Optimal Allocation of Testing Resources for Modular Software Systems

Chin-Yu Huang[1], Jung-Hua Lo[1], Sy-Yen Kuo[1], and Michael R. Lyu[2]
[1]*Department of Electrical Engineering*
National Taiwan University
Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw
[2]*Computer Science & Engineering Department*
The Chinese University of Hong Kong
Shatin, Hong Kong
lyu@cse.cuhk.edu.hk

Abstract

In this paper, based on software reliability growth models with generalized logistic testing-effort function, we study three optimal resource allocation problems in modular software systems during testing phase: 1) minimization of the remaining faults when a fixed amount of testing-effort and a desired reliability objective are given, 2) minimization of the required amount of testing-effort when a specific number of remaining faults and a desired reliability objective are given, and 3) minimization of the cost when the number of remaining faults and a desired reliability objective are given. Several useful optimization algorithms based on the Lagrange multiplier method are proposed and numerical examples are illustrated. Our methodologies provide practical approaches to the optimization of testing-resource allocation with a reliability objective. In addition, we also introduce the testing-resource control problem and compare different resource allocation methods. Finally, we demonstrate how these analytical approaches can be employed in the integration testing. Using the proposed algorithms, project managers can allocate limited testing-resource easily and efficiently and thus achieve the highest reliability objective during software module and integration testing.

1. Introduction

Modern computer systems, containing both hardware and software, have become very complex. Software takes a large portion of the system cost and requires the achievement of high reliability, i.e., a high confidence in the ability of the software to perform without error. In fact, software reliability represents a customer-oriented view of software quality. It relates to practical operation of a program rather than simply its design and implementation. Measuring or predicting software reliability is a challenge and its result is important as a quantitative assessment of the performance of the underlying software systems. This is especially true before the software systems are released to the market. Therefore, research efforts in software reliability engineering have been conducted over the past three decades and many software reliability growth models (SRGMs) have been proposed [1-2]. In fact, in addition to software reliability measurement, SRGMs can also help us to predict the fault detection coverage in the testing phase.

Practically, a software testing process consists of several testing stages including *module testing*, *integration testing*, *system testing* and *installation testing*. 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 [2-4]. In general, most popular and commercial software products are complex systems and composed of a number of modules. As soon as the modules are developed, they have to be tested in a variety of ways. If the modules are simple enough, test cases with analytic solutions can be generated to exercise the mathematical accuracy of the modules [3-4]. Similar tests can be run with various levels of coupling of the modules. Furthermore, all the testing activities of different modules should be completed within a limited time since they will consume approximately 40%~50% of the total amount of software development resources [4-7]. Therefore, project managers should know how to allocate the specified testing-resources among all the modules and develop quality software with high reliability. Many papers have addressed the optimal resource allocation problem over the years, including Kubat et al. (1983 & 1989) [4-5], Yamada et al. (1991 & 1995) [6-7], Leung (1992 & 1996) [8-10], Hou et al. (1996) [11], Xie et al. (1999 & 2001) [12-13], and Lyu et al. (2002) [14].

In this paper, we consider three kinds of software testing-resource allocation problems and propose several strategies for module testing in order to help the managers to make the best decisions. That is, we provide systematic methods for the software project managers to allocate specific amount of testing-resource expenditures for each module under some constraints, such as (1) minimizing the number of remaining faults with a reliability objective, (2) minimizing the amount of testing-effort with a reliability objective, or (3) minimizing the cost with a reliability

Proceedings of the 13 th International Symposium on Software Reliability Engineering (ISSRE'02)
1071-9458/02 $17.00 © 2002 IEEE

objective. Here we use an SRGM with generalized *logistic* testing-effort function to describe the time-dependency behaviors of detected software faults and the testing-resource expenditures spent during module testing. Besides, we also study a testing-resource allocation strategy for software integration testing when all modules are interconnected and tested together.

The remaining of the paper consists of four sections. Section 2 describes an SRGM with generalized logistic testing-effort function, which is based on non-homogeneous Poisson processes (NHPPs). In Section 3, the methods for testing resource allocation and optimization for modular software testing are introduced. We investigate these optimization problems for three different requirements: *minimizing the number of remaining faults*, *minimizing the amount of testing-effort*, and *minimizing the cost*. Several numerical examples for the optimum testing-effort allocation problem are also demonstrated. Besides, a testing-resources control problem and comparisons among different resource allocation methods are presented. Furthermore, testing resource allocation and optimization for integration testing are discussed in Sections 4. Finally, the concluding remarks are given in Section 5.

2. 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. [6-7] modified the G-O model and incorporated the concept of testing-effort in an NHPP model to get a better description of the software fault phenomenon. Later, we also proposed 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 our past experimental results, this proposed approach is well suitable for estimating the reliability of software application during the development process [15-17]. 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.

Based on the 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:

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

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, r is the error detection rate per unit testing-effort at testing time t and $r>0$.

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

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

In Eq. (1), $m(t)$ is non-decreasing with respect to testing time t. Knowing its value can help us to determine whether the software is ready for release and how much more testing resources are required [1]. It can provide an estimate of the number of failures that will eventually be encountered by the customers. Besides, we also proposed a generalized *logistic* testing-effort function with structuring index, 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 [16]. The generalized *logistic* testing-effort function is depicted as follow:

$$W_\kappa(t) = N \times \left(\frac{(\kappa+1)/\beta}{1 + Ae^{-\alpha\kappa t}} \right)^{1/\kappa} \quad (2)$$

where N is the total amount of testing effort to be eventually consumed, α is the consumption rate of testing-effort expenditures, A is a constant, and κ is a structuring index with a large value for modeling well-structured software development efforts. In this case, the testing effort reaches its maximum value at time

$$t^{\kappa}_{\max} = \frac{\ln \frac{A}{\kappa}}{\alpha\kappa}$$

The conditional reliability function after the last failure occurs at time t is obtained by

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

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

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

From the above equation and Eq. (1) we can determine the testing time needed to reach a desired reliability R_0. We also can 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 [17].

$$R(t) \equiv \frac{m(t)}{a} \quad (3)$$

We can solve this equation and obtain a unique t satisfying

$R(t)=R_0$. Note that $R(t)$ is 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 reached at a specified time. If we know that the software reliability of a computer system has reached an acceptable reliability level, then we can determine the right time to release this software.

3. Testing-resource allocation for module testing

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

Assumptions [4-7]**:**

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 is faulty, the whole software system fails.
5. The system manager has to allocate the total testing resources W to each software module to minimize the number of faults remaining in the system during the testing period. Besides, the desired software reliability after the testing phase is greater than or equal to the reliability objective R_0.

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

$$M(t)=\sum_{i=1}^{N} v_i m_i(t) = \sum_{i=1}^{N} v_i a_i (1-\exp(-r_i W_i(t))) \quad (4)$$

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 this testing phase [18]. This indicates that the number of remaining faults in the system can be estimated by

$$\sum_{i=1}^{N} v_i a_i \exp(-r_i W_i(t)) \equiv \sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) \quad (5)$$

We can further formulate three optimization problems as follows.

3.1. Minimizing the number of remaining faults with a given fixed amount of testing-effort and a reliability objective

Firstly, the optimization problem is that the total amount of testing-effort is fixed, and we want to allocate these efforts to each module in order to minimize the number of remaining faults in the software systems. Specially, suppose the total amount of testing-effort is W, and module i is allocated W_i testing-effort, and then the optimization problem can be represented as follows:

The objective function is:

$$\text{Minimize:} \quad \sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) \quad (6)$$

Subject to the constrains:

$$\sum_{i=1}^{N} W_i \le W, \quad W_i \ge 0, \ i=1, 2,..., N. \quad (7)$$

$$R = 1-\exp(-r_i W_i(t)) \ge R_0 \quad (8)$$

From Eq. (8), we can obtain W_i

$$W_i \ge \frac{1}{r_i}\ln(1+R_0), \quad i=1,2,...,N. \quad (9)$$

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

Thus, we can have

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

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

That is, the optimal testing resource allocation can be depicted as below [7]:

$$\text{Minimize:} \sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) \quad (10)$$

$$\text{Subject to} \sum_{i=1}^{N} W_i = W, \quad W_i \ge 0 \text{ and, } W_i \ge C_i$$

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

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

$$\text{Minimize:} \quad \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i) \quad (11)$$

$$\text{Subject to} \quad \sum_{i=1}^{N} X_i \le W - \sum_{i=1}^{N} C_i, \quad X_i \ge 0, \ i=1,2, ..., N. \quad (12)$$

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. Consequently, Eq. (11) and Eq. (12) can be simplified as follows:

Minimize:

$$L(X_1, X_2, ..., X_N, \lambda) = \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i) \exp(-r_i X_i) + \lambda(\sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i) \quad (13)$$

Based on the Kuhn-Tucker conditions [19-21], the necessary conditions for a minimum value of Eq. (13) to exist are as follows:

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

A2: $\lambda \geq 0$.

A3: $\sum_{i=1}^{N} X_i \leq W - \sum_{i=1}^{N} C_i$, $W_i \geq 0$, i=1, 2,..., N.

Therefore, from Eq. (13), we have

$$\frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial X_i} = -v_i a_i r_i \times \exp(-r_i C_i) \times \exp(-r_i X_i) + \lambda = 0 \quad (14)$$

$$\frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial \lambda} = \sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i = 0$$

Thus, the solution X_i^0 is

$$X_i^0 = (\ln(v_i a_i r_i \times \exp(-r_i C_i)) - \ln \lambda^0)/r_i, \quad i=1, 2,..., N. \quad (15)$$

The solution λ^0 is

$$\lambda^0 = \exp\left[\frac{\sum_{i=1}^{N}(1/r_i)(\ln v_i a_i r_i \times \exp(-r_i C_i)) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N}(1/r_i)}\right] \quad (16)$$

Hence, we get $X^0 = (X_1^0, X_2^0, X_3^0, ..., X_N^0)$ as an optimal solution to Eq. (13). However, the above X^0 may have some negative components if $v_i a_i r_i \times \exp(-r_i C_i) < \lambda^0$, making X^0 infeasible for Eq. (11) and Eq. (12). 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

$$X_i = \frac{1}{r_i}[\ln(v_i a_i r_i \times \exp(-r_i C_i)) - \ln \lambda], \quad i=1, 2,..., N-l.$$

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

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:

$$\begin{cases} X_i^* = (\ln(v_i a_i r_i) \times \exp(-r_i C_i) - \ln \lambda)/r_i, \quad i = 1,2,...,N-l, \\ where \ \lambda = \exp\left[\dfrac{\sum_{i=1}^{N-l}(1/r_i)(\ln v_i a_i r_i \times \exp(-r_i C_i)) - W + \sum_{i=1}^{N} C_i}{\sum_{i=1}^{N-l}(1/r_i)}\right] \\ X_i^* = 0, otherwise \end{cases} \quad (17)$$

Algorithm 1 always converges in, at worst, N–1 steps. Thus the value of objective function given by Eq. (11) at the optimal solution $(X_1^*, X_2^*, ..., X_N^*)$ is

$$\sum_{i=1}^{N} v_i a_i \exp(-r_i C_i) \exp(-r_i X_i^*) \quad (18)$$

3.2. Minimizing the amount of testing-effort given the number of remaining faults and a reliability objective

Secondly, suppose Z specifies the number of remaining faults in the system and we have to allocate an amount of testing-effort to each software module to minimize the total testing-effort. The optimization problem can be represented as follows:

The objective function is:

$$Minimize: \ \sum_{i=1}^{N} W_i, \quad (19)$$

Subject to the constrains:

$$\sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) = Z, \quad , W_i \geq 0 \quad (20)$$

$$R = 1 - \exp(-r_i W_i(t)) \geq R_0 \quad (21)$$

Similarly, from Eq. (21), we can obtain W_i

$$W_i \geq \frac{1}{r_i}\ln(1 + R_0), \quad i = 1,2,...,N. \quad (22)$$

Following the similar steps described in section 3.1 and let $X_i = W_i - C_i$, we can transform above equations to:

$$Minimize: \ \sum_{i=1}^{N}(X_i + C_i), \quad (23)$$

Subject to

$$\sum_{i=1}^{N} v_i a_i \exp(-r_i C_i) \exp(-r_i X_i) = Z, X_i + C_i \geq 0 \quad (24)$$

To solve this problem, the Lagrange multiplier method can be used. Eq. (23) and Eq. (24) are combined as the following equation:

$$\textit{Minimize: } L(X_1, X_2, ..., X_N, \lambda) = \sum_{i=1}^{N}(X_i + C_i) + \lambda(\sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i) - Z) \quad (25)$$

From the Kuhn-Tucker conditions, the necessary conditions for a minimum of Eq. (25) to exist are:

$$\frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial X_i} = -\lambda v_i a_i r_i \exp(-r_i C_i) \exp(-r_i X_i) + 1 = 0 \quad (26)$$

$$\frac{\partial L(X_1, X_2, ..., X_N, \lambda)}{\partial \lambda} = \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i) - Z = 0 \quad (27)$$

Thus, the solution X_i^0 is

$$X_i^0 = \ln[\lambda v_i a_i r_i \exp(-r_i C_i)] / r_i, \; i=1, 2, ..., N. \quad (28)$$

The solution λ^0 is

$$\lambda^0 = \frac{\sum_{i=1}^{N}(1/r_i)}{Z} \quad (29)$$

That is

$$X_i^0 = (\ln(\frac{v_i a_i r_i}{Z}\exp(-r_i C_i)\sum_{i=1}^{N}\frac{1}{r_i})) / r_i, \; i=1, 2, ..., N. \quad (30)$$

Hence, we get $X^0 = \left(X_1^0, X_2^0, X_3^0, ..., X_N^0\right)$ as an optimal solution to Eq. (25). However, the above X^0 may have some negative components if

$$v_i a_i r_i \exp(-r_i C_i) < \frac{Z}{\sum_{i=1}^{N}\frac{1}{r_i}},$$

making X^0 infeasible for Eq. (23) & (24). In this case, the solution X^0 can be corrected by the following steps. Similarly, we propose a simple algorithm to determine the optimal solution for the testing-effort allocation problem.

Algorithm 2:

Step 1: Set l=0.

Step 2: Calculate

$$X_i = \frac{1}{r_i}[\ln(\frac{v_i a_i r_i}{Z}\exp(-r_i C_i)\sum_{i=1}^{N-l}\frac{1}{r_i})], \quad i=1, 2, ..., N-l.$$

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

Else

update $X_{N-l}^* = 0$; l=l+1.

End- IF.

Step 5: Go to **Step 2**.

The optimal solution has the following form:

$$X^*_i = \frac{1}{r_i}[\ln(\frac{v_i a_i r_i}{Z}\exp(-r_i C_i)\sum_{i=1}^{N-l}\frac{1}{r_i})], \quad i = 1,2,...,N-l. \quad (31)$$

Algorithm 2 always converges in, at worst, $N-1$ steps.

3.3. Minimizing the cost given the number of remaining faults and a reliability objective

Thirdly, suppose the number of remaining faults in the system is still specified by Z. We should allocate an amount of testing-effort to each software module to minimize the cost of further software development and test. Therefore, the optimization problem can be represented as follows:

The objective function is:

$$\textit{Minimize: } \sum_{i=1}^{N} Cost_i(W_i), \quad (32)$$

Subject to the constrains:

$$\sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) \leq Z, \quad W_i \geq 0. \quad (33)$$

$$R = 1 - \exp(-r_i W_i(t)) \geq R_0 \quad (34)$$

From Eq. (34), we can obtain W_i

$$W_i \geq \frac{1}{r_i}\ln(1 + R_0), \quad i = 1,2,...,N. \quad (35)$$

Similarly, using the methods described in section 3.1 and let $X_i = W_i - C_i$, we can transform the above equations to:

$$\textit{Minimize: } \sum_{i=1}^{N} Cost_i(X_i + C_i), \quad (36)$$

$$\textit{Subject to } \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i) = Z, \quad , X_i \geq 0. \quad (37)$$

Here we assume that the cost function $Cost_i(W_i)$ is the cost required to develop module i with W_i and is differentiable [5]. Similarly, using the Lagrange multiplier method, the above equation can be simplified as follows:

Minimize:

$$L = \sum_{i=1}^{N} Cost_i(X_i + C_i) + \lambda(\sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i) - Z) \quad (38)$$

Again based on the Kuhn-Tucker conditions [31], the necessary conditions for obtaining the minimum of Eq. (38) are:

$$\frac{\partial L}{\partial X_i} = \frac{\partial Cost_i(X_i + C_i)}{\partial X_i} - \lambda v_i a_i r_i \times \exp(-r_i C_i) \times \exp(-r_i X_i) = 0, \; i=1, 2, ..., N. \quad (39)$$

$$\frac{\partial L}{\partial \lambda} = \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i)\exp(-r_i X_i) - Z = 0 \qquad (40)$$

Here let $\zeta_i(X_i) = \frac{\partial Cost_i(X_i + C_i)}{\partial X_i}$, then Eq. (39) becomes

$$\frac{\partial}{\partial X_i} L = \zeta_i(X_i) - \lambda v_i a_i r_i \times \exp(-r_i C_i) \times \exp(-r_i X_i) = 0 \qquad (41)$$

The optimal solution X_i^0 is

$$\frac{\zeta_i(X_i)}{\exp(-r_i X_i)} - \lambda v_i a_i r_i \times \exp(-r_i C_i) = 0 \qquad (42)$$

In fact, the above is a simple optimization problem and can be easily solved by numerical methods [12].

3.4. Numerical examples

In this subsection, three numerical examples for the optimum testing-effort allocation problem are demonstrated. Here we assume that the estimated parameters a_i, r_i, and κ in Eq. (2), for i=1, 2,…, 10, are summarized in Table 1. Moreover, the weighting vectors v_i in Eq. (4) are also listed.In this subsection, three numerical examples for the optimum testing-effort allocation problem are demonstrated. Here we assume that the estimated parameters a_i, r_i, and κ in Eq. (2), for i=1, 2,…, 10, are summarized in Table 1. Moreover, the weighting vectors v_i in Eq. (4) are also listed. In the following, we illustrate three examples to show how the optimal allocation of testing-effort expenditures to each software module is determined.

3.4.1. Example of minimizing the number of remaining faults given a fixed amount of testing-effort and a reliability requirement. Suppose the total amount of testing-effort expenditures W is given. Besides, all the parameters a_i and r_i of Eq. (4) for each software module have been estimated by using the *maximum likelihood estimation* (MLE) or the *least squares estimation* (LSE). We apply the proposed model to actual software failure data [15-17]. We have to allocate the expenditures to each module to minimize the number of remaining faults. From Table 1 and Algorithm 1 in Section 3.1, the optimal testing-effort expenditures for the software systems are estimated and shown in Table 2. Furthermore, the numbers of initial faults, the estimated remaining faults of the three examples, and the reduction in the number of remaining faults are also shown in Table 3.

Table 1: The estimated values of a_i, r_i, v_i, and κ.

Module	a_i	r_i (10^{-4})	κ	v_i in ex.1	v_i in ex. 2	v_i in ex. 3
1	89	4.1823	1	1.0	1.0	0.5
2	25	5.0923	1	1.5	0.6	0.5
3	27	3.9611	1	1.3	0.7	0.7
4	45	2.2956	1	0.5	0.4	0.4
5	39	2.5336	1	2.0	1.0	1.5
6	39	1.7246	1	0.3	0.2	0.2
7	59	0.8819	1	1.7	0.5	0.6
8	68	0.7274	1	1.3	0.6	0.6
9	37	0.6824	1	1.0	0.1	0.9
10	14	1.5309	1	1.0	0.5	0.5

Table 2: The optimal testing-effort expenditures using algorithm 1.

Module	$X_i + C_i^*$ for ex.1	$X_i + C_i^*$ for ex.2	$X_i + C_i^*$ for ex.3
1	6254	8105	6015
2	3826	3547	2833
3	4117	4409	4052
4	2791	5191	4402
5	7825	8145	9030
6	0	403	0
7	13366	8267	8280
8	11820	11833	9343
9	0	0	6046
10	0	0	0

Table 3: The reduction in the number of remaining faults.

Examples	Initial faults	Remaining faults	Reduction (%)
1	514.0	172.0	33.6
2	268.7	68.5	25.5
3	276.7	97.4	35.2

3.4.2. Example of minimizing the amount of testing-effort given the number of remaining faults and reliability requirements. Suppose the total number of remaining faults Z is given. We have to allocate the expenditures to each module to minimize the total amount of testing-effort expenditures. Using Algorithm 2 in Section 2.2 and Table 1, the optimal solutions are derived and shown in Table 4. Furthermore, the relationship between the total amount of testing-effort expenditures and the reduction rate of the remaining faults are also depicted in Figure 1.

Table 4: The optimal testing-effort expenditures using algorithm 2.

Module	$X_i + C_i^*$ for ex.1	$X_i + C_i^*$ for ex.2	$X_i + C_i^*$ for ex.3
1	7700	6962	5941
2	5013	2608	2772
3	5643	3302	3974
4	5424	3109	4268
5	10211	6258	8908
6	1770	0	0
7	20220	2847	7931
8	20131	5263	8919
9	7759	0	5595
10	2388	0	0

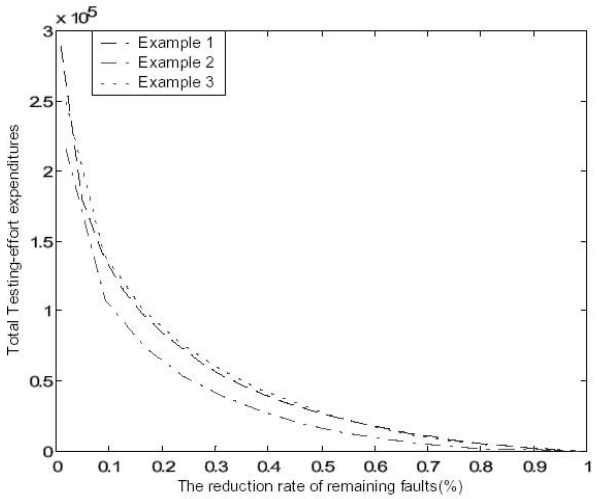

Figure 1: The reduction rate of remaining faults vs. the total testing-effort expenditures.

3.5. Testing- resources control problem

In order to reduce the risk and achieve a given operational quality at a specified time, we can use SRMGs to estimate and control the required testing effort. The main problem is how to estimate the number of extra faults during module testing which have to be found [6, 17, 22]. Let us consider the following scenario:

(1). Due to economic considerations, software testing and debugging will eventually have to stop at a time point, T_2.

(2). Based on the SRGM selected by the software developers or test teams, for each module its behavior of consuming testing resources during testing is estimated at time T_1 ($0<T_1<T_2$). Therefore, the project manager has to decide how to spend W_i^* (where $W_i \geq C_i$ and $C_i = \max\{0, D_1, D_2, D_3, ..., D_N\}$

Based on the above scenario, if we know how to adjust the consumption rate of testing effort expenditures in the logistic testing effort function (i.e., α_{2i}) at T_2, then the project manager can increase the testing-resource expenditures in $(T_1, T_2]$. First, from Eq. (2), we know that testing-effort consumption function at T_1 for each module is as follows:

$$w_{1i\kappa}(t) = \frac{N_{1i} A_{1i} \alpha_{1i} \kappa e^{-\alpha_{1i}\kappa t}}{\kappa} \times \left(1 + A_{1i} e^{-\alpha_{1i}\kappa t}\right)^{-(\frac{1+\kappa}{\kappa})} \tag{43}$$

Therefore, the mean testing-effort consumption function in $(T_1, T_2]$ is

$$w_{2i\kappa}(t) = \frac{N_{2i} A_{2i} \alpha_{2i} \kappa e^{-\alpha_{2i}\kappa t}}{\kappa} \times \left(1 + A_{2i} e^{-\alpha_{2i}\kappa t}\right)^{-(\frac{1+\kappa}{\kappa})} \tag{44}$$

and

$$N_{2i}(t) = \frac{N_{1i} A_{1i} \alpha_{1i} e^{-\alpha_{1i}\kappa t}}{A_{2i} \alpha_{2i} e^{-\alpha_{2i}\kappa t}} \times \left(\frac{1 + A_{1i} e^{-\alpha_{1i}\kappa t}}{1 + A_{2i} e^{-\alpha_{2i}\kappa t}}\right)^{-(\frac{1+\kappa}{\kappa})} \tag{45}$$

where $T_2 \geq t > T_1$.

Applying Eq. (45), we know

$$\int_{T_1}^{T_2} w_{2i\kappa}(\tau) d\tau = W_i^* - \int_0^{T_1} w_{1i\kappa}(\tau) d\tau \tag{46}$$

That is,

$$W_i^* = \left(\frac{N_{2i}}{\kappa\sqrt{1 + A_{2i} e^{-\alpha_{2i}\kappa T_2}}}\right) - \left(\frac{N_{2i}}{\kappa\sqrt{1 + A_{2i} e^{-\alpha_{2i}\kappa T_1}}}\right) + \left(\frac{N_{1i}}{\kappa\sqrt{1 + A_{1i} e^{-\alpha_{1i}\kappa T_1}}}\right) \tag{47}$$

Therefore, α_{2i} can be easily solved by numerical methods for the selected SRGM. The test teams can get the modified testing-effort consumption function in $(T_1, T_2]$ and predict the cumulative number of faults at time T_2 during module testing, and detect more extra faults during the time

interval $(T_1, T_2]$.

3.6. Comparisons between different resource allocation methods

In general, a testing-effort allocation method is called an average allocation method if the total amount of testing-effort is allocated to each module evenly. That is,

$$X_i^{ave} = \frac{W - \sum_{i=1}^{N} C_i}{N} \quad \text{for } \textit{module } i=1, 2,..., N. \tag{48}$$

Let Z_{ave} be the number of remaining faults in the whole system by the average testing-effort allocation method, then we have

$$Z_{ave} = \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i) \exp(-r_i X^{ave}{}_i) \tag{49}$$

Similarly, a testing-effort allocation method is called a proportional allocation method if the total amount of testing-effort allocated to module i is proportional to the number of remaining faults in module i. That is,

$$X_i^{prop} = W \times \frac{a_i \exp(-r_i C_i) \exp(-r_i X_i)}{\sum_{i=1}^{N} a_i \exp(-r_i C_i) \exp(-r_i X_i)} - \sum_{i=1}^{N} C_i ,$$

$i=1, 2,..., N.$ (50)

Let Z_{prop} be the number of remaining faults in the whole system by the proportional testing-effort allocation method, then we have

$$Z_{prop} = \sum_{i=1}^{N} v_i a_i \exp(-r_i C_i) \exp(-r_i X_i{}^{prop}) \tag{51}$$

Therefore, with Z_{ave}, Z_{prop}, and Z_{opt}, we can know whether the optimal allocation method is better than the average or proportional allocation methods or not.

4. Testing-resource allocation for integration testing

Integration testing involves testing the combinations of program modules and their interfaces. Integration testing should be well planned during the design phase and then accomplished with an appropriate balance of developers with design knowledge and independent testers with minimal design biases. During this testing phase, a fixed amount of test cases is prepared, representing actual values from program's input domain. Here we assume that N modules comprise the whole software system and the selected test cases can be divided into S categories during the integration testing. Let α_{ik} be the total throughput of test cases on module i for test data in category k. As we know, more executions of a software module will increase the chances of finding the remaining undetected faults in that module. In general, if the total throughput of test data is large, then the mean detection time of a fault will be decreased. Thus, the fault detection rate of module i can be denoted as $r_i(\alpha_{ik})$ and we can assume that $r_i(\alpha_{ik})$ is an increasing function of throughput [4-5, 23-27]. Here, due to the limitation of space, we only use the resource allocation problem described in Section 3.1 for illustration. Similar analysis and discussion can be applied to other cases in Section 3.2 and 3.3. The problem described in Section 3.1, minimizing the number of remaining faults given a fixed amount of testing-effort and a reliability requirement, can be formulated as

The objective function is:

$$\textit{Minimize: } \sum_{k=1}^{S} \sum_{i=1}^{N} v_i a_i \exp(-r_i(\alpha_{ik}) \times W_k) \tag{52}$$

Subject to the constrains:

$$\sum_{k=1}^{S} W_k = W, \quad W_k \geq 0, \; k=1, 2,..., S. \tag{53}$$

$$R = 1 - \exp(-r_i(\alpha_{ik}) W_k) \geq R_0 \tag{54}$$

From Eq. (54), we can obtain $W_k \geq \frac{1}{r_i(\alpha_{ik})} \ln(1 + R_0)$.

Following the similar steps described in section 3.1 and let $X_i = W_i - C_i$, we can transform the above equations to:

$$\textit{Minimize: } \sum_{k=1}^{S} \sum_{i=1}^{N} v_i a_i \exp(-r_i(\alpha_{ik}) C_k) \exp(-r_i(\alpha_{ik}) X_k) \tag{55}$$

$$\textit{Subject to } \sum_{k=1}^{S} X_k \leq W - \sum_{i=1}^{S} C_k, X_i \geq 0, \; k=1, 2,..., S. \tag{56}$$

Similarly, using the Lagrange multiplier method, the above equation can be simplified as follows:

$$L(W_1, W_2,..., W_S, \lambda) = \sum_{k=1}^{S} \sum_{i=1}^{N} v_i a_i \exp(-r_i(\alpha_{ik}) C_k) \times \exp(-r_i(\alpha_{ik}) X_k) + \lambda \left(\sum_{i=1}^{N} X_i - W + \sum_{i=1}^{N} C_i \right)$$

$$\frac{\partial L(W_1, W_2,..., W_S, \lambda)}{\partial X_k} = -\sum_{i=1}^{N} v_i a_i r_i(\alpha_{ik}) \exp(-r_i(\alpha_{ik}) C_k) \times \exp(-r_i(\alpha_{ik}) X^*{}_k) + \lambda^* = 0 \tag{57}$$

Thus,

$$\begin{cases} \sum_{i=1}^{N} v_i a_i r_i(\alpha_{ik}) \exp(-r_i(\alpha_{ik})C_k) \exp(-r_i(\alpha_{ik})X^*_k) = \lambda^*, \\ \quad if \quad X^*_k > 0 \\ \sum_{i=1}^{N} v_i a_i r_i(\alpha_{ik}) \exp(-r_i(\alpha_{ik})C_k) \exp(-r_i(\alpha_{ik})X^*_k) \le \lambda^*, \\ \quad if \quad X^*_k = 0 \\ \sum_{k=1}^{S} X_k \le W - \sum_{i=1}^{S} C_k, \quad k = 1,2,...,S \end{cases} \tag{58}$$

The above is an optimization problem and the optimal X^*_k and λ^* can be solved by numerical methods.

Algorithm 3:

Step 1: Set $l = 0$; guess λ_1 and specify ξ tolerance.

Step 2: Calculate $l = l + 1$. Solve $W^0{}_\kappa$ by numerical methods.

$$\sum_{i=1}^{N} v_i a_i r_i(\alpha_{ik}) \exp(-r_i(\alpha_{ik})C_k) \exp(-r_i(\alpha_{ik})X^0{}_k) = \lambda_l, \tag{59}$$

for all k, k =1, 2, 3,…, S.

Step 3: IF $X^0_k > 0$ then

set $X^*_k = X^0_k$

Else

set $X^*_k = 0$

End- IF.

Step 4: If $W - \sum_{i=1}^{S} C_k - \xi \le \sum_{k=1}^{S} X^*_k \le W - \sum_{i=1}^{S} C_k + \xi$ then

stop

End-if.

Step 5: If $\sum_{k=1}^{S} X^*_k < W - \sum_{i=1}^{S} C_k - \xi$ then

select $\lambda_{l+1} < \lambda_l$

Else

select $\lambda_{l+1} > \lambda_l$

End-if.

Step 6: Go to Step 2.

Note that if $r_i(\alpha_{ik})(C_k + X_k) << 1$, then we can have [4, 19]:

$$\exp(-r_i(\alpha_{ik})(C_k + X_k)) \approx 1 - r_i(\alpha_{ik})(C_k + X_k) + \frac{(r_i(\alpha_{ik})(C_k + X_k))^2}{2} \tag{60}$$

Therefore, follow the similar algorithmic procedures as before, we can obtain X^*_k and λ^*.

5. Conclusions

In this paper, we consider three kinds of software testing-resource allocation problems. The first problem is to minimize the number of remaining faults given a fixed amount of testing-effort and a reliability objective. The second problem is to minimize the amount of testing-effort given the number of remaining faults and a reliability objective. The third problem minimizes the cost given the number of remaining faults and a reliability objective. We propose several strategies for module testing in order to help software project managers to solve these problems and to make the best decisions. Namely, we provide several systematic solutions based on an NHPP model, allowing these managers to easily allocate a specified amount of testing-resource expenditures for each software module under some constraints. We describe numerical examples on the optimal testing-resource allocation problems to show applications and impacts of the proposed strategies during module testing. Finally, we extend our approach to integration testing for the testing-resource management strategies, and provide complete analytical solutions for them.

6. Acknowledgement

This research was supported by the National Science Council, Taiwan, ROC., under Grant NSC 90-2213-E-002-113 and also substantially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region (Project No. CUHK4222/01E). Further, we thank the anonymous referees for their critical review and comments.

7. References

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

[2] 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.

[3] D. W. Coit, "Economic Allocation of Test Times for Subsystem-Level Reliability Growth Testing," *IIE Transactions*, vol. 30, no. 12, pp. 1143–1151, Dec.1998.

[4] 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.

[5] P. Kubat, "Assessing reliability of Modular Software," *Operation Research Letters*, vol. 8, No. 1, pp. 35-41, 1989.

[6] 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.

[7] 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, pp. 295-301, 1995.

[8] 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.

[9] 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.

[10] 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, December 1996.

[11] R. H. Huo, S. Y. Kuo, and Y. P. Chang, "Efficient Allocation of Testing Resources for Software Module Testing Based on the Hyper-Geometric Distribution Software Reliability Growth Model," *IEEE Trans. on Reliability*, Vol. 45, No.4, pp. 541-549, Dec. 1996.

[12] 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), Dec. 1999, Hong Kong, China.

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

[14] 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.

[15] 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.

[16] C. Y. Huang and S. Y. Kuo, "Analysis and Assessment of Incorporating Logistic Testing Effort Function into Software Reliability Modeling," to appear in *IEEE Trans. on Reliability*, Sept. 2002.

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

[18] C. Y. Huang, J. H. Lo and S. Y. Kuo, "A Pragmatic Study of Parametric Decomposition Models for Estimating Software Reliability Growth," *Proceedings of the 9th International Symposium on Software Reliability Engineering* (ISSRE'98), pp. 111-123, Nov. 1998, Paderborn, Germany.

[19] G. L. Nemhauser, A. H. G. Rinnooy Kan, M.J. Todd, "*Optimization: Handbooks in Operations Research and Management Science; v.1*," North-Holland

[20] D. P. Heyman and M. J. Sobel, "*Stochastic Models: Handbooks in Operations Research and Management Science; v.2*," New York, N.Y. North-Holland

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

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

[23] 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.

[24] Russell A. Fink, "Reliability Modeling of Freely-Available Internet-Distributed Software," *Proceedings of the 5th International Symposium on Metrics*, pp. 101-104, 1998.

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

[26] R. L. Bulfin and C. Y. Liu, "Optimal Allocation of Redundant Components for Large Systems," *IEEE Trans. Reliability*, vol. R-34, pp. 241–247, 1985.

[27] 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.