Optimal Allocation of Testing-Resource Considering Cost, Reliability, and Testing-Effort

Chin-Yu Huang[1], Jung-Hua Lo[2], Sy-Yen Kuo[3], and Michael R. Lyu[4]

[1]*Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan*
[2]*Department of Information Management, Lan Yang Institute of Technology, I-Land, Taiwan*
[3]*Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan*
[4]*Computer Science & Engineering Department, The Chinese University of Hong Kong, Shatin, Hong Kong*

Abstract

We investigate an optimal resource allocation problem in modular software systems during testing phase. The main purpose is to minimize the cost of software development when the number of remaining faults and a desired reliability objective are given. An elaborated optimization algorithm based on the Lagrange multiplier method is proposed and numerical examples are illustrated. Besides, sensitivity analysis is also conducted. We analyze the sensitivity of parameters of proposed software reliability growth models and show the results in detail. In addition, we present the impact on the resource allocation problem if some parameters are either overestimated or underestimated. We can evaluate the optimal resource allocation problems for various conditions by examining the behavior of the parameters with the most significant influence. The experimental results greatly help us to identify the contributions of each selected parameter and its weight. The proposed algorithm and method can facilitate the allocation of limited testing-resource efficiently and thus the desired reliability objective during software module testing can be better achieved.

1. Introduction

The size and complexity of computer systems have grown rapidly for the last several decades. Software costs as a percentage of total computer system costs continue to increase; while associated hardware costs continue to decrease. The quantitative assessment of software quality can be conducted through many approaches; however, it is sometimes difficult for the project managers to measure software quality and productivity. Nevertheless, reliability may be the most important quality attribute of commercial software since it quantifies software failures during the development process. Although we can test maintainability, usability, or efficiency, but the key issue for software testing is still reliability. Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment [1]. Its evaluation includes two types of activities: reliability estimation and reliability prediction. Since the early 1970s, many analytical software reliability growth models (SRGMs) have been proposed for estimation of reliability growth of products during software development processes. There are two main categories of reliability estimation models: SRGMs and statistical models. The models in the former class can estimate the software reliability using the failure history of the program. On the other hand, the latter models apply the success/failure information of a program from a random sample of test cases without making any corrections on the discovered errors [2-3].

Most SRGMs are typically based on failure data such as number of failures, time of occurrence, failure severity, or the interval between two consecutive failures, whereas other models describe the relationship among the calendar testing, the amount of testing-effort, and the number of software faults detected by testing. The testing-effort can be represented as the number of CPU hours, the number of executed test cases, etc [4-6]. SRGMs sometimes show good performance in terms of predictability of the software reliability, but sometimes they do not. This fact may be caused by insufficient information on how the software has been developed, maintained, and operated.

Furthermore, many SRGMs neglect cost. Some software cost models consider reliability as one of the factors affecting cost [7]. For example, the well-known COCOMO model takes reliability as one of its fifteen cost drivers [8]. Musa et al. [3] also discuss a model for determining the minimal life cycle cost of software, in which they assumed that testing cost is a nonlinear function of software failure rate. Similarly, some papers provide optimal software release policies and include reliability in the cost function. Kubat formulates a mathematical programming model to determine module reliabilities by minimizing software development costs [9-10]. Berman et al. also propose an optimization model for deriving cost allocations while satisfying a budget constraint [11]. Morevoer, cost analysis can be performed by multiplying the difference in expected total number of defects by either a relative or a fixed cost parameter. Following Okumoto and Goel, and Yamada et al., we can evaluate the total software testing cost by using the cost of testing-effort expenditures during software testing phase [3, 12-13].

Practically, a software testing process consists of several testing stages including module testing, integration testing, system testing and installation testing. During the testing phase, software faults can be detected and removed. The quality of the tests usually corresponds to the maturity of the software test process, which in turn relates to the maturity of the overall software development process. In general, most popular and commercial software products are complex systems composed of a number of modules. Typically, module testing is the most time-critical part of testing to be performed. All the testing activities of different modules should be completed within a limited time, and these activities normally consume approximately 40%~50% of the total amount of software development resources. Therefore, project managers should know how to allocate the specified testing-resources among all the modules and develop quality software with high reliability. Many recent papers have addressed the problem of optimal resource allocation [9-29]. For example, the reliability allocation approach of Leung [14, 17-19] used the operational profile to define a software utility function. Kubat presented a stochastic model to minimize cost subject to an overall system failure intensity goal [9-10, 14]. He took an implicit usage view of the system by modeling transitions through modules according to a Markov process, which is similar in concept to the modular software reliability model proposed by Littlewood [28]. Besides, Hou et al. investigated software release policies to minimize testing cost while satisfying a system reliability objective. They considered minimizing the number of undetected software faults under a budget constraint, as well as minimizing testing resources constrained by undetected faults [14, 20-21]. The purpose of these research efforts is to allocate testing-resources efficiency to testing activities so that the reliability of software systems will be maximized or the remaining faults can be minimized.

In this paper, we will show how to minimize the cost of software, given the number of remaining faults and a desired reliability objective. We provide a systematic method for the software project managers to allocate specific amount of testing-resource expenditures for each module under given constraints. An SRGM with generalized *logistic* testing-effort function to describe the time-dependency behaviors of detected software faults is used. The paper is organized as follows. In Section 2, an SRGM with generalized *logistic* testing-effort function based on NHPPs is presented. The derivation of an optimal testing-resource allocation problem for modular software testing is developed in Section 3. We investigate the optimization problem of minimizing the software development cost with a given fixed amount of testing-effort and a reliability objective. Furthermore, several numerical examples are described and a sensitivity analysis is illustrated in Sections 4. We can evaluate the optimal resource allocation problems for various conditions by examining about the behavior of some parameters with the most significant influence. Finally, the conclusions are drawn in Section 5.

2. Reviews of SRGM with generalized logistic testing-effort function

A number of SRGMs have been proposed on the subject of software reliability. Among these models, Goel and Okumoto used an NHPP as the stochastic process to describe the fault process [1]. Yamada et al. [6-8] modify the G-O model and incorporate the concept of testing-effort in an NHPP model to get a better description of the software fault detection phenomenon. We also propose a new SRGM with the logistic testing-effort function to predict the behavior of failure occurrences and the fault content of a software product. Based on our past experimental results, this approach is suitable for estimating the reliability of software application during the development process [6-9, 19-23]. Here are the modeling assumptions:

(1). The fault removal process is modeled by an NHPP.
(2). The software application is subject to failures at random times caused by the remaining faults in the system.
(3). The mean number of faults detected in the time interval $(t, t+\Delta t)$ by the current testing-effort is proportional to the mean number of remaining faults in the system at time t, and the proportionality is a constant over time.
(4). Testing effort expenditures are described by a *generalized logistic* testing-effort function.
(5). Each time a failure occurs, the corresponding fault is immediately removed and no new faults are introduced.
(6). The hazard rate for software occurring initially after the testing is proportional to the elapsed time τ and the remaining faults.

With these assumptions, if the number of faults detected by the current testing-effort expenditures is proportional to the number of remaining faults, then we obtain the following differential equation:

$$\frac{dm(t)}{dt} \times \frac{1}{w_{\kappa}(t)} = r \times [a - m(t)] \qquad (1)$$

where $m(t)$ is the expected mean number of faults detected in time $(0, t)$, $W\kappa(t)$ is the current testing-effort consumption at time t, a is the expected number of initial faults, and r is the fault detection rate per unit testing-effort at testing time t and $r>0$.

Solving Eq. (1) under the boundary condition $m(0)=0$ (i.e., the mean value function $m(t)$ is equal to zero at time 0), we have

$$\begin{aligned} m(t) &= a(1-\exp[-r(W_\kappa(t)-W_\kappa(0))]) \\ &= a(1-\exp[-r(W(t))]) \end{aligned} \qquad (2)$$

In Eq. (2), $m(t)$ is non-decreasing with respect to testing time t. Knowing its value can help us determine whether the software is ready for release and if not, how much more testing resources are required [1, 6]. It can provide an estimate of the number of failures that will eventually be encountered by the customers. When $t\to\infty$, the expected number of faults to be detected is

$$m(\infty) = a \times \left(1-\exp[\frac{N}{1+A}]\right) \cong a \quad \text{(if } A >> N\text{)} \qquad (3)$$

Besides, a *generalized logistic* testing-effort function with structuring index is proposed, which can be used to consider and evaluate the effects of possible improvements on software development methodology, such as top-down design or stepwise refinement [4, 6, 25]. The *generalized logistic* testing-effort function is depicted as follows:

$$W_\kappa(t) = N \times \left(\frac{(\kappa+1)/\beta}{1+Ae^{-\alpha\kappa t}}\right)^{1/\kappa} \qquad (4)$$

where N is the total amount of testing effort to be consumed, α is the consumption rate of testing-effort expenditures, A is a constant, and κ is a structuring index, whereas a large value is used for modeling well-structured software development efforts.

In addition, given that the testing has continued up to time t, the probability that a software failure does not occur in the time interval $(t, t+\Delta t)$ $(\Delta t \geq 0)$ is given by

$$R(t) \equiv R(t+\Delta t \mid t) = \exp[-(m(t+\Delta t)-m(t))] \qquad (5)$$

Taking the logarithm on both sides of the above equation, we obtain

$$\ln R(t) = -(m(t+\Delta t)-m(t)) \qquad (6)$$

From the Eq. (6) and Eq. (1) we can determine the testing time needed to reach a desired reliability R_0 [6]. On the other hand, from assumption (6), we can also obtain software reliability [23]

$$R(\tau) = \exp\left[-r \times a \times \exp[-rW(t)] \times \tau\right] \qquad (7)$$

That is, Eq. (7) means that no software failure occurs during the time interval $(0, \tau]$ after the testing and it is seldom used except in some papers [8, 16]. Therefore, we define another measure of software reliability, i.e., the ratio of the cumulative number of detected faults at time t to the expected number of initial faults [4, 21-22].

$$R(t) \equiv \frac{m(t)}{a} \qquad (8)$$

We can solve Eq. (8) and obtain a unique t satisfying $R(t)=R_0$. Note that $R(t)$ is an increasing function in t. Using $R(t)$, we can easily get the required testing time needed to reach the reliability objective R_0 or decide whether the reliability objective can be satisfied at a specified time. If we know that the reliability of a software system has achieved an acceptable reliability level, then we can determine the right time to release this software.

3. Testing-resource allocation policies for module testing

In this section, we will consider resource allocation problems based on an SRGM with *generalized logistic* testing-effort function during software testing phase.

3.1. Model's assumptions and descriptions

Assumptions [6, 25]:

(1). The software system is composed of N independent modules that are tested individually. The number of software faults remaining in each module can be estimated by an SRGM with *generalized logistic* testing-effort function.

(2). For each module, the failure data have been collected and the parameters of each module can be estimated.

(3). The total amount of testing resource expenditures available for the module testing processes is fixed and denoted by W.

(4). If any of the software modules fails upon execution, the whole software system is in failure.

(5). The system manager has to allocate the total testing resources W to each software module and minimize the number of faults remaining in the system during the testing period. Besides, the desired software reliability after the testing phase should achieve the reliability objective R_0.

From Section 2, the mean value function of a software system with N modules can be formulated as:

$$M(t) = \sum_{i=1}^{N} v_i m_i(t) = \sum_{i=1}^{N} v_i a_i (1-\exp(-r_i W_i(t))) \qquad (9)$$

where v_i is a weighting factor to measure the relative importance of a fault removal from module i in the future. If $v_i=1$ for all i=1, 2,…, N, the objective is to minimize the total number of faults remaining in the software system after the testing phase. This indicates that the number of remaining faults in the system can be estimated by

$$\sum_{i=1}^{N} v_i a_i \exp(-r_i W_i(t)) \equiv \sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) \qquad (10)$$

3.2. Minimizing the software cost with a given fixed amount of testing-effort and a reliability objective

In this subsection, we should allocate an amount of testing-effort to each software module to minimize the software testing cost. In general, testing might stop when a 90% upper confidence bound on the number of remaining faults is below a desired bound. Alternatively, testing could stop when total lifecycle cost is minimized. The cost of a failure is greater in the field than in system test. Therefore, the marginal benefit of testing for an increment of execution time is the expected decrease in the cost of field failures, accounting for the expected number of failures in that increment. The marginal cost of testing is the resources needed to test for an increment of execution time. To minimize the total cost, testing should proceed until the marginal benefit falls below the marginal cost [30]. There are some cost models published in the literature, such as Putnam's SLIM cost model, Checkpoint, Boehm's COCOMO model, RCA PRICE S model, or COCOMO'II, ..., etc [8].

Actually, software cost analysis can also be performed by multiplying the difference in expected total number of defects by either a relative or a fixed cost parameter. Kubat formulate a mathematical programming model which, for a given level of software reliability, determines module reliabilities by minimizing development and testing costs [9-10]. In his model the reliability of a program is the multiplication of the reliability of its modules, and the reliability of the system is a weighted sum of the reliability of its programs. The cost of each module is assumed to be a linear function of its reliability. The goal of the model is to find the reliability of each module so that the reliability of the system will be maximized within a given budget [7, 30]. We can evaluate the total software cost by using cost criterion, the cost of testing-effort expenditures during software development & testing phase, and the cost of correcting errors before and after release as follows [3-4, 12-13, 23]:

$$C(t) = C'_1\, m(t) + C'_2\,[m(\infty) - m(t)] + C'_3 \times \int_0^t w(x)dx \qquad (11)$$

where C'_1 is the cost of correcting an error during testing, C'_2 is the cost of correcting an error in operational use ($C'_2 > C'_1$), and C'_3 is the cost of testing per unit testing-effort expenditure. If we use Eq. (2), (3), (9), and (10) to substitute $m(t)$ in Eq. (11), we can develop a software cost model based on an NHPP (Non-homogeneous Poisson process) model and the total cost can be computed as follows:

$$\sum_{i=1}^{N} Cost_i(W_i) = C^*_1 \sum_{i=1}^{N} v_i a_i (1-\exp(-r_i W_i)) + C^*_2 \sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) + C^*_3 \times \sum_{i=1}^{N} W_i \qquad (12)$$

where C^*_1 is the cost of correcting an error during module testing, C^*_2 is the cost of correcting an undetected error during module testing, and C^*_3 is the cost of module testing per unit testing-effort expenditures. We know that $C^*_2 > C^*_1$ as C^*_2 is usually an order of magnitude greater than C^*_1 [8].

Altogether, based on labor, overhead, and related expenses, we can determine the cost per failure C^*_1 for failures that occur during module test, as well as the cost per unit testing-effort expenditures C^*_3. Besides, based on program maintenance, service impact, and related expenses, we can determine the cost per failure C^*_2 for failures that occur during the operational use [8, 30].

From Eq. (12), we can know the relationship between the total cost for each software module and the testing-resource expenditures. Therefore, the optimization problem is how to allocate testing efforts to each module, given that the total amount of testing-effort is fixed, and a reliability objective is set. Suppose the total amount of testing-effort is W, and module i is allocated W_i testing-effort, then the optimization problem can be represented as follows:

The objective function is:

$$Minimize: \quad \sum_{i=1}^{N} Cost_i(W_i) \qquad (13)$$

Subject to the constrains:

$$\sum_{i=1}^{N} W_i \le W, \quad W_i \ge 0, \; i=1, 2,..., N. \qquad (14)$$

$$R_i(t) = 1 - \exp(-r_i W_i(t)) \ge R_0 \qquad (15)$$

From Eq. (15), we can obtain

$$W_i \ge \frac{-1}{r_i}\ln(1-R_0), \quad i = 1,2,...,N.$$

Let $$D_i \equiv \frac{-1}{r_i}\ln(1-R_0), \quad i = 1,2,...,N. \qquad (16)$$

Thus, we can have

$$\sum_{i=1}^{N} W_i \le W, \quad W_i \ge 0, i = 1,2,...,N \quad \text{and} \quad W_i \ge C_i,$$

where $C_i = \max(0, D_1, D_2, D_3, ..., D_N)$

Let $X_i = W_i - C_i$, we can transform above equations to:

$$\textbf{Minimize}: \quad \sum_{i=1}^{N} Cost^*_i\,(X_i) \qquad (17)$$

$$\textbf{Subject to} \quad \sum_{i=1}^{N} X_i \le W - \sum_{i=1}^{N} C_i, \quad X_i \ge 0, \; i=1,2, ..., N. \qquad (18)$$

where $$\sum_{i=1}^{N} Cost^*_i\,(X_i) = C^*_1 \times \sum_{i=1}^{N} v_i a_i (1 - \exp(-r_i C_i) \times \exp(-r_i X_i)) + C^*_2 \times \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i) + C^*_3 \times \sum_{i=1}^{N} (X_i + C_i) \qquad (19)$$

Note that the parameters v_i, a_i, and r_i should already be estimated by the proposed model. To solve the above problem, the Lagrange multiplier method can be applied. As we all know the conditions of Kuhn-Tucker are the most important theoretical results in the field of nonlinear programming. They must be satisfied at any constrained optimum, local or global, of any linear and most nonlinear programming problems [31-32]. Consequently, associating multiplier λ with Eq. (18), the above equations can be simplified as follows:

Minimize:

$$L(X_1, X_2, ..., X_N, \lambda) = C^*_1 \sum_{i=1}^{N} v_i a_i (1 - \exp(-r_i C_i)\exp(-r_i X_i))$$
$$+ C^*_2 \times \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i) \times \exp(-r_i X_i)$$
$$+ C^*_3 \times \sum_{i=1}^{N} (X_i + C_i) + \lambda(\sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i) \quad (20)$$

Based on the Kuhn-Tucker conditions (KTC), the necessary conditions for a minimum value of Eq. (20) are in existence and can be stated as follows [12-13, 16, 31-32]:

A1: $\frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial X_i} = 0$, i=1, 2,..., N. (21)

A2: $X_i \frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial X_i} = 0$, i=1, 2,..., N. (22)

A3: $\lambda \times \{\sum_{i=1}^{N} X_i - (W - \sum_{i=1}^{N} C_i)\} = 0$, i=1, 2,..., N. (23)

Theorem 1. A feasible solution X_i (i=1, 2,..., N) of Eq. (20) is optimal if and only if

(1) $\lambda \geq v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) \times \exp(-r_i X_i) - C^*_3$

(2) $X_i \times \{\lambda + C^*_3 - v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) \times \exp(-r_i X_i)\} = 0$

Proof: The proof is omitted since it is quite straightforward.

Corollary 1. Let X_i be a feasible solution of Eq. (20)

(i) X_i=0 if and only if
$\lambda \geq v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) - C^*_3$

(ii) If X_i>0, then
$X_i = \{\ln(v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i)) - \ln(\lambda + C^*_3)\}/r_i$

Proof:

(i) If X_i=0, then Theorem 1 implies that
$$\lambda \geq v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) - C^*_3$$
That is, if $\lambda = v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) - C^*_3$, then from Theorem 1 (2) we know that
$X_i \times \{v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) - v_i a_i r_i (C^*_2 - C^*_1) \times \exp(-r_i C_i) \times \exp(-r_i X_i)\} = 0$ or $X_i \times v_i a_i r_i (C^*_2 - C^*_1) \times \exp(-r_i C_i) \times \{1 - \exp(-r_i X_i)\} = 0$.

Since $v_i \neq 0$, $a_i \neq 0$, $r_i \neq 0$, and $\exp(-r_i C_i) \neq 0$, therefore we have X_i=0 or $1 - \exp(-r_i X_i) = 0$. That is, X_i=0. On the other hand, if $\lambda > v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) - C^*_3$, then $\lambda > v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) \geq v_i a_i r_i (C^*_2 - C^*_1) \times \exp(-r_i C_i) \times \exp(-r_i X_i)$ or $v_i a_i r_i (C^*_2 - C^*_1) \times \exp(-r_i C_i) - v_i a_i r_i (C^*_2 - C^*_1) \times \exp(-r_i C_i) \times \exp(-r_i X_i) \neq 0$. Therefore, from Theorem 1 (2), we have X_i=0. Q.E.D.

(ii) From Theorem 1 (2), we know that if X_i>0,
$\lambda = v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) \times \exp(-r_i X_i) - C^*_3$.
Therefore
$X_i = \{\ln(v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i)) - \ln(\lambda + C^*_3)\}/r_i$
and $0 \leq \lambda < v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i) - C^*_3$. Q.E.D.

From Eq. (20), we have

$$\frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial X_i} = C^*_1 v_i a_i r_i \exp(-r_i C_i)\exp(-r_i X_i) - C^*_2 v_i a_i r_i \exp(-r_i C_i) \times \exp(-r_i X_i) + C^*_3 + \lambda$$
$$= -(C^*_2 - C^*_1) v_i a_i r_i \times \exp(-r_i C_i)\exp(-r_i X_i) + C^*_3 + \lambda = 0 \quad (24)$$
$$\frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial \lambda} = \sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i = 0$$

Thus, the solution X_i^0 is

$$X_i^0 = \{\ln(v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i)) - \ln(\lambda^0 + C^*_3)\}/r_i, \quad i=1, 2, ..., N. \quad (25)$$

The solution λ^0 is

$$\lambda^0 = -C^*_3 + \exp\left[\frac{\sum_{i=1}^{N}(1/r_i)(\ln v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i)) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N}(1/r_i)}\right] \quad (26)$$

Hence, we get $X^0 = (X_1^0, X_2^0, X_3^0, ..., X_N^0)$ as an optimal solution to Eq. (20). However, the above X^0 may have some negative components if $v_i a_i r_i (C^*_2 - C^*_1) \times \exp(-r_i C_i) < \lambda^0 + C^*_3$, making X^0 infeasible for Eq. (17) and Eq. (18). If this is the case, the solution X^0 can be corrected by the following steps.

Algorithm 1

Step 1: Set l=0.

Step 2: Calculate the following equations

$$X_i = \frac{1}{r_i}\{\ln(v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i)) - \ln(\lambda + C^*_3)\}$$
i=1, 2,..., $N- l$.

$$\lambda = -C^*_3 + \exp\left[\frac{\sum_{i=1}^{N}(1/r_i)(\ln v_i a_i r_i (C^*_2 - C^*_1)\exp(-r_i C_i)) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N}(1/r_i)}\right]$$

Step 3: Rearrange the index i such that
$$X_1^* \geq X_2^* \geq ... \geq X_{N-l}^*.$$

Step 4: IF $X_{N-l}^* \geq 0$ then
stop (i.e., the solution is optimal)
Else

```
        $X^*_{N-l} = 0$; $l$=$l$+1
    End-IF.
```
Step 5: Go to **Step 2.**

The optimal solution has the following form:

$$\begin{cases} X_i^* = \dfrac{1}{r_i}\{\ln(v_i a_i r_i (C*_2 - C*_1)\exp(-r_i C_i)) - \ln(\lambda + C*_3)\} \\ \qquad\qquad\qquad\qquad\qquad\qquad , i = 1,2,\ldots,N-l. \\ where\ \lambda = -C*_3 + \\ \exp\left[\dfrac{\sum_{i=1}^{N}(1/r_i)(\ln v_i a_i r_i (C*_2 - C*_1)\exp(-r_i C_i)) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N}(1/r_i)}\right] \\ X_i^* = 0,\ otherwise \end{cases} \tag{27}$$

It is noted that Algorithm 1 converges in, at worst, N–1 steps. Thus the value of objective function given by Eq. (20) at the optimal solution $(X_1^*, X_2^*, \ldots, X_N^*)$ is

$$C*_1 \sum_{i=1}^{N} v_i a_i (1 - \exp(-r_i C_i)\exp(-r_i X_i^*)) + C*_2 \times \\ \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i^*) + C*_3 \sum_{i=1}^{N}(X_1^* + C_i) \tag{28}$$

4. Numerical illustration

4.1. Numerical examples

In this section, we assume that the estimated parameters a_i, r_i, and κ in Eq. (9), for a software system consisting of 10 modules, are summarized in Table 1. All the parameters a_i and r_i for each software module were estimated by using the *maximum likelihood estimation* (MLE) or the *least squares estimation* (LSE). We apply the proposed model to software failure data set [12-13, 15, 21, 26]. Here we have to allocate the expenditures to each module and minimize the number of remaining faults. Besides, we let the cost parameters $C*_1$ =2, $C*_2$ =10, and $C*_3$=0.5. Moreover, the weighting vectors v_i in Eq. (9) are also listed in Table 1. We need to allocate the expenditures to each module and minimize the expected cost of software during module testing. In the following, we illustrate one example to show how the optimal allocation of testing-effort expenditures to each software module is determined.

Suppose that the total amount of testing-effort expenditures W is 50,000 man-hours and R_0=0.9. From Table 1 and Algorithm 1 in Section 3.2, the optimal testing-effort expenditures for the software systems are also estimated and shown in Table 1. It is noted that the weighting vectors of module 9 is 0.05. Through Eq. (12) and Table 1, it is easy to obtain the total expected software cost. Conversely, if for some reasons and specific requirements we intend to decrease more software cost, we have to re-plan and re-consider the allocation of testing-resource expenditures; i.e., using the same values of a_i, r_i, and v_i, the optimal testing-effort expenditures should be re-estimated. Following the same procedure described in Section 3.2, we can still easily find out how much amount of testing-resource expenditures is expected.

Table 1: The optimal testing-effort expenditures with estimated values of a_i, r_i, v_i, and κ

Module	a_i	r_i	κ	v_i	X_i^*
1	89	4.1823×10^{-4}	1	1.0	7632
2	25	5.0923×10^{-4}	1	0.6	3158
3	27	3.9611×10^{-4}	1	0.7	4009
4	45	2.2956×10^{-4}	1	0.4	4329
5	39	2.5336×10^{-4}	1	1.5	8964
6	39	1.7246×10^{-4}	1	0.5	4568
7	59	8.819×10^{-5}	1	0.5	6023
8	68	7.274×10^{-5}	1	0.6	9112
9	37	6.824×10^{-5}	1	0.05	0
10	14	1.5309×10^{-4}	1	1	2203

4.2. Sensitivity analysis

In this section, a sensitivity analysis of the proposed model is conducted to study the effect of the principal parameters, such as the expected initial faults and the fault detection rate. In Eq. (1), we know that there are some parameters affecting the mean value function, such as the expected total number of initial faults, fault detection rate, the total amount of testing-effort, the consumption rate of testing-effort expenditures, or the structuring index, etc. Therefore, we need to estimate all these parameters for each software module very carefully since they play important roles for these optimal resource allocation problems. In general, each parameter is estimated based on the available data, which is often sparse. Therefore, in this section, we analyze the sensitivity of some principal parameters but not for all parameters due to the limitation of size. Nevertheless, we still can evaluate the optimal resource allocation problems for various conditions by examining about the behavior of some parameters that have the most significant influence [33-35]. We thus perform the sensitivity of optimal resource allocation problems with respect to the estimated parameters so that attention can be paid to those parameters deemed most critical. In the following paragraph, we will give some numerical examples to understand the sensitivity of optimal resource allocation problems with respect to the estimated parameters.

4.2.1. Effect of variations on expected initial faults. From Table 1, we can know that the optimal testing-effort expenditures (OTEE) with estimated values of a_i, r_i, v_i , and κ under $C_1^* = 2, C_2^* = 10, C_3^* = 0.5$, and W=50,000. We investigate the possible change of optimal testing-effort expenditures when the expected initial faults a_1 is changed by $\pm$ 100x%. First, we define

$$\text{Relative Change (RC)} = \frac{MTEE - OTEE}{OTEE} \qquad (29)$$

where $OTEE$ is the original optimal-testing-effort-expenditures and $MTEE$ is the modified optimal-testing-effort-expenditures.

Assuming we have obtained the optimal testing-effort expenditures to each software module that minimizes the expected cost of software, then we can calculate the *MTEE* concerning the changes of expected initial fault a_i for the specific module i. The procedure can be repeated for various values of a_i. For example, for the data set used in Section 4.1, if the expected initial fault a_1 of module 1 is increased or decreased by 40%, 30%, 20%, or 10%, then the modified testing-effort expenditures for each software module can be obtained by following the similar procedures described in Section 3.2. Figure 1 plots relative change of the optimal testing-effort expenditures for the case of 40%, 30%, 20%, and 10% increase to a_1. The result indicates that the estimated values of optimal testing-effort expenditures will be changed when a_1 changes. That is, if a_1 is increased by 40%, then the estimated value of optimal testing-effort expenditure for module 1 is changed from 7632 to 8400 and its RC is 0.100628931 (about 10% increment). But for modules 2, 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are about 0.95%, 0.95%, 1.52%, 0.67%, 1.93%, 2.87%, 2.30%, and 4.49% decrement, respectively. Therefore, the variation in a_1 has the significant influence on the optimal allocation of testing-effort expenditures.

Besides, from Figure 1, we can also know that if the change of a_1 is small, the sensitivity of the optimal testing-resources allocation with respect to the value of a_1 is low. Next, we will show the same comparison results in case that a_1 is decreased by 100x%. From Figure 2, it is shown that if a_1 is decreased by 30%, the estimated value of optimal testing-effort expenditure for module 1 is changed from 7632 to 6818 and its RC is 0.106656184 (about 10.66% decrement). It is noted that for modules 2, 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are about 1.01%, 1.02%, 1.64%, 0.71%, 2.06%, 3.05%, 2.44%, and 4.86% increment, respectively. A similar procedure and conclusion can be obtained for the other parameters, such as a_2, a_3, …, or a_{10}. We perform an extensive sensitivity analysis for the expected initial faults as shown above. However, each a_i is considered in isolation. Here we try to study the effects of simultaneous changes of a_i & a_j ($j \neq i$). If we let a_1 & a_2 both be increased by 40%, then the estimated values of optimal testing-effort expenditure for modules 1 and 2 are changed from 7632 to 8370 (about 9.67% increment) and 3158 to 3764 (about 19.18% increment), respectively. But for modules 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are about 1.75%, 2.79%, 1.23%, 3.52%, 5.25%, 4.19%, and 8.22% decrement, respectively. Therefore, the variation in a_1 & a_2 has the significant influence on the optimal allocation of testing-effort expenditures. Similarly, from Figure 3, we can see that if the changes of a_1 & a_2 are less, the sensitivity of the optimal testing-resources allocation with respect to the values of a_1 & a_2 is low. On the other hand, from Figure 4, it is also shown that if a_1 & a_2 are both decreased by 30%, the estimated values of optimal testing-effort expenditure for modules 1 and 2 are changed from 7632 to 6850 (about 10.24% decrement) and 3158 to 2516 (about 20.32% decrement), respectively. It is also noted that for module 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are, respectively, about 1.87%, 2.98%, 1.31%, 3.77%, 5.56%, 4.47%, and 8.81% increment.

Finally, we can conclude that if a_i is changed, it will impose much influence on the estimated values of optimal testing-effort expenditure for module i. A decrease in a_i will decrease the estimated value of optimal testing-effort expenditure for module i but the estimated value of optimal testing-effort expenditures for the other module j ($j \neq i$) will be increased, and vice versa.

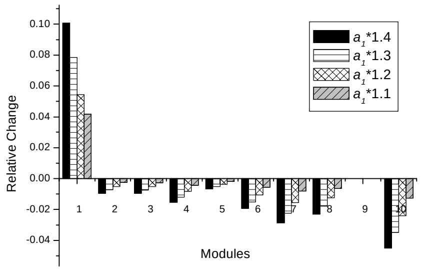

Figure 1: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% increase to a_1.

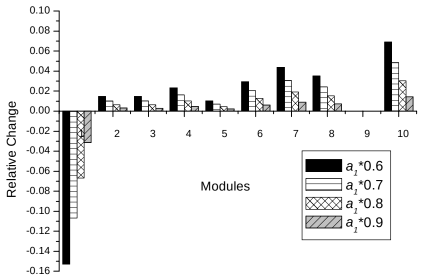

Figure 2: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% decrease to a_1.

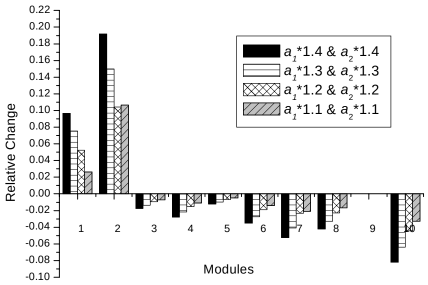

Figure 3: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% increase to a_1 & a_2.

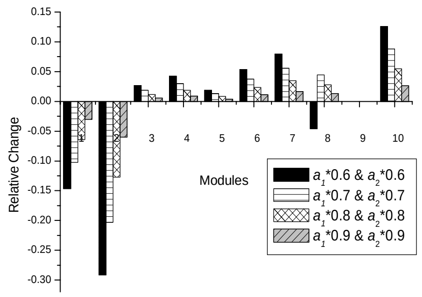

Figure 4: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% decrease to a_1 & a_2.

4.2.2. Effect of variations on fault detection rate. In this subsection we investigate the sensitivity of fault detection rate. Similarly, for the data set used in Section 4.1, if the fault detection rate r_1 of module 1 is increased or decreased, respectively, by 40%, 30%, 20%, or 10%, then the modified testing-effort expenditures for each software module can be calculated by following the similar procedures described in Section 3.2. Figure 5 plots the relative change of the optimal testing-effort expenditures for the case of 40%, 30%, 20%, and 10% increase to r_1. We can find that the estimated values of optimal testing-effort expenditures will be changed when r_1 changes. That is, if r_1 is increased by 40%, then the estimated value of optimal testing-effort expenditure for module 1 is changed from 7632 to 6079 and its RC is -0.203 (about 20% decrement). But for modules 2, 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are about 1.93%, 1.96%, 3.12%, 1.36%, 3.94%, 5.83%, 4.68%, and 9.21% increment, respectively. Therefore, compared with a_1, the variation in r_1 has less influence on the optimal allocation of testing-effort expenditures. Besides, from Figure 5, we can also see that if the change of r_1 is small, the sensitivity of the optimal testing-resources allocation with respect to the value of r_1 is low.

From Figure 6, it is seen that if r_1 is decreased by 30%, the estimated value of optimal testing-effort expenditure for module 1 is changed from 7632 to 9554 and its RC is 0.252 (about 25% increment). It is also noted that for modules 2, 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are about 2.37%, 2.39%, 3.83%, 1.68%, 4.86%, 7.21%, 5.78%, and 11.30% decrement, respectively. Nevertheless, a similar procedure and conclusion can be obtained for the other parameters, such as r_2, r_3, …, or r_{10}.

Furthermore, if we let r_1 & r_2 be increased by 40%, then the estimated values of optimal testing-effort expenditure for modules 1 and 2 are changed from 7632 to 6094 (about 20% decrement) and 3158 to 2783 (about 12% decrement), respectively. But for modules 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are about 2.49%, 3.99%, 1.75%, 5.03%, 7.47%, 5.99%, and 11.80% increment. Therefore, the variation in r_1 & r_2 imposes the most significant influence on the optimal allocation of testing-effort expenditures. Similarly, from Figure 7, we can also observe that if the changes of r_1 & r_2 are small, the sensitivity of the optimal testing-resources allocation with respect to the values of a_1 & a_2 is low. On the other hand, from Figure 8, it is shown that if r_1 & r_2 are both decreased by 30%, the estimated values of optimal testing-effort expenditure for modules 1 and 2 are changed from 7632 to 9534 (about 24.92% increment) and 3158 to 3387 (about 7.25% increment), respectively. It is noted that for modules 3, 4, 5, 6, 7, 8, and 10, the estimated values of optimal testing-effort expenditures are about 2.77%, 4.43%, 1.94%, 5.60%, 8.32%, 6.66%, and 13.07% decrement, respectively.

Finally, it is obvious that if r_i is changed, there is less influence on the estimated values of optimal testing-effort expenditure for module i, compared with a_i. A decrease in r_i, thus increases the estimated value of optimal testing-effort expenditure for module i and decreases the estimated value of optimal testing-effort expenditures for the other module j ($j \neq i$), and vice versa.

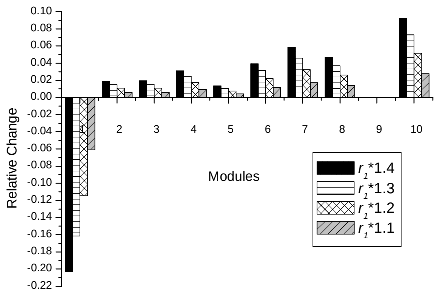

Figure 5: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% increase to r_1.

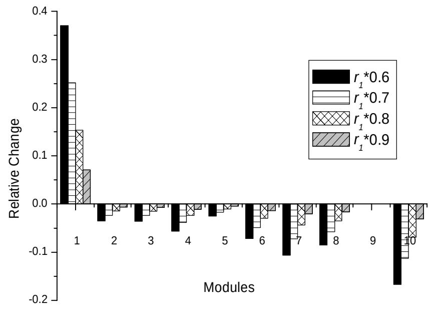

Figure 6: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% decrease to r_1.

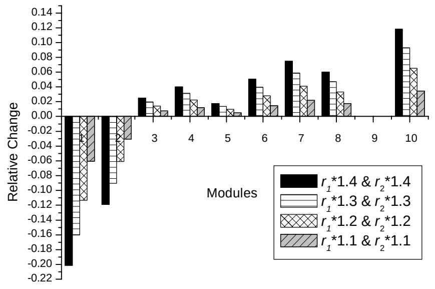

Figure 7: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% increase to r_1 & r_2.

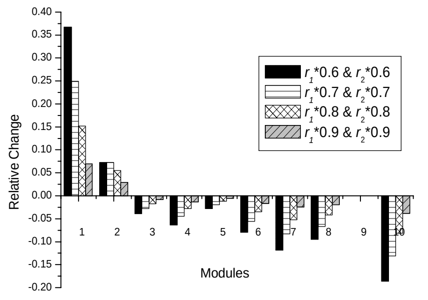

Figure 8: Relative change of OTEE for the case of 40%, 30%, 20%, and 10% decrease to r_1 & r_2.

5. Conclusions

This paper proposes a method to optimize the software testing-resource allocation problem. It minimizes the cost of software development, with a given number of remaining faults and a reliability objective. We develop a comprehensive strategy for module testing in order to help software project managers make the best decisions in practice. Numerical examples are described and discussed. In addition, an extensive sensitivity analysis is presented to study the effects of various principal parameters on the optimization problem of testing-resource allocation. We perform an extensive sensitivity analysis for each of the main parameters.

6. Acknowledgement

This research was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC 92-2213-E-007-073 and also substantially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. CUHK4182/ 03E).

References

[1] M. R. Lyu (1996), *Handbook of Software Reliability Engineering*, McGraw Hill.

[2] M. Xie (1991), *Software Reliability Modeling*, World Scientific Publishing Company.

[3] J. D. Musa, A. Iannino, and K. Okumoto (1987), *Software Reliability, Measurement, Prediction and Application*, McGraw Hill.

[4] C. Y. Huang, J. H. Lo, S. Y. Kuo, and M. R. Lyu, “Software Reliability Modeling and Cost Estimation Incorporating Testing-Effort and Efficiency,” *Proceedings of the 10th IEEE International Symposium on Software Reliability Engineering* (ISSRE'99), pp. 62-72, Nov. 1999, Boca Raton, FL, U.S.A.

[5] S. Y. Kuo, C. Y. Huang, and M. R. Lyu, “Framework for Modeling Software Reliability, Using Various Testing-Efforts and Fault-Detection Rates,” *IEEE Trans. on Reliability*, Vol. 50, No. 3, pp. 310-320, Sep. 2001.

[6] C. Y. Huang and S. Y. Kuo, “Analysis and Assessment of Incorporating Logistic Testing Effort Function into Software Reliability Modeling,” *IEEE Trans. on Reliability*, Vol. 51, No. 3, pp. 261-270, Sept. 2002.

[7] O. Berman and M. Cutler, “Optimal Software Implementation Considering Reliability and Cost,” *Computers and Operations Research*, Vol. 25, No. 10, pp. 857-868, 1998.

[8] B. Boehm, C. Abts, and S. Chulani, “Software Development Cost Estimation Approaches–A Survey,” *Annals of Software Engineering*, Vol. 10, Issue 1-4, pp. 177-205, 2000.

[9] P. Kubat and H. S. Koch, “Managing Test-Procedure to Achieve Reliable Software,” *IEEE Trans. on Reliability*, vol. 32, No. 3, pp. 299-303, 1983.

[10] P. Kubat, “Assessing Reliability of Modular Software,” *Operation Research Letters*, vol. 8, No. 1, pp. 35-41, 1989.

[11] O. Berman and N. Ashrafi, “Optimization Models for Reliability of Modular Software Systems,” *IEEE Trans. on Software Engineering*, vol. 19, no. 11, pp. 1119-1123, Nov. 1993.

[12] S. Yamada, T. Ichimori, and M. Nishiwaki, “Optimal Allocation Policies for Testing Resource Based on a Software Reliability Growth Model,” *International Journal of Mathematical and Computer Modelling*, Vol. 22, pp. 295-301,

1995.

[13] 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.

[14] M. E. Helander, M. Zhao, and N. Ohlsson, “Planning Models for Software Reliability and Cost,” *IEEE Trans. on Software Engineering*, vol. 24, no. 6, pp. 420-434, June 1998.

[15] H. Ohtera and S. Yamada, “Optimal Allocation and Control Problems for Software-Testing Resources,” *IEEE Trans. on Reliability*, vol. 39, No. 2, pp. 171-176, 1990.

[16] T. Ichimori, H. Masuyama, and S. Yamada, “A Two-Resource Allocation Problem according to an Exponential Objective: Optimum Distribution of Searching Effort,” *International Journal of Reliability, Quality and Safety Engineering*, Vol. 1, No. 2, pp. 135-146, 1994.

[17] Y. W. Leung, “Dynamic Resource Allocation for Software Module Testing,” *The Journal of Systems and Software*, vol.37, No.2, pp.129-139, May 1997.

[18] Y. W. Leung, “Software Reliability Allocation under Uncertain Operational Profiles,” *Journal of the Operational Research Society*, vol.48, No.4, pp.401-411, April 1997.

[19] Y. W. Leung, “Optimal Reliability Allocation for Modular Software System Designed for Multiple Customers,” *IEICE Trans. on Information and Systems*, vol.79, No.12, pp.1655-1662, Dec. 1996.

[20] R. H. Huo, S. Y. Kuo, and Y. P. Chang, “Needed Resources for Software Module Test, Using the Hyper-Geometric Software Reliability Growth Model,” *IEEE Trans. on Reliability*, Vol. 45, No.4, pp. 541-549, Dec. 1996.

[21] R. H. Huo, “Software Reliability Modeling and Its Applications,” *Ph.D. Dissertation*, Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan, 1996.

[22] J. H. Lo, C. Y. Huang, S. Y. Kuo, and M. R. Lyu, “Optimal Resource Allocation and Reliability Analysis for Component-Based Software Applications,” to appear in *Proceedings of the 26rd IEEE Annual International Computer Software and Applications Conference* (COMPSAC 2002), pp. 7-12, Aug. 2002, Oxford, England.

[23] B. Yang and M. Xie, “Testing-Resource Allocation for Redundant Software Systems,” *Proceedings of the 1999 Pacific Rim International Symposium on Dependable Computing* (PRDC'99), pp. 78-83, Dec. 1999, Hong Kong, China.

[24] B. Yang and M. Xie, “Optimal Testing-time Allocation for Modular Systems,” *International Journal of Quality and Reliability Management*, Vol. 18, No. 8, pp. 854-863, 2001.

[25] 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 Trans. on Reliability,* Vol. 51, No. 2, pp. 183-192, June 2002.

[26] M. R. Lyu, S. Rangarajan, and A. P. A. van Moorsel, “Optimization of Reliability Allocation and Testing Schedule for Software Systems,” *Proceedings of the 8th International Symposium on Software Reliability Engineering* (ISSRE'97), pp. 336-346, Nov. 1997, Albuquerque, New Mexico.

[27] C. Y. Huang, J. H. Lo, S. Y. Kuo, and M. R. Lyu, “Optimal Allocation of Testing Resources for Modular Software Systems,” *Proceedings of the Thirteenth IEEE International Symposium on Software Reliability Engineering* (ISSRE 2002), pp. 129-138, Nov. 2002, Annapolis, Maryland.

[28] B. Littlewood, “Software Reliability Model for Modular Program Structure”, *IEEE Trans. on Reliability*, vol. 28, No. 3, pp. 241-4246, 1979.

[29] F. Zahedi and N. Ashrafi, “Software Reliability Allocation Based on Structure, Utility, Price, and Cost,” *IEEE Trans. on Software Engineering*, vol. 17, no. 4, pp. 345-355, April 1991.

[30] J. Marciniak and R. Vienneau, “Software Engineering Baselines,” *Technical Report*, Data & Analysis Center for Software, July 1996.

[31] L. S. Lasdon (1970), *Optimization Theory for Large Systems*, MacMillan.

[32] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty (1993), *Nonlinear Programming: Theory and Algorithms*, 2nd Ed., John Wiley & Sons.

[33] M. Xie and G. Y. Hong, “A Study of the Sensitivity of Software Release Time,” *Journal of Systems and Software*, Vol. 44, Issue 2, pp. 163-168, Dec. 1998.

[34] S. S. Gokhale, and K. S. Trivedi, “Reliability Prediction and Sensitivity Analysis Based on Software Architecture,” *Proceedings of the Thirteenth IEEE International Symposium on Software Reliability Engineering* (ISSRE 2002), pp. 64-75, Nov. 2002, Annapolis, Maryland.

[35] H. W. Jung and B. Choi, “Optimization Models for Quality and Cost of Modular Software Systems,” *European Journal of Operational Research*, pp. 613-619, 1999.