrix for transient states, and $\boldsymbol{\xi}$ is an exit rate vector from transient states to the absorbing state.

To present the failure in operational phase, we define f_i as the failure probability as the execution of component i. In this paper, we suppose that the failure probability is given by

$$f_i = 1 - q_i^{\mathrm{E}[N_i(\infty) - N_i(t_r)]}. \tag{12}$$

In the equation, $\mathrm{E}[N_i(\infty) - N_i(t_r)]$ means the expected number of residual faults in component i at the release time which is given by

$$\mathrm{E}[N_i(\infty) - N_i(t_r)] = a_i - H_i(t_r). \tag{13}$$

Also q_i is the probability that a remaining fault does not cause a failure of component i; i.e., f_i means the probability that at least one remaining fault causes a failure of component i. Then the underlying infinitesimal generator can be rewritten by

$$\boldsymbol{D}_f = \left(\begin{array}{c|cc} \boldsymbol{T}_f & \boldsymbol{\xi}_f & \boldsymbol{f} \\ \hline \boldsymbol{0} & 0 & 0 \\ \boldsymbol{0} & 0 & 0 \end{array}\right), \tag{14}$$

where $\boldsymbol{T}_f$ is the matrix generated by replacing λ_i with $\lambda_i(1-\mathrm{f}_i)$ and $\boldsymbol{f}$ is a column vector whose i-th entry is $\lambda_i f_i$. Note that $\boldsymbol{D}_f$ has two absorbing states corresponding to success and failure of execution, respectively.

The quantitative software reliability is defined by the probability that an execution is successfully finished. From the mathematical argument of CTMC, we have the software reliability in the discrete case:

$$\begin{aligned} R_D &= \boldsymbol{\pi}\int_0^{\infty} \exp\left(\boldsymbol{T}_f t\right)\boldsymbol{\xi}_f dt \\ &= \boldsymbol{\pi}\lim_{u\longrightarrow\infty}\int_0^{u} \exp\left(\boldsymbol{T}_f t\right)\boldsymbol{\xi}_f \\ &= \boldsymbol{\pi}\boldsymbol{T}_f^{-1}\left(\lim_{u\longrightarrow\infty}\exp\left(\boldsymbol{T}_f u\right) - \boldsymbol{I}\right)\boldsymbol{\xi}_f \\ &= \boldsymbol{\pi}\left(-\boldsymbol{T}_f\right)^{-1}\boldsymbol{\xi}_f, \end{aligned} \tag{15}$$

where $\boldsymbol{\pi}$ is a probability vector to decide the initial component of execution.

(ii) Continuous Case. In the continuous case, the sequence of execution can be described by a CTMC with infinitesimal generator:

$$\boldsymbol{C} = \begin{pmatrix} -\lambda_1 & p_{1,2}\lambda_1 & \cdots & p_{1,n}\lambda_1 \\ p_{2,1}\lambda_2 & -\lambda_2 & \cdots & p_{2,n}\lambda_2 \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1}\lambda_n & p_{n,2}\lambda_n & \cdots & -\lambda_n \end{pmatrix} \tag{16}$$

Note that $\sum_{j=1}^{n} p_{i,j} = 1$ for $i = 1, \ldots, n$. Similar to the discrete case, f_i denotes the failure probability at component i. Then we have a CTMC with one absorbing state corresponding to the failure state.

$$\boldsymbol{C}_f = \begin{pmatrix} \boldsymbol{T}_f & \boldsymbol{f} \\ \boldsymbol{0} & 0 \end{pmatrix}. \tag{17}$$

In this case, the software reliability is defined by the probability that the system does not fail during the mission time $[t_r, t_r + t_m]$. From the mathematical argument of CTMC, the quantitative software reliability can be formulated by

$$R_C(t_m) = \boldsymbol{\pi} \exp\left(\boldsymbol{T}_f (t_m - t_r)\right) \mathbf{1}, \tag{18}$$

where $\mathbf{1}$ is a column vector whose entries are 1.

3. Software Testing-Resource Allocation Problems

Based on the models described in Section 2, we formulate the software testing-resource allocation problems. The problem is to decide test efforts for n modules $\omega_1, \ldots, \omega_n$ which minimizes testing cost or maximizes the software reliability in operational phase. Let $\overline{C}$, $\overline{W}$, and $\underline{R}$ be the upper limits of cost and efforts and the lower limits of reliability, respectively. The problems reliability-constrained cost minimization (RCCM) and budget-constrained reliability maximization (BCRM) can be formulated as follows.

(i) RCCM in Discrete Case

$$\begin{aligned} \min_{\omega_1,\ldots,\omega_n} \quad & C(t_r) \\ \text{s.t.} \quad & W(t_r) \le \overline{W}, \\ & R_D \ge \underline{R} \end{aligned} \tag{19}$$

(ii) RCCM in Continuous Case

$$\begin{aligned} \min_{\omega_1,\ldots,\omega_n} \quad & C(t_r) \\ \text{s.t.} \quad & W(t_r) \le \overline{W}, \\ & R_C(t_m) \ge \underline{R} \end{aligned} \tag{20}$$

(iii) BCRM in Discrete Case

$$\begin{aligned} \max_{\omega_1,\ldots,\omega_n} \quad & R_D \\ \text{s.t.} \quad & C(t_r) \le \overline{C}, \\ & W(t_r) \le \overline{W} \end{aligned} \tag{21}$$

(iv) BCRM in Continuous Case

$$\begin{aligned} \min_{\omega_1,\ldots,\omega_n} \quad & R_C(t_m) \\ \text{s.t.} \quad & C(t_r) \le \overline{C}, \\ & W(t_r) \le \overline{W} \end{aligned} \tag{22}$$

They are nonlinear optimization problems and can be solved by numerical approaches such as Nelder-Mead method [31].

4. Numerical Illustration

In this section, we investigate the difference on the optimal testing-resource allocation between Ukimoto et al.'s model and our model. Suppose that the software consists of 10 modules and its architecture (module transition) is given in Figure 1, which is a reference model of architecture model introduced in [25]. The number on each arrow means the transition probability $p_{i,j}$. As seen in the figure, the system has an absorbing state as an output, and thus this is the discrete case. However, to compare our model with Ukimoto et al.'s model, we assume the execution restarts with INPUT just after the execution attains OUTPUT. In such situation, the system becomes the continuous case.

Table 1 shows the expected number of initial faults a_i, the fault detection rate r_i, the fixed effort β_i, and the mean execution time $1/\lambda_i$ used in this example. Also, release time, mission time, fixing cost, and testing cost are set as $t_r = 30.0$, $t_m = 60.0$, $c_1 = 5.0$, and $c_2 = 1.0$, respectively. Moreover, in our model, we set the failure probability per fault as $q_i = 0.99$ for all $i = 1, \ldots, n$.

Ukimoto et al.'s model considers maintenance cost which depends on the expected number of faults detected in operational phase (warranty period). Concretely, when c_3 is the fixing cost per fault in operational phase, the maintenance cost is formulated by

$$C_M(t_m, t_r) = \sum_{i=1}^{n} c_3 \left(H_i(t_m) - H_i(t_r)\right). \tag{23}$$

Note that $H_i(t)$ in the above equation is defined by (7); i.e., it requires the time fraction in execution ϕ_i. In this case, the time fraction is obtained from a steady-state probability of the CTMC. Define the row vector $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_n)$. Then the time fraction can be computed by finding the vector satisfying $\boldsymbol{\phi} = \boldsymbol{\phi}\boldsymbol{C}$ and $\sum_{i=1}^{n} \phi_i = 1$. The last column of Table 1 shows the time fraction of execution. By using the maintenance cost,

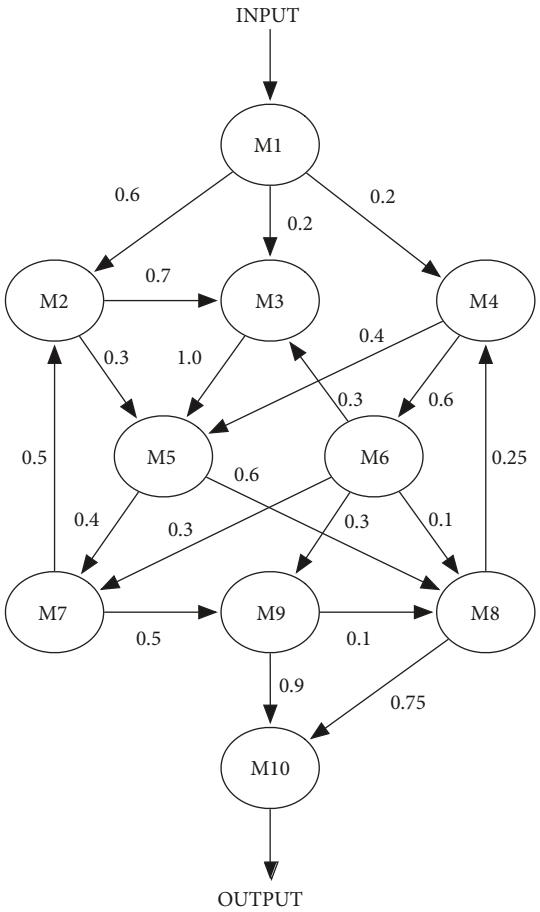

Figure 1: Module architecture.

Table 1: Model parameters.

Module	a_i	r_i	β_i	$1/\lambda_i$	ϕ_i
M1	3.2	0.022	0.0	0.1	0.1297
M2	2.5	0.017	0.0	0.1	0.1177
M3	5.4	0.018	0.0	0.1	0.1181
M4	5.8	0.038	0.0	0.1	0.0543
M5	7.1	0.026	0.0	0.1	0.1751
M6	6.9	0.035	0.0	0.1	0.0326
M7	3.3	0.051	0.0	0.1	0.0798
M8	3.2	0.038	0.0	0.1	0.1133
M9	4.8	0.031	0.0	0.1	0.0497
M10	3.1	0.043	0.0	0.1	0.1297

Table 2: Optimal testing-resource allocation of RCCM.

	Ukimoto		Proposed	
Module	ω_i	Residual	ω_i	Residual
M1	1.388	1.280	6.654	0.040
M2	0.819	1.647	6.258	0.103
M3	2.294	1.564	10.994	0.014
M4	1.800	0.745	4.520	0.034
M5	2.406	1.087	7.266	0.025
M6	2.044	0.807	10.991	0.000
M7	1.166	0.554	8.877	0.000
M8	1.281	0.743	3.738	0.045
M9	1.782	0.915	4.381	0.082
M10	1.202	0.657	4.953	0.005

one of the testing-resource allocation problems described in [21] is given by

$$\begin{aligned} \min_{\omega_1,\ldots,\omega_n} \quad & C(t_r) + C_M(t_m, t_r) \\ \text{s.t.} \quad & W(t_r) \le \overline{W}, \\ & \sum_{i=1}^{n} (a_i - H_i(t_r)) \le \overline{F}. \end{aligned} \tag{24}$$

Note that Ukimoto et al.'s model uses the expected number of residual faults instead of quantitative software reliability. In the experiment, the fixing cost per fault is set as $c_3 = 1000.0$.

Table 2 presents the optimal testing efforts obtained from RCCM problem in both models under $\overline{W} = 5000.0$, $\overline{F} = 10.0$, and $\underline{R} = 0.9$. Also the column 'Residual' indicates the expected number of residual faults at release time. From the table, we find that the optimal testing efforts in our model are much greater than those in Ukimoto et al.'s model. Since much efforts are spent in our model, the expected number of residual faults becomes smaller than those in Ukimoto et al.'s model. The amount of testing efforts depends on the number of initial faults and the detection rate of respective components. For instance, the numbers of initial faults in components M5 and M6 are 7.1 and 6.9 which are relatively higher than others. Thus much effort is spent in these components. Also M5 is the most frequently executed among them in terms of ϕ_i. Therefore, the testing effort for M5 is greatest in Ukimoto et al.'s model. However, in our model, the module with the greatest testing effort is M3. In Figure 1, M3 is the module that is executed before M5. That is, this result is affected by considering detailed transition probabilities of operational profile. On the other hand, Table 3 indicates the minimum costs (testing cost and maintenance cost), total amounts of testing effort (total effort), the total number of residual faults at release time (residual), and the quantitative software reliability in the operational phase (reliability). From the table, in Ukimoto et al.'s model, residual attains to the upper limit $\overline{F}$, and reliability attains to the lower limit $\underline{R}$ in our case. Also, in the result on testing cost, there is a remarkable difference between Ukimoto et al.'s model and our model. The strategy obtained from Ukimoto et al.'s model is that much cost is spent to the maintenance without thought of quality (reliability) of software product. On the other hand, the strategy of our model is that much cost is spent in testing phase to guarantee the quality of software.

Next we show the example of BCRM. In BCRM, we set $\overline{W} = 3000.0$ and $\overline{C} = 3000$. Note that the upper limit of cost $\overline{C}$ is for the development cost, which does not include the maintenance cost. Tables 4 and 5 present the optimal testing efforts and their associated criteria. Dissimilar to RCCM, Ukimoto et al.'s model provides the high reliability. In this

Table 3: Cost, effort, and reliability under the optimal testing-resource allocation of RCCM.

	Ukimoto	Proposed
Testing cost	661.97	2283.75
Maintenance cost	1251.83	—
Total effort	485.47	2058.98
Residual	10.00	0.35
Reliability	0.04	0.90

Table 4: Optimal testing-resource allocation of BCRM.

	Ukimoto		Proposed	
Module	ω_i	Residual	ω_i	Residual
M1	11.456	0.0017	11.574	0.0015
M2	11.900	0.0058	13.871	0.0021
M3	12.129	0.0077	14.313	0.0024
M4	6.003	0.0062	6.492	0.0035
M5	9.459	0.0044	10.526	0.0019
M6	14.434	0.0000	8.035	0.0015
M7	4.743	0.0023	4.980	0.0016
M8	8.787	0.0001	6.483	0.0020
M9	7.961	0.0029	9.418	0.0008
M10	4.978	0.0050	6.453	0.0008

Table 5: Cost, effort, and reliability under the optimal testing-resource allocation of BCRM.

	Ukimoto	Proposed
Testing cost	2981.819	2990.759
Maintenance cost	0	—
Total effort	2755.5	2764.35
Residual	0.036	0.018
Reliability	0.9882	0.9946

example, since the upper limit of cost is enough, both models provide the high reliability. However, the effort allocation is slightly different between them.

5. Conclusion

In this paper, we have presented testing-resource allocation problems by considering software reliability in operational phase. Concretely, by using architecture-based software reliability model, we have formulated the quantitative software reliability in operational phase and they are incorporated into the optimization problems to determine the optimal testing-resource allocation. In the numerical example, we have compared the optimal testing-resource allocation in Ukimoto et al.'s model and our model. As a result, the decision derived from our model is more severe to the quality of software product, compared to the decision from Ukimoto et al.'s model. In other words, from the reliability point of view, Ukimoto et al.'s model involves the risk that the released software fails, and the reliability of released software might be lower than the reliability we expect. The safety and mission critical systems require the high reliability. For such systems, the strict evaluation of operational reliability based on the software architecture is needed.

In future, we will investigate the tendency of BCRM problem in our model by compared with existing problems. Furthermore, by combining empirical software reliability engineering [32–34], we will discuss how to determine the model parameters in testing-resource allocation problems.

Data Availability

The model parameters in the experiment have been shown in the paper.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] H. Ohtera and S. Yamada, "Optimal allocation and control problems for software testing-resources," *IEEE Transactions on Reliability*, vol. R-39, no. 2, pp. 171–176, 1990.

[2] D. W. Coit, "Economic allocation of test times for subsystem-level reliability growth testing," *Institute of Industrial Engineers (IIE). IIE Transactions*, vol. 30, no. 12, pp. 1143–1151, 1998.

[3] F. Zahedi and N. Ashrafi, "Software reliability allocation based on structure, utility, price, and cost," *IEEE Transactions on Software Engineering*, vol. 17, no. 4, pp. 345–356, 1991.

[4] N. Ashrafi and O. Berman, "Optimal design of large software systems considering reliability and cost," *IEEE Transactions on Reliability*, vol. 41, no. 2, pp. 281–287, 1992.

[5] O. Berman and N. Ashrafi, "Optimization models for reliability of modular software systems," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1119–1123, 1993.

[6] S. Yamada, T. Ichimori, and M. Nishiwaki, "Optimal allocation policies for testing-resource based on a software reliability growth model," *Mathematical and Computer Modelling*, vol. 22, no. 10-12, pp. 295–301, 1995.

[7] M. Nishiwaki, S. Yamada, and T. Ichimori, "Testing-resource allocation policies based on an optimal software release problem," *Mathematica Japonica*, vol. 43, no. 1, pp. 91–97, 1996.

[8] F. Belli and P. Jedrzejowicz, "An approach to the reliability optimization of software with redundancy," *IEEE Transactions on Software Engineering*, vol. 17, no. 3, pp. 310–312, 1991.

[9] Y.-W. Leung, "Optimal reliability allocation for modular software system designed for multiple customers," *IEICE Transaction on Information and Systems*, vol. E79-D, no. 12, pp. 1655–1662, 1996.

[10] Y.-W. Leung, "Dynamic Resource-allocation for Software-module Testing," *The Journal of Systems and Software*, vol. 37, no. 2, pp. 129–139, 1997.

[11] Y.-W. Leung, "Software reliability allocation under an uncertain operational profile," *Journal of the Operational Research Society*, vol. 48, no. 4, pp. 401–411, 1997.

[12] R.-H. Hou, S.-Y. Kuo, and Y.-P. Chang, "Needed resources for software module test, using the hyper-geometric software reliability growth model," *IEEE Transactions on Reliability*, vol. 45, no. 4, pp. 541–549, 1996.

[13] P. C. Jha, D. Gupta, B. Yang, and P. K. Kapur, "Optimal testing resource allocation during module testing considering

cost, testing effort and reliability," *Computers & Industrial Engineering*, vol. 57, no. 3, pp. 1122–1130, 2009.

[14] S. A. Wadekar and S. S. Gokhale, "Exploring cost and reliability tradeoffs in architectural alternatives using a genetic algorithm," in *Proceedings of the 10th IEEE International Symposium on Software Reliability Engineering (ISSRE-1999)*, pp. 104–113, IEEE, Boca Raton, FL, USA, 1999.

[15] M. R. Lyu, S. Rangarajan, and A. P. A. van Moorsel, "Optimal allocation of test resources for software reliability growth modeling in software development," *IEEE Transactions on Reliability*, vol. 51, no. 2, pp. 183–192, 2002.

[16] B. Yang and M. Xie, "Optimal testing-time allocation for mudular systems," *International Journal of Quality and Reliability Management*, vol. 18, no. 8, pp. 854–863, 2001.

[17] M. E. Helander, M. Zhao, and N. Ohlsson, "Planning models for software reliability and cost," *IEEE Transactions on Software Engineering*, vol. 24, no. 6, pp. 420–434, 1998.

[18] A. Ngo-The and G. Ruhe, "Optimized resource allocation for software release planning," *IEEE Transactions on Software Engineering*, vol. 35, no. 1, pp. 109–123, 2009.

[19] R. Pietrantuono, S. Russo, and K. S. Trivedi, "Software reliability and testing time allocation: An architecture-based approach," *IEEE Transactions on Software Engineering*, vol. 36, no. 3, pp. 323–337, 2010.

[20] J. D. Musa, "Operational profiles in software-reliability engineering," *IEEE Software*, vol. 10, no. 2, pp. 14–32, 1993.

[21] S. Ukimoto, T. Dohi, and H. Okamura, "Software testing-resource allocation with operational profile," in *Proceedings of the 27th ACM Symposium on Applied Computing (SAC-2012)*, pp. 1203–1208, Trento, Italy, March 2012.

[22] B. Littlewood, "A reliability model for systems with Markov structure," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 24, no. 2, pp. 172–177, 1975.

[23] B. Littlewood, "Software reliability model for modular program structure," *IEEE Transactions on Reliability*, vol. R-28, no. 3, pp. 241–246, 1979.

[24] J.-C. Laprie, "Dependability evaluation of software systems in operation," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 6, pp. 701–714, 1984.

[25] R. C. Cheung, "A user-oriented software reliability model," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 2, pp. 118–125, 1980.

[26] J. Ledoux and G. Rubino, "A counting model for software reliability analysis," *International Journal of Modelling and Simulation*, vol. 17, no. 4, pp. 289–297, 1997.

[27] J. Ledoux, "Availability modeling of modular software," *IEEE Transactions on Reliability*, vol. 48, no. 2, pp. 159–168, 1999.

[28] K. Goševa-Popstojanova, A. P. Mathur, and K. S. Trivedi, "Comparison of architecture-based software reliability models," in *Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE'01)*, pp. 22–31, IEEE, 2001.

[29] K. Goševa-Popstojanova and K. S. Trivedi, "Architecture-based approach to reliability assessment of software systems," *Performance Evaluation*, vol. 45, no. 2-3, pp. 179–204, 2001.

[30] H. Singh, V. Cortellessa, B. Cukic, E. Gunel, and V. Bharadwaj, "A bayesian approach to reliability prediction and assessment of component based systems," in *Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE'01)*, pp. 12–21, IEEE, China, November 2001.

[31] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[32] K. Shibata, K. Rinsaka, and T. Dohi, "Metrics-based software reliability models using non-homogeneous poisson processes," in *Proceedings of the 2006 17th International Symposium on Software Reliability Engineering*, pp. 52–61, IEEE, Raleigh, NC, USA, November 2006.

[33] H. Okamura, Y. Etani, and T. Dohi, "A multi-factor software reliability model based on logistic regression," in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering (ISSRE '10)*, pp. 31–40, IEEE, San Jose, CA, USA, November 2010.

[34] H. Okamura and T. Dohi, "A novel framework of software reliability evaluation with software reliability growth models and software metrics," in *Proceedings of the 15th IEEE International Symposium on High Assurance Systems Engineering (HASE '14)*, pp. 97–104, IEEE, Miami Beach, FL, USA, January 2014.