Optimal Resource Allocation and Reliability Analysis for Component-Based Software Applications

Jung-Hua Lo, Sy-Yen Kuo, Michael R. Lyu*, and Chin-Yu Huang
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw

Abstract

In this paper we propose an analytical approach for estimating the reliability of a component-based software. This methodology assumes that the software components are heterogeneous and the transfers of control between components follow a discrete time Markov process. Besides, we also formulate and solve two resource allocation problems. Finally, we demonstrate how these analytical approaches can be employed to measure the reliability of a software system including multiple-input/multiple-output systems and distributed software systems. Experimental results show that the proposed methods can solve the testing-effort allocation problems and improve the quality and reliability of a software system.

1. Introduction

With the great advancement of computer technology, software designers are motivated to integrate commercial off-the-shelf (COTS) software components for rapid software development. To ensure high reliability for such applications using software components as their building blocks to construct a software system, dependable components have to be deployed to meet the reliability requirements. Therefore, it is necessary to assess the reliabilities of such systems by investigating the architectures, the testing strategies, and the component reliabilities [1-4].To ensure the overall reliability of a software application, software components in the system have to meet certain reliability requirements, subject to some resource constraints [5-9]. These resources include human power, CPU hours, and elapsed time, etc. Without loss of generality, we call all these resources as the testing-effort [10]. Hence, to develop a good reliable software system, a project manager must determine in advance how to effectively allocate these resources [7]. In this paper, we investigate two optimal testing-effort allocation problems: minimization of the number of remaining faults in a system given a fixed amount of testing-effort and minimization of the total amount of testing-effort given specific reliability requirements [5-9]. The organization of this paper is as follows. Section 2 presents an analytical approach to estimating the reliability of a system. Two optimum testing-effort allocation problems are discussed in Section 3. Section 4 illustrates how the proposed approach is actually applied on the three applications. Conclusions and future works are presented in Section 5.

*Michael R. Lyu is with Computer Science and Engineering Department, The Chinese University of Hong Kong, Shatin, Hong Kong.

2. Reliability analysis for component-based systems

A software system can be regarded as composed of logically individual components, which can be implemented and tested independently [2, 11-14]. In this section, we propose an approach to estimating the reliability of a component-based system by taking the architecture of the software system and the reliabilities of the components into consideration. For example, if a system consists of n components with reliabilities denoted by $R_1,\ldots, R_n$ respectively, the reliability of an execution path, 1, 3, 2, 3, 2, 3, 4, 3, n, is given by $R_1 \times R_2^{\,2} \times R_3^{\,4} \times R_4 \times R_n$. Thus, the objective here is to estimate the reliability of a system by averaging over all path reliabilities [16].

2.1. Reviews of some testing-effort functions

In the field of software reliability modeling, Yamada et. al. [6] adopted the concept of testing-effort within an NHPP model to get a better description of the software fault phenomenon. The testing-effort can be measured by the man power, the number of test cases, the number of CPU hours,..., etc. Furthermore, if the number of faults detected by the current testing-effort expenditures is proportional to the number of remaining faults, then we we have

$$m(t) = a(1 - e^{-r(W(t)-W(0))}) \qquad (1)$$

where $m(t)$=the number of faults detected in time $(0, t)$,
$W(t)$= the testing-effort consumption in time $(0, t)$,
a = the expected number of initial faults,
r = the error detection rate per unit testing

2.2. A new approach to representing the effects of weighting on the components

Proceedings of the 26 th Annual International Computer Software and Applications Conference (COMPSAC'02)
0730-3157/02 $17.00 © 2002 IEEE

From [15], we know the fact that a superposition or a decomposition of independent Poisson processes is also a Poisson process. Consider a series system of n independent components. Assume that the system failure intensity is $\lambda_s(t)$ and the failure intensity of component i is $\lambda_i(t)$, i=1,…,n, the relationship between $\lambda_s(t)$ and $\lambda_i(t)$ is given as follows:

$$\lambda_s(t) = \lambda_1(t) + \lambda_2(t) + ... + \lambda_n(t). \quad (2)$$

Xie et. al. [16] proposed an additive model for assessing the reliability of this type of integrated system. Although the approach of the additive model is straightforward, there are some potential drawbacks. For example, Eq. (2) reveals that each component has equal influence on the overall system no matter how often it executes. In practice, the frequency of a component being executed affects the overall system reliability. A higher frequency indicates a greater effect of that component on the performance of the system. This fact shows that the components should have distinct weights according to the architecture of the software system. Consequently, we propose a new approach to representing the effects of weighting on the components. This means that we add a weight vector to Eq. (2) and the new solution is as follows:

$$\lambda_s(t) = w_1\lambda_1(t) + w_2\lambda_2(t) + ... + w_n\lambda_n(t) \quad (3)$$

where the vector of weights (w_1, w_2, ..., w_n) represents the importance of each component and is obtained based on the system architecture.

2.3. Markov process to model the architecture of a component-based system

In this section, we consider systems with different architecture styles and utilize the Markov process to model the failure behaviors of the applications. Three general input-output cases were employed. In addition, we develop three methodologies to estimating the reliability of a software system.

Definition : Let $\{X_n, n=0, 1, 2\ldots\}$ be a Markov process with some absorbing states and some transient states. Define the random variable, N_{ij}, to represent the number of visits to state j before entering an absorbing state given $X_0=i$. The expected value of N_{ij} , $E(N_{ij})$, is denoted by μ_{ij} . Moreover, let η_k denote the probability of absorption when a process terminates at an absorbing state k. Furthermore, let the probability of reaching state k from state i in n steps be denoted as f_{ik}^n . □

The proofs of Theorem 1 and Theorem 2 are similar to Theorem 3 and are omitted.

Theorem 1 (single-input/single-output system): Consider a single-input and single-output system consisting of N components with reliabilities R_1,…, R_N. Let $\{X_n\}$ be the Markov process where state N is an absorbing state, i.e., an output node, while states {1, 2, ..., N–1} are transient states. In particular, assume state 1 is the input node. Therefore, we have the reliability of the system:

$$R_s = R_1 \times R_N \times \prod_{i=2}^{N-1} pow(R_i, \mu_{1i}),$$

where the $pow(x, y)$ is the power function, namely, $pow(x, y)=x^y$. □

Theorem 2 (single-input/multiple-output system): Consider a single-input and r-output system consisting of N components with individual reliabilities denoted by R_1,…, R_N.. Let $\{X_n\}$ be the Markov process where {N, N–1, ..., N–r+1} are absorbing states (i.e. r output nodes) and {1, 2, ..., N–r} are transient states. In particular, assume state 1 is the input node. Therefore, we have the reliability of the system:

$$R_s = R_1 \times \prod_{i=2}^{N-r} pow(R_i, \mu_{1i}) \times \prod_{k=N-r+1}^{N} pow(R_j, \eta_j). \quad \square$$

Theorem 3 (multiple-input/multiple-output system): Consider an s-input and r-output system consisting of N components with reliabilities R_1,…, R_N.. Let $\{X_n\}$ be a Markov process where {N, N–1, ..., N–r+1} are absorbing states (i.e. r output nodes) and {1, 2, ..., N–r} are transient states. In particular, assume states {1, 2, ..., s} are the input nodes with probability p_1, p_2, ..., p_s, respectively. Therefore, the system reliability, R_s , equals

$$\prod_{i=1}^{s} pow(R_i, p_i) \times \prod_{j=s+1}^{N-r} pow(R_j, \sum_{l=1}^{s} p_l\mu_{lj}) \times \prod_{k=N-r+1}^{N} pow(R_k, \eta_k)$$

Proof:

(1) The transformation probability matrix can be written:

$$P = \begin{bmatrix} I_{r\times r} & 0 \\ R_{(N-r)\times r} & Q_{(N-r)\times(N-r)} \end{bmatrix}_{N\times N}$$

where Q is the transformation probability matrix corresponding to the transient states, $R_{(N-r)\times r}$ is the matrix of transition probabilities from transient to recurrent state, and I is an identity matrix of size r.

(2) Next, we focus on μ_{ij} where $N - r \geq i, j \geq 1$. By the definition of μ_{ij} and the conditional probability on X_1, we can have μ_{ij} as follows:

$E(N_{ij}|X_1=1)\Pr(X_1=1|X_0=i)+..+E(N_{ij}|X_1=N-r)\Pr(X_1=N-r|X_0=i)+$
$E(N_{ij}|X_1=N-r+1)\Pr(X_1=N-r+1|X_0=i)+..+E(N_{ij}|X_1=N)\Pr(X_1=N|X_0=i)$

Because {N, ..., N–r+1} are absorbing states, it indicates that the mathematical terms, $E(N_{ij} \mid X_1=K)\Pr(X_1=K \mid X_0=i)$ where $N \geq K \geq N - r + 1$, equal to one. That is,

$$\mu_{ij} = \delta_{ij} + \sum_{k=1}^{N-r} \mu_{kj}P_{ik} \quad \text{where } \delta_{ij} = \begin{cases} 1 & if \ \ i = j \\ 0 & if \ \ i \neq j \end{cases} \quad (4)$$

Equivalently, we can transform Eq. (4) into a matrix form:

M=I+QM, and get M=(I–Q)$^{-1}$where *M* is composed of μ_{ij}.

(3) Assume that the starting state *i* is the transient state and the ending state *j* is the absorbing states. Therefore, we have $f_{ij}^1 = R_{ij}$ and $f_{ij}^n = \sum_{k=1}^{N-r} Q_{ik} f_{kj}^{n-1}$. Equivalently, we can have $\eta_i = ((I-Q)^{-1}_{(N-r)\times(N-r)} \times R_{(N-r)\times r})_{1i}$. Therefore, in this case, the process may be absorbed from one of the input nodes. Thus we have the result for multiple inputs:

$$\eta_i = \sum_{j=1}^{s} p_j \times ((I-Q)^{-1}_{(N-r)\times(N-r)} \times R_{(N-r)\times r}).$$

(4) At last, we can conclude that the nodes *i* ($s \ge i \ge 1$) will have p_i visits, the node *i* ($N-r \ge i \ge s+1$) have $\sum_{l=1}^{s} p_l \mu_{li}$ visits on average, and the output nodes *i* ($N \ge i \ge N-r+1$) have η_i visits. Therefore, the system reliability , R_s, equals

$$\prod_{i=1}^{s} pow(R_i, p_i) \times \prod_{j=s+1}^{N-r} pow(R_j, \sum_{l=1}^{s} p_l \mu_{lj}) \times \prod_{k=N-r+1}^{N} pow(R_k, \eta_k)$$

□

3. Optimum testing-effort allocation problems

In this section we describe a general problem of allocating testing resources to software components so that the software applications can be constructed effectively, given that the applications have prescribed reliability requirements [5-9]. Furthermore, we consider two testing-effort allocation problems [5-8] based on the proposed model:

(1)minimizing the number of software faults remaining in the system given fixed amount of testing-effort,

(2)minimizing the total testing-effort given the fixed reliability requirements.

3.1. Minimizing the number of remaining faults

Suppose that each application is distributed with a limited amount of testing-effort over its components, and component *i* is allotted W_i testing-effort, and thus the optimization problem can be represented as follows:

Minimize $\sum_{i=1}^{N} v_i a_i \exp(-r_i W_i)$, subject to the requirements:

$$\alpha_{11} W_1 + \alpha_{12} W_2 + ... + \alpha_{1N} W_N \le \varepsilon_1$$

…

$$\alpha_{M1} W_1 + \alpha_{M2} W_2 + ... + \alpha_{MN} W_N \le \varepsilon_M$$

$$\alpha_{ij} \ge 0,\ \varepsilon_i \ge 0,\ W_j \ge 0,\ i = 1,2,...,M,\ j = 1,2,...,N$$

Note that the parameters v_i, a_i, r_i and α_{ij} have already been estimated by the proposed model and ε_i is the limited amount of testing-effort available for components used by application *i*.

3.2. Minimizing the total testing-effort

On the other hand, suppose the applications have pre-specified reliability requirements, one has to allocate an amount of testing-effort to each component to minimize the total testing-effort such that all applications meet their reliability requirement. Therefore, the optimization problem can be represented as follows:

Minimize $\sum_{i=1}^{N} W_i$, subject to the requirements:

$$\beta_{11} \exp(-r_1 W_1) + \beta_{12} \exp(-r_2 W_2) + ... + \beta_{1N} \exp(-r_N W_N) \le \gamma_1$$

…

$$\beta_{M1} \exp(-r_1 W_1) + \beta_{M2} \exp(-r_2 W_2) + ... + \beta_{MN} \exp(-r_N W_N) \le \gamma_M$$

$$\beta_{ij} \ge 0,\ \gamma_i \ge 0,\ W_j \ge 0,\ i = 1,2,...,M,\ j = 1,2,...,N$$

Note that the parameters r_i and β_{ij} have already been estimated by the proposed model and γ_i is the reliability requirement used by application *i*.

3.3. Results for single application environment

3.3.1. Minimizing the number of remaining faults given fixed amount of testing-effort. The optimization problem is that the total amount of testing-effort is fixed, and we want to allocate these efforts to each component to maximize the system reliability. Suppose the total amount of testing-effort is *W*, and component *i* is allotted W_i testing-effort, and thus the optimization problem can be represented as follows:

Minimize: $\sum_{i=1}^{N} v_i a_i \exp(-r_i W_i)$, subject to

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

Note that the parameters a_i and v_i have already been estimated by the proposed model in Section 2. To solve the above problem, the Lagrange multiplier method [20] can be applied. Furthermore, we propose a simple optimization algorithm to solve the above problem.

Algorithm 1 for minimizing the number of remaining faults

Step 1: Set *l*=0.

Step 2: Calculate the following equations where *i*=1,…,*N-l*.

$$\ln \lambda = \frac{\sum_{i=1}^{N-l} (1/r_i)(\ln v_i a_i r_i) - W}{\sum_{i=1}^{N} (1/r_i)}, W_i = \frac{1}{r_i}[\ln(v_i a_i r_i) - \ln \lambda].$$

Step 3: Rearrange the index *i* such that $W_1^* \ge ... \ge W_{N-l}^*$.

Step 4: If $W_{N-l}^* \ge 0$ then stop, else update $W_{N-l}^* = 0$ and *l*=*l*+1.

Step 5: Go to Step 2. □

3.3.2. Minimizing the total testing-effort given the fixed reliability requirement. On the other hand, suppose the

number of remaining faults in the system is specified by γ, one has to allocate an amount of testing-effort to each component to minimize the total testing-effort. Therefore, the optimization problem can be represented as follows:

Minimize $\sum_{i=1}^{N} W_i$, subject to $\sum_{i=1}^{N} v_i a_i \exp(-r_i W_i) = \gamma,\ W_i \geq 0$

We also propose a simple optimization algorithm to solve it.

Algorithm 2 for minimizing the total amount of testing-effort expenditures:

Step 1: Set l=0.

Step 2: Calculate $W_i = \frac{1}{r_i}[\ln(\frac{v_i a_i r_i}{\gamma} \sum_{i=1}^{N-l} \frac{1}{r_i})], i = 1,2,...,N-l.$

Step 3: Rearrange the index i such that $W_1^* \geq ... \geq W_{N-l}^*$.

Step 4: If $W_{N-l}^* \geq 0$ then stop, else update $W_{N-l}^* = 0$ and l=l+1.

Step 5: Go to Step 2. □

4. Numerical examples

4.1. Reliability evaluation of component-based systems

The following examples adapted from [2, 11] are used to illustrate the three architecture cases discussed in Section 2. Without loss of generality, we use the terminating application reported in [11] as a running example and let the estimated reliabilities of the components be regarded as unchanged throughout the following three subsections and listed in Table 1.

4.1.1. Example 1: a single-input/single-output system.

The first example is a single-input/single-output system. It consists of 10 components where component 1 is the input component and component 10 the output component. Figure 1 depicts the control-flow graph of the example, and the transition probabilities among the components are given as follows: $P_{1,2} = 0.6$, $P_{1,3} = 0.2$, $P_{1,4} = 0.2$, $P_{2,3} = 0.7$, $P_{2,5} = 0.3$, $P_{3,5} = 1.0$, $P_{4,5} = 0.4$, $P_{4,6} = 0.6$, $P_{5,7} = 0.4$, $P_{5,8} = 0.6$, $P_{6,3} = 0.3$, $P_{6,7} = 0.3$, $P_{6,8} = 0.1$, $P_{6,9} = 0.3$, $P_{7,2} = 0.5$, $P_{7,9} = 0.5$, $P_{8,4} = 0.25$, $P_{8,10} = 0.75$, $P_{9,8} = 0.1$, $P_{9,10} = 0.9$. Therefore, the expected number of visits on each transient state before absorption from the input node (component 1) and the probability of absorption can be derived as follows:

$\mu_{11} = 1, \mu_{12} = 1.4717, \mu_{13} = 1.3254, \mu_{14} = 0.5289, \mu_{15} = 1.9784,$
$\mu_{16} = 0.3173, \mu_{17} = 1.7433, \mu_{18} = 1.3155, \mu_{19} = 0.9669, \eta_{10} = 1.$

Thus, the system reliability is estimated as R_1= 0.7715.

Table 1: The estimated reliabilities of the components.

1	2	3	4	5	6	7	8	9	10
0.99	0.98	0.99	0.96	0.98	0.95	0.98	0.96	0.97	0.99

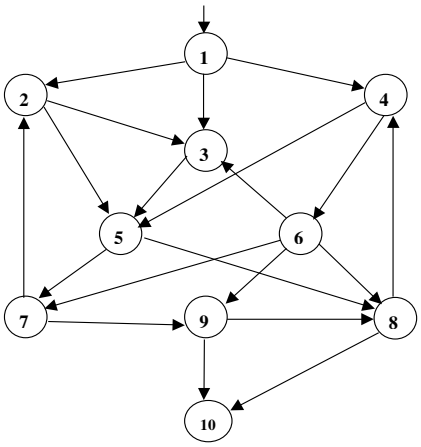

Figure 1: A single-input/single-output system.

4.1.2. Example 2: a single-input/multiple-output type.

In this example, we delete two links of the original program control graph in Example 1, and obtain the modified graph as shown in Figure 2. The modification is a simple transformation from a single-output system to a multiple-output system and the corresponding transition probabilities are similar to Example1.

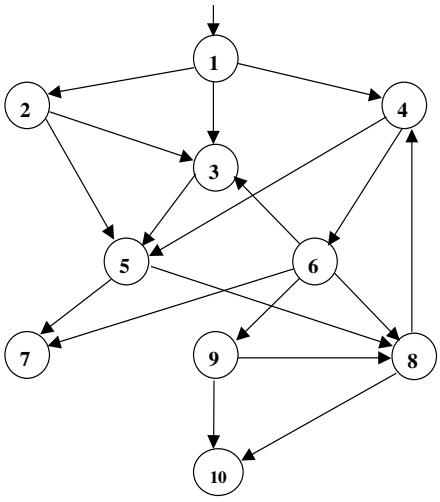

Figure 2: A single-input/multiple-output system.

Therefore, following the same approach we can have following results: $\mu_{11} = 1, \mu_{12} = 0.6, \mu_{13} = 0.6845,$
$\mu_{14} = 0.3581, \mu_{15} = 1.0077, \mu_{16} = 0.2149, \mu_{18} = 0.6326,$
$\mu_{19} = 0.0645,\ \eta_7 = 0.4676, \eta_{10} = 0.5324.$ Thus, the reliability of the application, R_2, is 0.8890.

4.1.3. Example 3: a multiple-input/multiple-output type.

In this example, the process will start from one of the two

input components (components 1 and 2) with equal probability and terminates at the output components (components 7 and 10). That is, the modification is a transformation from a single-input system to a multiple-input system. Figure 3 depicts the control-flow graph of the example and the transition probabilities are similar to Example 2 except $P_{1,3} = 0.5$ and $P_{1,4} = 0.5$.

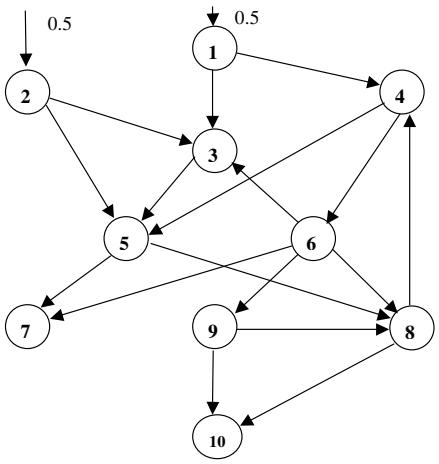

Figure 3: A multiple-input/multiple-output system.

Therefore, according to Theorem 3, portion of the vector of weights in Eq. (3) can be obtained by $w_k = \sum_{l=1}^{2} p_l \mu_{lk}$. Therefore, we have ($w_1$, w_2, w_3, w_4, w_5, w_6, w_8, w_9) = (0.5, 0.5, 0.673, 0.4057, 0.9853, 0.2434, 0.6228, 0.073). On the other hand, with the aim to computing the probability of absorption at each absorbing state, the following information about the two absorbing states is obtained based on Theorem 3: $w_7 = 0.4672, w_{10} = 0.5327$. Thus, we have the reliability of the system is R_3= 0.8929.

4.2. Optimum testing-effort allocation problems

In this subsection, three numerical examples for the optimum testing-effort allocation problem are demonstrated. In particular, assume the components are adopted from [6] and the estimated parameters a_i, r_i, in Section 3.3.1, for i=1,…, 10, in the software system are summarized in Table 2. Moreover, the control flows of the three systems of interest are similar to the cases in Section 3 and the weighting vector v_i in Eq. (3), for i=1,…, 10, are also listed in Table 2.

4.2.1. Minimizing the number of remaining faults. Assume the total amount of testing-effort expenditures W is 50,000. One has to allocate the expenditures to each component to minimize the number of remaining faults. Using the algorithm 1 in Section 4.1, the optimal testing-effort expenditures for three systems are estimated and shown in Table 3. Furthermore, the number of initial faults and the number of remaining faults estimated for three examples are also shown in Table 4.

Table 2: The estimated values of a, r_i and v_i.

	a_i	$r_i(10^{-4})$	v_i in Example1	v_i in Example 2	v_i in Example 3
1	89	4.1823	1	1	0.5
2	25	5.0923	1.4717	0.6	0.5
3	27	3.9611	1.3254	0.6845	0.6730
4	45	2.2956	0.5289	0.3581	0.4057
5	39	2.5336	1.9784	1.0077	0.9853
6	39	1.7246	0.3173	0.2149	0.2434
7	59	0.8819	1.7433	0.4676	0.4672
8	68	0.7274	1.3155	0.6326	0.6228
9	37	0.6824	0.9669	0.0645	0.073
10	14	1.5309	1	0.5324	0.5327

Table 3: The optimal solution using Algorithm1.

	W_i^* for example 1	W_i^* for example 2	W_i^* for example 3
1	6215	8112	6512
2	3756	3552	3241
3	4125	4459	4477
4	2964	4721	5396
5	7718	8261	8192
6	0	835	1697
7	13465	7538	7800
8	11757	1598	12713
9	0	0	0
10	0	0	0

Table 4: The reduction in the number of faults.

	Initial faults	Remaining faults	Reduction (%)
Example 1	517.0	173.6	33.6
Example 2	266.7	67.5	25.3
Example 3	221.4	66.4	30.0

Table 5: The optimal solution using Algorithm2.

	W_i^* for example 1	W_i^* for example 2	W_i^* for example 3
1	7700	6954	5340
2	4976	2602	2278
3	5692	3237	3239
4	5669	2612	3233
5	10168	6275	6256
6	2096	0	0
7	20505	2046	2240
8	20293	5943	5971
9	7265	0	0
10	2388	0	0

4.2.2. Minimizing the total testing-effort expenditures. Assume the total number of remaining faults Z 100. One has to allocate the expenditures to each component to minimize the total amount of testing-effort expenditures. Using algorithm 2 in Section 4.2 and Table 2, the optimal solutions for Section 3.3.2 are derived and shown in Table 5. Furthermore, the relationship between the total amount of testing-effort expenditures and the reduction rate of remaining faults is depicted in Figure 4.

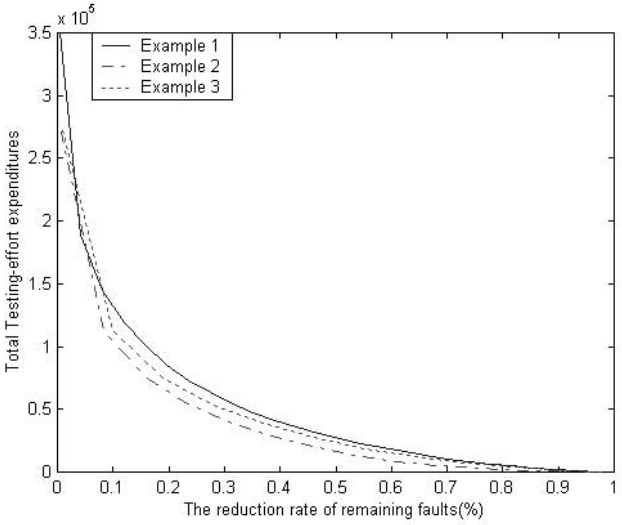

Figure 4: The reduction rate of remaining faults v.s. the total testing-effort expenditures.

5. Conclusions

This paper presents a new approach to analyzing the reliability of a component-based software, based on the reliabilities of the individual components and the architecture of the system. Furthermore, we derive some useful mathematical properties to show that the model is indeed very powerful. Three general cases are utilized to validate the proposed approach. On the other hand, two testing-effort allocation problems are also studied and efficient solutions are provided. Experimental results show that the proposed methods can solve the testing-effort allocation problems and improve the quality and reliability of the software system.

Acknowledgment

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.

References

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

[2] S. S. Gokhale, "Analysis of Software Reliability and Performance," *Ph.D. Dissertation*, Department of Electrical and Computer Engineering, Duke University, Durham, 1998.

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

[4] J. D. Musa (1998). *Software Reliability Engineering: More Reliable Software, Faster Development and Testing*. McGraw-Hill.

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

[6] H. Ohtera and S. Yamada, "Optimal Allocation & Control Problems for Software-testing Resources," *IEEE Trans. on Reliability*, 39, vol. 2, pp. 171-176, June 1990.

[7] Y. W. Leung, "Software Reliability Growth Model with Debugging Efforts," *Microelectron. Reliab.* Vol. 32, No. 5, pp. 699-704, 1992.

[8] 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 Trans. on Reliability*, Vol. 45, No. 4, pp. 541-549, December 1996.

[9] Michael R. Lyu and Aad P. A. van Moorsel, "Optimization of Reliability Allocation and Testing Schedule for Software Systems," *Proceedings of the 8th International Symposium on Software Reliability Engineering*, pp. 336-347, November 1997, Los Almitos, California.

[10] C. Y. Huang, J. H. Lo, S. Y. Kuo, and Michael R. Lyu, "Software Reliability Modeling and Cost Estimation Incorporating Testing-Effort and Efficiency," *Proceedings of the 10th International Symposium on Software Reliability Engineering*, pp. 62-72, November 1999, Florida.

[11] R. C. Cheung, "A User-Oriented Software Reliability Model," *IEEE Trans. on Software Engineering*, SE-6(2), pp. 118-125, March 1980.

[12] W. L. Wang, Y. Wu, and M. H. Chen, "An Architecture-Based Software Reliability Model," *Proceedings of the Pacific Rim International Symposium on Dependable Computing*, pp. 143-150, Dec. 1999, HongKong.

[13] S. Krishnamurthy and A. P. Mathur, "On the Estimation of Reliability of a Software System using Reliabilities of its Component", *Proceedings of the 8th International Symposium on Software Reliability Engineering*, pp. 146-155, November 1997, Albuquerque, New Mexico.

[14] B. Littlewood, "Software Reliability Model for Modular Program Structure", *IEEE Trans. on Reliability*, vol. 28, No. 3, pp. 241-246, August 1979.

[15] K. S. Trivedi. Probability and Statistics with Reliability, Queuing and Computer Science Applications. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[16] M. Xie and C. Wohlin, "An additive Reliability Model for the Analysis of Modular Software Failure Data," *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pp. 188-194, October 1995, Toulouse, France.

[17] M. S. Bazaraa and C. M. Shetty, Nonlinear Programming: Theory and Algorithm, John Wiley & Sons, 1993.