Optimal Allocation of Test Resources for Software Reliability Growth Modeling in Software Development

Michael R. Lyu, *Senior Member, IEEE*, Sampath Rangarajan, *Member, IEEE*, and Aad P. A. van Moorsel, *Member, IEEE*

***Abstract*—Component-based software development approach has become a trend in integrating modern software systems. To ensure the overall reliability of an integrated software system, its software components have to meet certain reliability requirements, subject to some testing schedule and resource constraints. Efficiency improvement of the system-testing can be formulated as a combinatorial optimization problem with known cost, reliability, effort, and other attributes of the system components. This paper considers “software component testing resource allocation” for a system with single or multiple applications, each with a pre-specified reliability requirement. The relation between failure rates of components and “cost to decrease this rate” is modeled by various types of reliability-growth curves. Closed-form solutions to the problem for systems with one single application are developed, and then “how to solve the multiple application problem using nonlinear programming techniques” are described. Also examined are the interactions between the system components, and inter-component failure dependencies are included in the modeling formula. In addition to regular systems, the technique is extended to address fault-tolerant systems. A procedure for a systematic approach to the testing resource allocation problem is developed, and its application in a case study of a telecommunications software system is described. This procedure is automated in a reliability allocation tool for an easy specification of the problem and an automatic application of the technique.**

This methodology gives the basic approach to optimization of testing schedules, subject to reliability constraints. This adds “interesting new optimization opportunities in the software testing phase” to the existing optimization literature that is concerned with structural optimization of the software architecture. Merging these two approaches improves the reliability planning accuracy in component-based software development.

***Index Terms*—Component-based technology, reliability allocation, software reliability engineering, software testing.**

NOTATION

abs$(\cdot)$	absolute function
A_i	application i, $1 \le i \le M$
c_j	coverage measure for C_j
C_j	component j, $1 \le j \le N$
d_i	fixed amount of testing time allowed for A_i
D_j	the testing time invested in C_j
D	total testing-time invested
$f_j(\lambda_j)$	function relating (λ_j) to testing time
M	number of software applications
N	number of software components
$R_i(t)$	pre-specified reliability requirement for A_i
w_j	weighting function for D_j in the total testing time
δ_i	sum of failure rates of the components in A_i
δ	fixed total failure-rate constraint
Δ	partial derivative operator
$\epsilon_{j,k}$	$k = 0, 1, 2$: parameters in the Pareto function with respect to C_j
$\lambda_{j,0}$	initial failure rate in C_j at time 0
λ_j	failure rate in C_j during testing
μ_j	failure decay parameter for C_j during testing
θ	Lagrange multiplier
ρ_j	$1 - c_j$
$\sigma_{i,j}$	usage indicator for A_i on C_j.

Manuscript received October 15, 1999; revised May 1, 2001. This work was supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project CUHK 4222/01E). The described research was initially carried out at Lucent Bell Labs.

M. R. Lyu is with the Computer Science and Engineering Department, The Chinese University of Hong Kong, Shatin, Hong Kong (e-mail: Lyu@cse.cuhk.edu.hk).

S. Rangarajan is with Ranch Networks, Morganville, NJ 07751 USA (e-mail: Sampath@ranchnetworks.com).

A. P. A. van Moorsel is with the Software Technologies Laboratory, Hewlett-Packard Labs, Palo Alto, CA 94304 USA (e-mail: Aad@hpl.hp.com).

Publisher Item Identifier S 0018-9529(02)05620-8.

I. INTRODUCTION

A. Background

MODERN complex software-systems are often developed with components supplied by contractors or independent teams under various environments. In particular, component-based software engineering [4], [14] has drawn tremendous attention in developing cost-effective and reliable applications to meet short time-to-market requirements. For systems integrated with such modules or components, the system-testing problem can be formulated as a combinatorial optimization problem with known cost, reliability, effort, and other attributes of the system components. The best known system-reliability problem of this type is the series-parallel redundancy allocation problem, where either system reliability is maximized or total system testing cost/effort is minimized. Both formulations generally involve system-level constraints on allowable cost, effort, and/or minimum system-reliability levels. This series-parallel redundancy-allocation problem has been widely studied for hardware-oriented systems with the approaches of dynamic programming [9], [21], integer programming [3], [10], [18], nonlinear optimization [24], and heuristic techniques [5], [22]. In [7] the optimal apportionment of reliability and redundancy is considered for multiple objectives using fuzzy optimization techniques. Reference [6] applies a specific reliability-growth model for hardware components and determines their optimal testing allocation in order to achieve an overall system reliability.

Some researchers also address the reliability-allocation problem for software components. The software reliability-allocation problem is addressed in [26] to determine how reliable software modules and programs must be to maximize the user's utility, subject to cost and technical constraints. Optimization models for software-reliability allocation for multiple software programs are further proposed in [2] using redundancies. These papers, however, do not consider testing-time of software components and the growth of their reliability. Optimal allocation of component-testing times in a software system based on a particular software reliability model is addressed in [17], but it assumes a single application in the system, and the reliability-growth model is limited to the Hyper-Geometric Distribution (S-shaped) Model [25].

This paper discusses a generic software-component reliability-allocation problem based on several types of software-reliability models in a multiple-application environment. This is the first effort to apply reliability-growth models for guiding component testing based on multiple applications. The solution procedure is given for the single application environment, for general continuous distributions, thus generalizing [5] and [17]. The situation is examined where software components can interact with each other, a condition not considered by other studies. Also included are scenarios for fault-tolerant attributes of a system where some component failures can be tolerated. The reliability specification and solution-seeking procedure, which has been automated by a software tool, is presented as an innovative mechanism to handle the difficult, important testing-resource allocation problem.

B. Project Applications

Several real projects on component-based techniques motivate this investigation. They are described in the following three case studies.

1) Distributed Software Systems: Distributed telecommunication systems often serve multiple application types, by executing various software components to meet various reliability requirements. For instance, in telephone switches, 1-800 calls require a processing reliability that is different from standard calls; similar examples exist in call centers, PBXs, or voice-mail systems.

During the testing of such systems, reliability is a prime concern, and adequate test and resource allocation are therefore very important. The examples in this paper make it clear that trustworthy reliability-growth curves can help considerably in efficient testing and debugging planning of such systems. The approach complements [27] and [28] which focus on reliability analysis of component-based software systems under various distributed execution scenarios.

2) Fault-Tolerant Systems: Fig. 1 shows a layered fault-tolerant software-architecture model that has been applied to many systems. Each layer can include several software components. Not all systems include all the layers.

This paper conjectures that error-propagation between layers occurs only in one direction: upwards. Thus, for example, faults not contained in the hardware can propagate up to the operating system or to the application software; however, faults not contained in the middle-ware layer can not propagate to the operating system but can propagate to the application software layer.

Error propagation occurs from layer i to layer $i+1$ if layer i has no fault-tolerance mechanisms: it does not exhibit a fail-silent behavior. From a modeling perspective, this layer contributes higher failure rate to the system than a layer with error detection and recovery mechanisms. Error detection and recovery software can reside in some or all of the layers.

This paper shows how fault-tolerant mechanisms can be included in the problem formulation for reliability-specification and resource-allocation, provided that coverage factors are available.

3) Object-Oriented Software: Object-oriented software often allows for a clear delineation between different software components. If object-oriented software methods are being used, the relation between components and applications can be assessed, and testing time can be assigned in the most efficient way.

Another optimization problem in object-oriented software testing arises when the best combination of objects must be selected to make an application as reliable as possible. This optimization problem is an example of a structure-oriented optimization problem, and can be solved by using methods in, e.g., [2], [26]. The intent of this paper is to optimize with respect to software development and testing, not with respect to software structure. The combination of structure-oriented optimization methods in [2] and [26], and the development-oriented methods in this paper can provide a powerful mechanism in component-based and object-based software system design.

C. Paper Organization

Section II specifies the optimal reliability allocation as two related problems:

- a problem with fixed target failure rate,
- a problem with fixed debugging time.

Section III presents the analytic solutions to these two problems for the single application environment: Section III-A for the exponential distribution, and Section III-B for general distributions.

Section IV discusses how the solutions for the problems in a multiple-application environment can be obtained. The results are extended to consider software-failure dependencies and to incorporate fault-tolerant systems.

Section V proposes the reliability-allocation problem specification and solution procedure into a step-by-step framework, and applies it to a case study. The systematic application of the reliability allocation framework is designed and implemented in an automatic software tool.

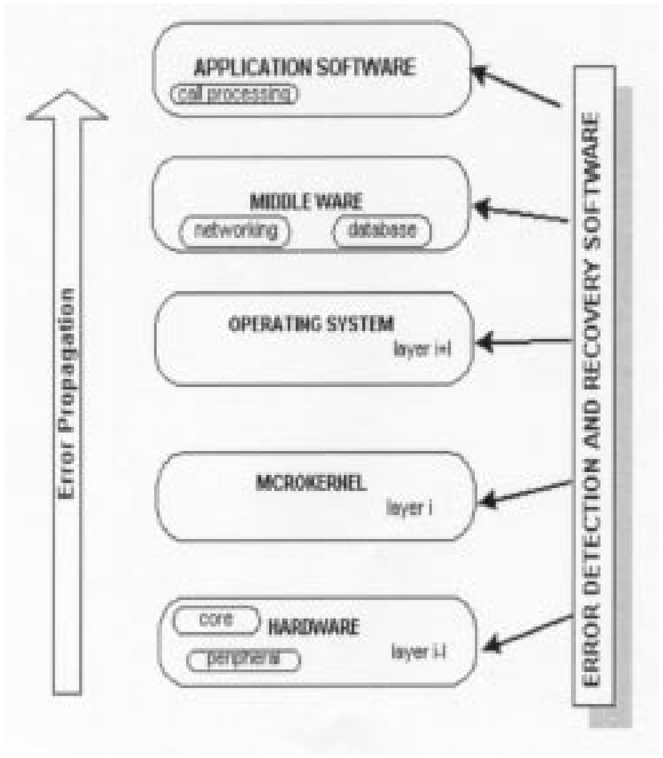


Fig. 1. Layered software architecture model.

II. Problem Specification

The project case studies in Sections I and II can be described as a general problem of assigning failure-rate requirements (at the time of release) to software components that will be used to build various applications, given that the applications have pre-specified reliability requirements [16].

Consider the situation where a set of N software components $C_1, \ldots, C_N$, can be used in various combinations for various applications.

- Let there be M such applications: $A_1, \ldots, A_M$.
- Let each application have a pre-specified reliability requirement $R_1(t), \ldots, R_M(t)$.

By investing development/testing/debugging time in components, then component failure rates can be made such that all applications meet their reliability requirement.

Therefore, the goal in reliability allocation is to assign failure-rate requirements to the N components, such that all the pre-specified reliability requirements of the M applications are satisfied, at minimal cost. The remainder of this Section II characterizes the cost in terms of component testing (including debugging) time. The optimization criterion thus is the minimization of this testing time. Reliability-growth models relate the component failure-rates to the amount of testing time.

A variation of this problem formulation arises if a fixed amount of testing time is available for each application. This requirement can occur because of the constraint on the cost incurred by the component developer and tester. In that case, "minimization of the failure rate of all the components" is the objective of the optimization problem. These 2 variations of the optimization problem are discussed in Sections II-A and II-B.

A. Fixed Failure Rate Constraint

Testing time is assigned to components so that the applications meet their reliability requirements, and the testing time is minimized.

Assumptions:

1) The failure rates of components relate to the reliability of applications through the exponential distribution: $R_i(t) = \exp(-\delta_i \cdot t)$.
2) The testing time, D_j, invested in component j decreases the failure rate λ_j according to some reliability-growth model.
3) Once the software components are released, their failure rates stay constant. (This is reasonable given that the application developer does not debug or change the component that is used.)
4) Components are s independent with respect to their failure behavior. (This assumption might not be appropriate when software components can interact with each other, potentially causing additional failures.)

The allocation problem is formulated as: The objective function is:

Minimize

$$D = \sum_{j=1}^{N} D_j, \tag{1}$$

subject to the constraints:

$$\sum_{j=1}^{N} \sigma_{i,j} \cdot \lambda_j \le \delta_i \quad \text{(for application } A_i\text{)}, \qquad i = 1, \ldots, M$$

$$\sigma_{i,j} = 1 \qquad \text{if } A_i \text{ uses } C_j,\ \sigma_{i,j} = 0 \text{ otherwise.}$$

$$\lambda_j \ge 0; \qquad D_1, \ldots, D_N \ge 0;\ \delta_1, \ldots, \delta_M \ge 0.$$

For the sake of notational simplicity, the D_j are equally "important" (costly) among the N components; if this is not true, then apply w_j to each D_j in the objective function.

Because a reliability-growth curve can be very complex, the objective function is nonlinear; hence this is a general nonlinear programming problem. Section III-A-1 considers a closed-form solution for the problem with a single application; Section IV-A discusses the numerical solution of the general case.

B. Fixed Testing-Budget

Testing time is assigned to components so that each application gets assigned at most a specified amount, and the application reliability is maximized. Consequently, the total failure rate of the components is the objective function to be minimized, leading to the following formulation as a mathematical programming problem. The objective function is:

Minimize

$$\lambda \equiv \sum_{j=1}^{N} \lambda_j, \tag{2}$$

subject to the constraints:

$$\sum_{j=1}^{N} \sigma_{i,j} \cdot D_j \le d_i, \qquad \text{for } A_i,\ i = 1, \ldots, M.$$

As in the problem in Section II-A, all variables are positive. Weight functions for the λ_j can be introduced in the objective function to reflect their impact.

Of special interest is the single application case which corresponds to the optimization problem where all applications together have a budget restriction on the testing time. The fixed testing-budget problem is a variant of the fixed failure-rate problem, and can be solved by similar means. Sections III-A-2 and IV-A discuss the single and multiple applications, respectively.

III. Solutions for Single Application Environment

When there is only one application in the system, an explicit solution of the reliability-allocation problem can usually be found. Section III-B gives the general solution for a large class of reliability-growth models. To explain the solution procedure, Section III-A give the solution for an exponential reliability-growth model.

A. *Exponential Reliability-Growth Model*

The exponential reliability growth model [11], [20] relates the λ_j to the invested D_j:

$$\lambda_j = \lambda_{j,0} \cdot \exp(-\mu_j \cdot D_j). \tag{3}$$

Over an infinite time interval, $\lambda_{j,0}/\mu_j$ faults are found. The λ_j is a function of time, although it is not explicitly in the notation. For this commonly-used reliability growth model, the allocation problem in the single application environment is solved.

1) Fixed Failure-Rate Constraint: This problem is formulated in the single application environment, assuming exponential reliability growth curves:

Minimize

$$D = \sum_{j=1}^{N} \frac{1}{\mu_j} \cdot \log\left(\frac{\lambda_{j,0}}{\lambda_j}\right), \tag{4}$$

subject to:

$$\sum_{j=1}^{N} \lambda_j \le \delta.$$

To solve (3), one can use the Lagrange method [1]. The optimization problem is equivalent to finding the minimum of:

$$F(\lambda_1, \ldots, \lambda_N) = D + \theta \cdot \left[\left(\sum_{j=1}^{N} \lambda_j\right) - \delta\right]. \tag{5}$$

The solution is:

$$\lambda_1 = \frac{\delta}{1 + \cdot \sum_{j=1}^{N} \frac{\mu_1}{\mu_j}},$$

$$\lambda_j = \frac{\mu_1}{\mu_j} \cdot \lambda_1, \qquad j = 2, \ldots, N. \tag{6}$$

The testing times allotted to the software components follow from substituting the values in (6) into (4), e.g.,

$$D_1 = \frac{1}{\mu_1} \cdot \log\left(\frac{\lambda_{1,0}}{\lambda_1}\right). \tag{7}$$

D_1 is negative if $\lambda_1 \ge \lambda_{1,0}$. To assure that no impossible solutions arise, Section III-B presents a procedure that checks for validity conditions and guarantees that the optimal solution follows a valid strategy.

Example 1: The system has 3 components, C_1, C_2, C_3, and 1 application A which uses all the components. $\lambda_{1,0} = \lambda_{2,0} = \lambda_{3,0} = 5$/year. The application requirement states that $\delta = 6$/year. $\mu_1 = \mu_2 = \mu_3 = 1$. Then $\lambda_1 = \lambda_2 = \lambda_3 = 2$/year and $D_1 = D_2 = D_3 = \log(2.5)$. Thus, when the initial failure rates and the rate of reduction in failure rates with debugging is the same for all the components, then an *average testing policy,* where all the component failure rates are brought down to the same value, provides a solution that meets the application requirement with minimum testing-time; the testing-time for each component is the same.

Example 2: The system has 3 components which have different initial failure rates: $\lambda_{1,0} = 5$/year, $\lambda_{2,0} = 6$/year, $\lambda_{3,0} = 7$/year. $\delta = 6$/year. $\mu_1 = \mu_2 = \mu_2 = 1$. Then $\lambda_1 = \lambda_2 = \lambda_3 = 2$. As in example 1, an *average testing policy* provides a solution that meets the application requirement with minimum testing time spent. But the testing time on each component is different because the initial failure rates are different. $D_1 = \log(5/2)$, $D_2 = \log(6/2)$, $D_3 = \log(7/2)$. The testing time on each component is proportional to the logarithm of the initial failure rate.

Example 3: The system has 3 components which have the same initial failure rates: $\lambda_{1,0} = \lambda_{2,0} = \lambda_{3,0} = 5$/year. $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 3$. $\delta = 6$. Now, computation shows that to minimize the testing time, $\lambda_1 = 6/1.834 = 3.273$, $\lambda_2 = 3/1.834 = 1.636$, $\lambda_3 = 2/1.834 = 1.091$. The optimal testing policy that leads to these failure rates requires a total testing time:

$$\frac{1}{1} \cdot \log\left(\frac{5 \cdot 1.834}{6}\right) + \frac{1}{2} \cdot \log\left(\frac{5 \cdot 1.834}{3}\right) + \frac{1}{3} \cdot \log\left(\frac{5 \cdot 1.834}{2}\right) = 1.49.$$

An *average testing policy* which assigns $\lambda_1 = \lambda_2 = \lambda_3 = 2$ leads to a total testing time of $\log(5/2) \cdot 1.834 = 1.68$ which is more than that of the optimal testing policy.

2) Fixed Testing Budget: The fixed-testing budget-problem can be formulated in the single application environment, assuming exponential reliability-growth curves, as:

Minimize

$$\lambda = \sum_{j=1}^{N} \lambda_j, \tag{8}$$

subject to the constraint

$$\sum_{j=1}^{N} D_j \le D.$$

Again, this can be solved using the Lagrange method, treating the optimization problem as equivalent to finding the minimum of

$$F(D_1, \ldots, D_N) = \lambda + \theta \cdot \left(\sum_{j=1}^{N} D_j - D\right). \tag{9}$$

The solutions are

$$\lambda_{j,0} \cdot [-\mu_j \cdot \exp(-\mu_j \cdot D_j)] \tag{10}$$

are equal for all $j = 1, \ldots, N$.

$$D_1 = \frac{D - \sum_{j=2}^{N} \frac{1}{\mu_j} \cdot \log\left(\frac{\lambda_{j,0}\cdot\mu_j}{\lambda_{1,0}\cdot\mu_1}\right)}{\sum_{j=1}^{N} \frac{\mu_1}{\mu_j}},$$
$$D_j = \frac{1}{\mu_j} \cdot \log\left(\frac{\lambda_j \cdot \mu_j}{\lambda_{0,1} \cdot \mu_1}\right) + \frac{\mu_1}{\mu_j} \cdot D_1, \qquad j = 2, \ldots, N. \tag{11}$$

Equations (9)–(11) determine how the testing times should be allocated to the components. The minimum λ follows directly from the values for the testing times.

Only if the $\lambda_{j,0}$ and μ_j values are independent of j, then an *average testing policy* where the available testing time is equally divided among the components, provides an optimal solution. If either the $\lambda_{j,0}$ or μ_j values are not the same for all j, then (9)–(11) must be computed to obtain an optimal allocation of the testing time.

Example 4: Consider the parameters from example 3 where the initial failure rates are the same ($\lambda_{j,0} = 5$/year, for all j), and the μ_j values are not the same for all j (e.g., $\mu_1 = 1, \mu_2 = 2$, $\mu_3 = 3$). The available testing time is 1 year. Then, $D_1 = 0.1567$, $D_2 = 0.4249$, $D_3 = 0.4184$; and $\lambda_1 = 4.27$/year, $\lambda_2 = 2.14$/year, $\lambda_3 = 1.43$/year, and $\lambda = 7.84$/year.

If an average allocation policy is used, then $D_1 = D_2 = D_3 = 0.333$, and $\lambda_1 = 3.58$/year, $\lambda_2 = 2.57$/year, $\lambda_3 = 1.84$/year for a $\lambda = 7.99$/year which is worse than the optimal allocation.

If the allocation is $D_1 = 0.4$, $D_2 = 0.4$, $D_3 = 0.2$, then $\lambda_1 = 3.35$/year, $\lambda_2 = 2.25$/year, $\lambda_3 = 2.74/\text{year}$, and $\lambda = 8.34$/year.

B. General Reliability-Growth Models

This section provides the procedure to obtain the closed-form solution for a generic reliability-growth model. The only restriction to the growth models is with respect to their first and second derivatives. The solution procedure follows directly from the solution of the Lagrange method, except that impossible solutions must be prevented. It generalizes the procedure for the hyper-geometric model [17] to general continuous distributions.

Consider the fixed failure-rate constraint case. Let the relation between the failure rate and the testing times be functions of f_j:

$$D_j = f_j(\lambda_j). \tag{12}$$

Without loss of generality, the N components can be reordered according to the absolute values of the derivatives at the beginning of the debugging interval, at which $\lambda_j = \lambda_{j,0}$:

$$\begin{aligned} &\text{abs}\left(\frac{d}{d\lambda_j} f_j(\lambda_j)|_{\lambda_j=\lambda_{j,0}}\right) \\ &\geq \text{abs}\left(\frac{d}{d\lambda_{j+1}} f_{j+1}(\lambda_{j+1}|_{\lambda_{j+1}=\lambda_{j+1,0}}\right), \\ &\text{for } j = 1, 2, \ldots N-1. \end{aligned} \tag{13}$$

The algorithm to obtain the closed-form solution uses this ordering (13).

```
Algorithm for Closed-Form Solution
1. $K = N$;
2. For $j = 1$ to $K$
  express $\lambda_j$ as a function $g_j(\lambda_K)$ for $\lambda_j$
  such that $(d/d\lambda_j)f_j(\lambda_j) = (d/d\lambda_K)f_K(\lambda_K)$;
3. Find $\lambda_K$ from $\sum_{j=1}^{K} g_j(\lambda_K) = \delta$;
4. If $\lambda_K > \lambda_{K,0}$ Then
  $K = K - 1$, and go to step 2;
  Else
  For $j = 1$ to $K$
    Compute $\lambda_j$ from both $g_j(\lambda_K)$ in step 3
  and from the $\lambda_K$ in step 3;
    End_For
    End_If
  End_Algorithm
```

The important feature in this Algorithm is the ability to determine which component should be assigned zero testing-time if an impossible solution is obtained (an impossible solution arises if $\lambda_j > \lambda_{j,0}$ for some component). If the first derivatives $(d/d\lambda_j)f_j(\lambda_j) < 0$ for all j, and the second derivatives $(d^2/d\lambda_j^2)f_j(\lambda_j) > 0$ for all j, then the component ranked lowest according to the derivatives at time zero can be discarded. This happens in step 4 of the Algorithm; i.e., the sufficient conditions on the derivatives imply that: a) the failure rate decreases over time, and b) the rate of decrease gets smaller as time increases. If these conditions on the first and second derivatives do not hold, then specific conditions must be established to determine which components should not be assigned testing time.

The Algorithm can be similarly formulated for the fixed-testing budget problem, but is not done here.

Pareto Growth Model: As an illustration, consider the Pareto distribution for the fixed failure-rate constraint problem. The failure rate is

$$\lambda_j(t) = \epsilon_{j,0} \cdot (\epsilon_{j,1} + t)^{-\epsilon_{j,2}},$$

$\epsilon_{j,0}$, $\epsilon_{j,1}$, $\epsilon_{j,2}$ are constants.

Hence,

$$D_j = \left(\frac{\epsilon_{j,0}}{\lambda_j}\right)^{1/\epsilon_{j,2}} - \epsilon_{j,1}. \tag{14}$$

The Pareto class of failure-rate distributions is useful because it is a generalization of the exponential, Weibull, and gamma classes [15], [19]. The Crow model used in [6] is a special case of the Pareto model.

The first derivative is less than 0, and the second derivative is greater than 0, if $\epsilon_{j,0}$, $\epsilon_{j,1}$, $\epsilon_{j,2}$ are all positive. Hence, using $\partial D_j/\partial\lambda_j$, the result is, in step 2 of the Algorithm ($K = N$ in the first iteration):

$$-\frac{\epsilon_{j,0}^{1/\epsilon_{j,2}}}{\epsilon_{j,2}} \cdot \lambda_j^{-(1/\epsilon_{j,2}+1)} = -\frac{\epsilon_{K,0}^{1/\epsilon_{K,2}}}{\epsilon_{K,2}} \cdot \lambda_K^{-(1/\epsilon_{K,2}+1)}. \tag{15}$$

In step 3, one must equate the total failure rate to δ; it must be done numerically, because a closed-form expression using the Pareto distribution is too intricate. As soon as a possible solution for λ_K is obtained, compute the individual failure rate using:

$$\lambda_j = \left(\frac{\epsilon_{j,2}}{\epsilon_{K,2}} \cdot \frac{\epsilon_{K,0}^{1/\epsilon_{K,2}}}{\epsilon_{j,0}^{1/\epsilon_{j,2}}} \cdot \lambda_K^{-(1/\epsilon_{K,2}+1)} \right)^{-(1/(1/\epsilon_{j,2}+1))}. \quad (16)$$

Example 5: Assume the following parameters for the Pareto distribution. $\epsilon_{1,0} = 5$, $\epsilon_{1,1} = 1$, $\epsilon_{1,2} = 3$, $\epsilon_{2,0} = 2$, $\epsilon_{2,1} = 1$, $\epsilon_{2,2} = 6$, $\epsilon_{3,0} = 4$, $\epsilon_{3,1} = 1$, $\epsilon_{3,2} = 5$. For $\delta = 7$, (the sum of failure rates of the components) the optimal policy requires: $\lambda_1 = 3.395$, $\lambda_2 = 1.556$, $\lambda_3 = 2.047$. The total testing time is 0.3236.

IV. Solutions for Multiple Application Environment

When there are multiple applications in the system, the reliability-allocation becomes too intricate to solve explicitly. However, its solution can be obtained using nonlinear programming software such as AMPL [8].

A. An Example

An example of a 3-component, 3-application system is specified and solved.

Example 6: There are C_1, C_2, C_3 which can be used to build A_1, A_2, A_3.

- A_1 is built using C_1, C_2;
- A_2 is built using C_2, C_3;
- A_3 is built using C_1, C_2, C_3.

Thus, there are multiple applications, each with a failure-rate constraint. The fixed failure-rate constraint problem is:

Minimize:

$$D = \sum_{j=1}^{N} D_j, \quad (17)$$

under the constraints:

$$\begin{aligned} &\lambda_1 + \lambda_2 \le \delta_1 \qquad (\text{for } A_1), \\ &\lambda_2 + \lambda_3 \le \delta_2 \qquad (\text{for } A_2), \\ &\lambda_1 + \lambda_2 + \lambda_3 \le \delta_3 \qquad (\text{for } A_3). \end{aligned}$$

The fixed testing-budget problem is:

Minimize:

$$\lambda = \sum_{j=1}^{N} \lambda_j \quad (18)$$

under the constraints:

$$\begin{aligned} &D_1 + D_2 \le d_1, \quad D_2 + D_3 \le d_2, \\ &D_1 + D_2 + D_3 \le d_3. \end{aligned}$$

Use the parameters from example 3 where the initial failure rates for the 3 components are the same: $\lambda_{1,0} = \lambda_{2,0} = \lambda_{3,0} = 5$/year; and the μ values are different (e.g., $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 3$). Let the failure-rate requirements for the 3 applications be $\delta_1 = 6$, $\delta_2 = 5$, $\delta_3 = 7$. Model this as a nonlinear optimization problem with multiple constraints in AMPL, and use the MINOS solver; the result is: $\lambda_1 = 3.818$, $\lambda_2 = 1.909$, $\lambda_3 = 1.273$. The total testing-time for this failure-rate allocation is 1.207 years; (the individual testing times can be computed using the failure-rate allocation for each component). The failure-rate constraint for A_3 is strictly satisfied. For A_1 with the failure-rate requirement of 6/year, it is not strictly satisfied ($\lambda_1 + \lambda_2 = 5.73$); similarly for A_2 with the failure-rate requirement of 5/year ($\lambda_2 + \lambda_3 = 3.18$).

Consider an *average* testing policy where the constraint for A_3 is strictly satisfied without violating the other constraints: $\lambda_1 = 2.333$, $\lambda_2 = 2.333$, $\lambda_3 = 2.333$. The total testing-time based on this average testing policy is 1.39 years, much larger than that obtained with the optimal testing policy.

B. Software-Failure Dependencies and Fault-Tolerant Systems

The basic reliability-allocation problem formulation can be extended in various ways. Two extensions are discussed: software-failure dependencies and fault-tolerance aspects.

Section IV-A assumes that software components fail s-independently. In reality, this might well not be the case. For example, the feature interaction problem [12] describes many incidents where independently developed software components interact with each other unexpectedly, thus causing unanticipated failures. This extra failure incidence is incorporated by introducing *pair-wise* failure rates: $\lambda_{i,j}$ represents the failure rate due to the interaction of C_i and C_j, $i \le j$. These failures are caused by interactions of software components. Therefore, they are not detected in individual component testing, but by integration testing of pairwise components. These failure rates are computed by counting the numbers of failures involving pairwise components during the integration testing, and divide them by the pairwise components execution times spent in the integration testing. While failures involving 2 components might not be neglectable, failures involving at least 3 components are usually rare [13].

The constraints of the original problem are then modified as:

$$\sum_{j=1}^{N} \sigma_{i,j} \cdot \lambda_j + \sum_{\Omega(i,j)} \lambda_{i,j} \le \delta_i \qquad (\text{for } A_i),\ i = 1, \ldots M$$
$$\Omega(i,j) \equiv \{\forall (i,j) | \sigma_{1,i} = 1, \sigma_{1,j} = 1\}. \quad (19)$$

Subject to the constraint (19), minimize:

$$D = \sum_{j=1}^{N} D_j$$

Thus the fixed testing-time problem can be obtained by adding the pair-wise failure rates in the failure-rate constraints and including all individual and pairwise component testing times in the objective function.

When the system possesses fault-tolerant attributes, introduce *coverage factors* [23] into the original problem. Coverage is defined as the conditional probability that when a fault is activated in a component, it is detected and recovered without

TABLE I
SYSTEM COMPONENTS WITH CORRESPONDING PARAMETERS GROWTH-CURVE, AND APPLICATIONS THAT USE THE COMPONENT

Component	$\lambda_{j,0}$	μ_j	std. I	std. II	1-800 I	1-800 II
basic	10	1.0	in	in	in	in
scheduling	20	1.0	in	in	in	in
call proc.	200	0.2	not in	in	in	in
signal I	200	0.5	in	in	in	not in
signal II	20	1.0	in	in	not in	in
frequency			0.5	0.3	0.1	0.1

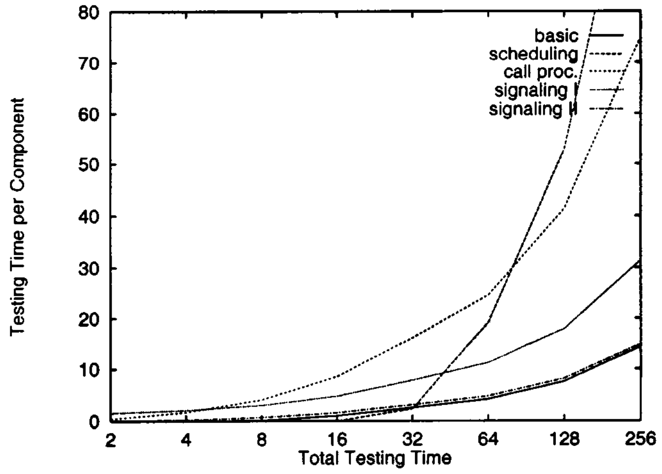


Fig. 2. Optimal allocation for components vs. total available testing-time.

causing system failure. Reformulate the fixed failure-rate constraint case, using $\rho_j \equiv 1 - c_j$, as:

Minimize

$$D = \sum_{j=1}^{N} D_j \tag{20}$$

subject to:

$$\sum_{j=1}^{N} \sigma_{i,j} \cdot \rho_j \cdot \lambda_j \le \delta_1 \qquad (\text{for } A_i),\ i = 1, \ldots, M.$$

Fault-tolerant attributes are usually provided by external system-components. The coverage factors are determined by the design features of these components, which is independent of how well the target components C_j are tested.

V. RELIABILITY-ALLOCATION SOLUTION FRAMEWORK

Reliability allocation has been discussed in terms of two constraints: fixed failure rates or fixed testing budgets. The problem of accounting for component interactions also has been discussed. The fault-tolerant attributes in the system to tolerate component failures are also incorporated. This section formulates a framework for specifying and solving a general reliability-allocation problem, and applies this framework to a specific application. A tool to automate the procedure is described.

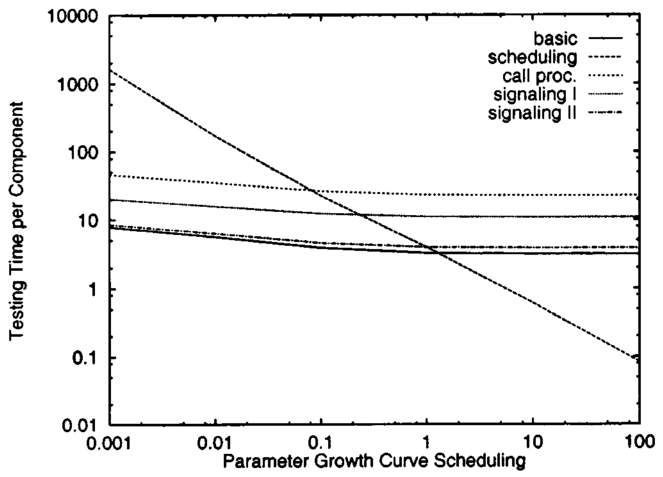


Fig. 3. Optimal testing-time allocation for all components vs. μ in growth curve of the scheduling software.

A. Problem Specification and Solution Procedure

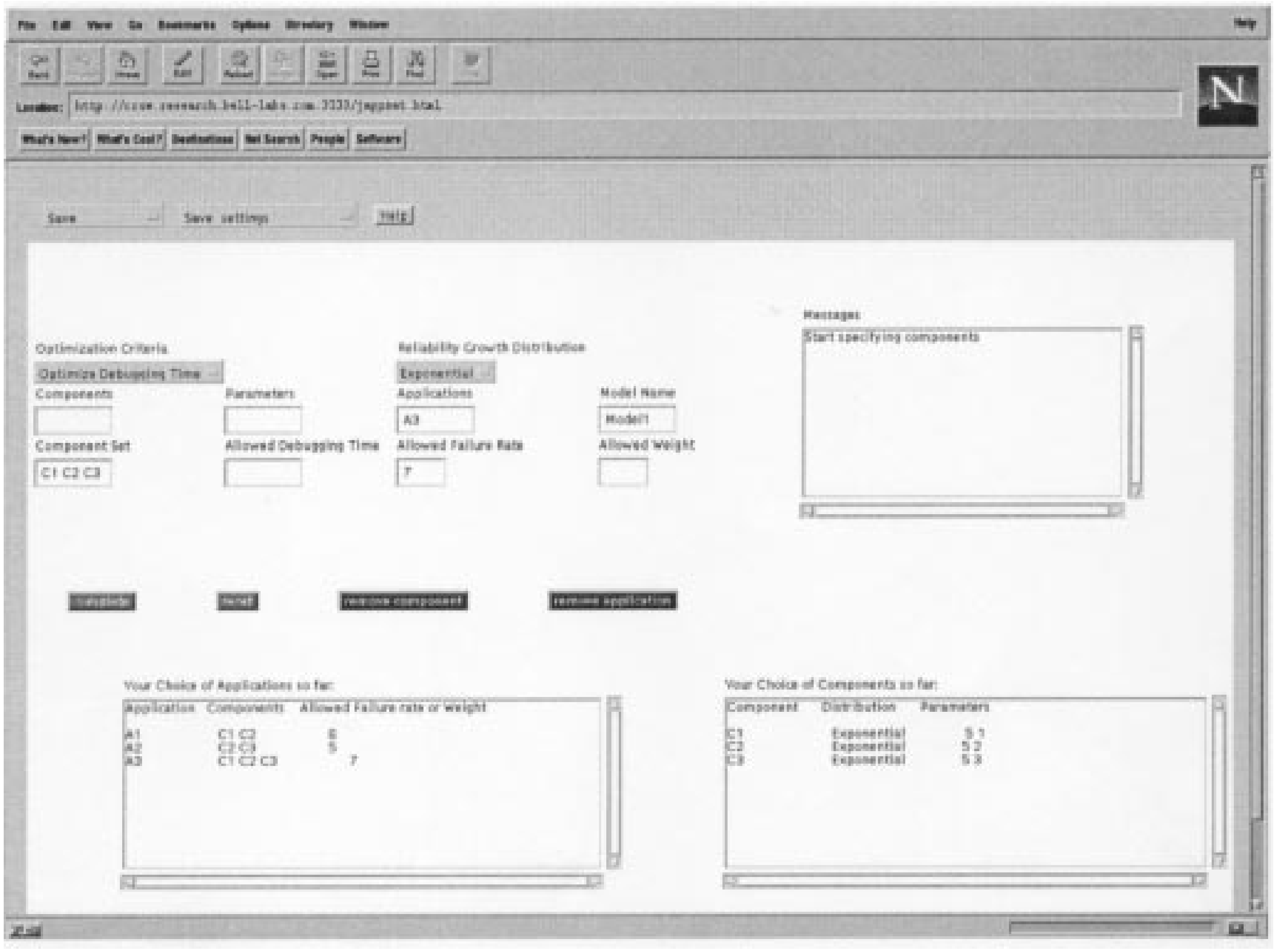
The following procedure specifies the reliability allocation problem, and obtains solutions either analytically or using numerical methods.

```
Procedure
  1) Determine if the problem is a fixed failure rate constraint or a fixed testing budget.
  2) Determine if there is single application or multiple applications in the system.
  3) Set the constraints on the failure rates or testing budgets.
  4) Obtain parameters of the reliability-growth curves of the components.
  5) Determine if the components interact. If so, obtain pair-wise failure rates.
  6) Determine if there are fault-tolerance features in the system. If so, obtain coverage measures for each component.
  7) Format the problem as a nonlinear programming problem with appropriate parameters.
  8) If the solution is analytically available, obtain it. Otherwise, use mathematical programming tools and solvers to obtain the results.
End_Procedure
```

Section V-B examines a case study where a required reliability-allocation problem is specified. It applies the procedure in Section V-A to the project to obtain numerical solutions for various scenarios.

B. Hypothetical Example

Consider a distributed software architecture used for switching telephone calls. Different call-types exercise different software-modules; the system is split into components such that reliability-growth models are available for all components. A prerequisite to this analysis is the availability of reliability-growth models, and this example clearly shows that it is beneficial to make decisions based on such models.



Messages: Start specifying components

Optimization Criteria: Optimize Debugging Time

Reliability Growth Distribution: Exponential

Components:

Parameters:

Applications: A3

Model Name: Model1

Component Set: C1 C2 C3

Allowed Debugging Time:

Allowed Failure Rate: 7

Allowed Weight:

Your Choice of Applications so far:

```
Application   Components   Allowed Failure rate or weight

A1            C1 C2         6
A2            C2 C3         5
A3            C1 C2 C3        7
```

Your Choice of Components so far:

```
Component   Distribution   Parameters

C1          Exponential      5 1
C2          Exponential     5 2
C3          Exponential     5 3
```

Fig. 4. Reliability allocation tool.

Table I shows the data input in this example. Neither the example nor the data correspond to existing systems or numbers. Consider 4 types of calls (2 types of standard calls, and 2 types of 1-800 calls), and 5 components (basic processing, scheduling, call processing, and 2 signal-processing modules). The terms "in" and "not in" in Table I denote which components are used by the applications; e.g., the standard calls of type 1 use all software modules except the call-processing module. The reliability-growth curves for the components are exponential, and have parameters μ and λ, as specified in Table I.

The results of this example show two things:

- the necessity to use mathematical optimization techniques to establish an optimal allocation scheme,
- the importance of selecting and parameterizing adequate reliability-growth models.

Fig. 2 depicts, with a given total amount of testing-time available (x-axis), the time allocated to test individual components (y-axis). Following the framework in Section V-A it was solved as a fixed testing budget problem with multiple applications. Assume, however, that the testing time is shared by all applications: consider the special case in Section II-B where the constraints map to a single constraint. Applications are weighted, based on their relative frequency of occurrence (in Table I), which can be automatically converted to weights on the component failure rates in the objective function. Using relative frequencies for various call applications parts of the operational profile are included (see, e.g., [15, chapter 5]) in the model. Solutions were obtained for the testing-time ranging from 2 to 256, assuming no failure dependency or explicit fault-tolerance mechanisms.

Fig. 2 shows very clearly the dependence of the optimal schedule on the total testing-time. For example, while the scheduling component should not be assigned debugging time if the available budget is small, it takes the largest chunk if the testing budget is large. The irregular assignment of testing-time to individual components in Fig. 2 cannot be obtained easily by means other than mathematical modeling. Without a systematic approach such as in this paper, one could not expect to get such precise results, and would be forced to make inefficient decisions.

Fig. 3 plots changes of the allocation of testing-time to the components, while "μ of the reliability-growth curve for the scheduling software" varies from 0.001 to 100. In this case, the allowed failure rates per application were taken to be 4, and the fixed failure-rate constraint problem was solved.

The parameter value greatly influences the optimal solution. If the decay parameter of the reliability-growth model of the scheduling component is small, then it takes enormous investments in debugging time to reach the desired failure rates. If the decay parameter is relatively large, then it takes minor effort for the scheduling component to obey the failure rate restrictions.

The s-correlation between the optimal testing-time and the parameters of the reliability-growth curve shows the importance of data collection to establish trustworthy growth models. Without such models, decisions about reliability allocation will be suboptimal.

C. RAT: The Reliability Allocation Tool

A reliability allocation tool (RAT) was designed and built with a GUI (graphical user interface) based on a Java Applet. The tool allows: a) multiple applications to be specified, and b) optimizations to be performed both under the fixed failure-rate and fixed testing-budget constraints. The user inputs the model using the GUI. and the input is converted into AMPL files and is solved using the MINOS solver, called by AMPL. Fig. 4 shows the GUI. The tool chooses the optimization criteria, where optimizing the failure-rate implies that the constraint is the fixed testing budget, and "optimizing testing time" implies that the constraint is a fixed failure rate. Components can be specified in the "Components" field, and applications can be specified in the "Applications" field. The reliability-growth distribution can be chosen for each component independently; parameters for these distributions can be specified in the "Parameters" box. At present, exponential and Pareto distributions are allowed, but we plan to extend the options to specifying other distributions. For a fixed failure-rate constraint, the allowed failure-rate for the applications can be specified in the "Allowed Failure Rate" field; similarly, if testing time is fixed, then it can be specified in the "Allowed Debugging Time" field. Information about the components that have been specified and the applications that have been input are shown in two separate areas. When the model is solved, it produces results in the "Message" field.

The major computational effort required for the RAT tool is on the MINOS solver. For the 5-component system in Section V-B, it takes about a few milliseconds to obtain the result in a Sun Untra workstation. As the time requirement increases only linearly with the number of components in the system (not including pair-wise failure rates and fault-tolerant attributes), the performance of the RAT tool is quite acceptable.

ACKNOWLEDGMENT

The authors are pleased to thank C. M. Kintala of Avaya Communications for many of his valuable suggestions and comments for this work.

REFERENCES

[1] M. Avriel, "Nonlinear programming," in *Mathematical Programming for Operations Researchers and Computer Scientist*, A. G. Holzman, Ed: Marcel Dekker, 1981, ch. 11.

[2] O. Berman and N. Ashrafi, "Optimization models for reliability of modular software systems," *IEEE Trans. Software Reliability*, vol. 19, pp. 1119–1123, Nov. 1993.

[3] 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.

[4] X. Cai, M. R. Lyu, K. F. Wong, and R. Ko, "Component-based software engineering: Technologies, development frameworks, and quality assurance schemes," in *Proc. Asia-Pacific Software Engineering Conf.*, Dec. 2000, pp. 372–379.

[5] D. W. Coit and A. E. Smith, "Reliability optimization of series-parallel systems using a genetic algorithm," *IEEE Trans. Reliability*, vol. 45, 1996.

[6] D. W. Coit, "Economic allocation of test times for subsystem-level reliability growth testing," *IIE Trans.*, vol. 30, no. 12, pp. 1143–1151, Dec. 1998.

[7] A. K. Dhingra, "Optimal apportionment of reliability and redundancy in series systems under multiple objectives," *IEEE Trans. Reliability*, vol. 41, pp. 576–582, Dec. 1992.

[8] R. Fourer *et al.*, *AMPL: A Modeling Language for Mathematical Programming*: The Scientific , 1993.

[9] D. E. Fyffe, W. W. Hines, and N. K. Lee, "System reliability allocation and a computational algorithm," *IEEE Trans. Reliability*, vol. R-17, pp. 64–69, 1968.

[10] P. M. Ghare and R. E. Taylor, "Optimal redundancy for reliability in series system," *Operational Res.*, vol. 17, pp. 838–847, 1969.

[11] A. L. Goel and K. Okumoto, "Time-dependent error detection rate model for software and other performance measures," *IEEE Trans. Reliability*, vol. R-28, pp. 206–211, Aug. 1979.

[12] N. Griffeth and Y.-J. Lin, Eds., *IEEE Communications Mag., Special Issue on Feature Interactions in Telecommunications Systems*, Aug. 1993.

[13] M. Kaâniche, K. Kanoun, M. Cukier, and M. Bastos Martini, "Software reliability analysis of three successive generations of a switching system," in *Proc. First European Dependable Computing Conf.*, Oct. 1994, pp. 473–490.

[14] W. Kozaczynski and G. Booch, "Component-based software engineering," *IEEE Software*, vol. 155, pp. 34–36, Sep./Oct. 1998.

[15] M. R. Lyu, Ed., *Handbook of Software Reliability Engineering*: McGraw-Hill and IEEE Computer Society Press, 1996.

[16] M. R. Lyu, S. Rangarajan, and A. P. A. van Moorsel, "Optimization of reliability allocation and testing schedule for software systems," in *Proc. 1997 Int. Symp. Software Reliability Eng.*, Nov. 1997, pp. 336–346.

[17] R.-H. Hou, 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," in *Proc. 7th Int. Symp. Software Reliability Eng.*, Oct./Nov. 1996, pp. 289–298.

[18] K. B. Misra and U. Sharma, "An efficient algorithm to solve integer programming problems arising in system reliability design," *IEEE Trans. Reliability*, vol. R-40, pp. 81–91, 1991.

[19] J. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*: McGraw-Hill, 1987.

[20] J. Musa, "Validity of execution-time theory of software reliability," *IEEE Trans. Reliability*, vol. R-28, pp. 181–191, Aug. 1979.

[21] Y. Nakagawa and S. Miyazaki, "Surrogate constraints algorithm for reliability optimization problems with two constraints," *IEEE Trans. Reliability*, vol. R-30, pp. 175–181, 1981.

[22] L. Painton and J. Campbell, "Genetic algorithms in optimization of system reliability," *IEEE Trans. Reliability*, vol. 44, pp. 172–178, 1995.

[23] D. P. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*, 2nd ed: Digital, 1992.

[24] F. A. Tillman, C. L. Hwang, and W. Kuo, "Determining component reliability and redundancy for optimum system reliability," *IEEE Trans. Reliability*, vol. R-26, pp. 162–165, 1977.

[25] Y. Tohma, K. Tokunaga, S. Nagase, and Y. Murata, "Structural approach to the estimation of the number of residual software faults based on the hyper-geometric distribution," *IEEE Trans. Software Eng.*, vol. 15, pp. 345–355, Mar. 1989.

[26] F. Zahedi and N. Ashrafi, "Software reliability allocation based on structure, utility, price, and cost," *IEEE Trans. Software Eng.*, vol. 17, pp. 345–355, Apr. 1991.

[27] S. M. Yacoub, B. Cukic, and H. H. Ammar, "A component-based approach to reliability analysis of distributed systems," in *Proc. 18th IEEE Symp. Reliable Distributed Syst.*, 1999, pp. 158–167.

[28] ——, "A scenario-based reliability analysis of component-based software," in *Proc. 10th Int. Symp. Software Reliability Eng.*, 1999, pp. 22–31.

Michael R. Lyu (SM'97) received the B.S. degree (1981) in electrical engineering from National Taiwan University, the M.S. degree (1985) in computer engineering from the University of California, Santa Barbara, and the Ph.D. degree (1988) in computer science from the University of California, Los Angeles.

He is a Professor with the Computer Science and Engineering Department of the Chinese University of Hong Kong. He worked at the Jet Propulsion Laboratory, Bellcore (now Telcordia), and Bell Labs, and taught at the University of Iowa. His current research interests include software reliability engineering, distributed systems, fault-tolerant computing, web technologies, web-based multimedia systems, and wireless communications. He has published over 100 refereed journal and conference papers in these areas. He is the editor of two book volumes: *Software Fault Tolerance* (New York: Wiley, 1995) and the *Handbook of Software Reliability Engineering* (New York: IEEE and McGraw-Hill., 1996).

Dr. Lyu initiated the first International Symposium on Software Reliability Engineering (ISSRE) in 1990. He is the Program Chairman for ISSRE'96, PRDC'99, and WWW10, and has served in program committees for numerous international conferences. He serves as General Chairman for ISSRE'2001. He is on the editorial board for the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON RELIABILITY, and the *Journal Information Science and Engineering*.

Sampath Rangarajan (M'91) received the Ph.D. degree (1990) in computer sciences, and the M.S. degree (1987) in electrical and computer engineering, both from the University of Texas at Austin.

He is Chief Architect of Technology at Ranch Networks, a start-up company that builds internet infrastructure products. Before joining Ranch Networks, he was a Member of Technical Staff in the Distributed Software Research Department, which is part of the Systems and Software Research Center at Bell Laboratories, Murray Hill, NJ. Previously, he was an Assistant Professor with the Department of Electrical and Computer Engineering at Northeastern University, Boston, MA (1992–1996) and a Research Associate with the University of Maryland Institute for Advanced Computer Studies (UMIACS) (1990–1992). His research interests are in distributed computing and networking.

Aad P. A. van Moorsel received the M.S. degree (1989) in mathematics from the University of Leiden, The Netherlands, and the Ph.D. degree (1993) in computer science from the University of Twente, The Netherlands. He developed performance, reliability, and performability evaluation techniques for telecommunication networks and distributed systems.

From 1994 to 1995 he was a research assistant with the University of Illinois at Urbana-Champaign, where he worked on software-tool support for performability evaluation. In 1996, he joined Lucent Technologies, where he was a Member of the Distributed Software Research Department at Bell Laboratories, Murray Hill, NJ. In this group he worked in performability engineering, software fault-tolerance, and system management. Since November 1999, he has been working with HP Laboratories, where he heads the E-Services Software Research Department. This department specializes in ground-breaking research in manageability and reliability of middleware and Internet applications.