Optimization of Reliability Allocation and Testing Schedule for Software Systems

Michael R. Lyu
Sampath Rangarajan
Aad P. A. van Moorsel
Bell Laboratories, Lucent Technologies
600 Mountain Avenue, Murray Hill, NJ 07974
{lyu,sampath,aad}@research.bell-labs.com

Abstract

To ensure an overall reliability of an integrated software system, software components of the system have to meet certain reliability requirements, subject to some testing schedule and resource constraints. The system testing activity can be formulated as a combinatorial optimization problem with known cost, reliability, effort, and other attributes of the system components. In this paper we consider the software component reliability allocation problem for a system with multiple applications. The failure rate of components used to build the applications are related to the testing cost through various types of reliability growth curves. We achieve closed-form solutions to problems where there is one single application in the system. Analytical solutions are not readily available when there are multiple applications; however, numerical solutions can be obtained using a non-linear programming tool. To ease the specification of the optimization problem, we develope a GUI front-end to existing mathematical software. We present a systematic outline of the problem formulation and solution, and apply this to an example of a telecommunication software system.

1. Introduction

Modern complex software systems are often developed with components supplied by contractors or independent teams under different environments. 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 [7, 16], integer programming [8, 3, 13], non-linear optimization [19], and heuristic techniques [17, 4]. In [5] the optimal apportionment of reliability and redundancy is considered for multiple objectives using fuzzy optimization techniques.

Some researchers also address the reliability allocation problem for software components. The software reliability allocation problem is addressed in [21] 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 take testing time of software components and the growth of their reliability into consideration. Optimal allocation of component testing times in a software system based on a particular software reliability model is addressed in [12], but it assumes a single application in the system, and the reliability growth model is limited to the Hyper-Geometric Distribution ("S-shaped") Model [20].

In this paper we discuss 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. We will also give the solution procedure for the single application environment, for general continuous distributions, thus generalizing [12]. We examine the situation where software components may interact with each other, a condition not considered by other studies. We also include scenarios for fault-tolerant attributes of a system where some component failures can be tolerated. The problem specification and solution seeking procedure, as well as a software tool for the automatic application of this procedure, are presented in this paper as an innovative mechanism to handle

the difficult yet important reliability allocation problem.

Several real projects motivate us for this investigation. These projects are described in the following case studies.

Distributed Software Systems. Distributed telecommunication systems often serve multiple application types, which execute different software modules, and have different reliability requirements. For instance, in telephone switches, 1-800 calls require processing different from standard calls, and similar examples exists 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. In the example we discuss in Section 6.2 it will become clear that trustworthy reliability growth curves can help considerably in efficient testing and debugging planning of such systems.

Fault-Tolerant Systems. Figure 1 shows a layered software architecture model that can be applied to many systems. Each layer can include several software components. Not all systems will necessarily include all the layers.

We conjecture that error propagation between layers only occurs in one direction, namely upwards. Thus, faults in the hardware that are not contained can propagate up to the operating system or to the application software; however, faults not contained in the middle-ware layer will 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, i.e., the layer does not exhibit fail-silent behavior. From a modeling perspective, this layer would contribute higher failure rate to the overall system than a layer with error detection and recovery mechanisms. Note that error detection and recovery software can reside in some or all of the layers.

In Section 5.2 we will see how fault-tolerant mechanisms can be included in the reliability allocation problem formulation, provided that coverage factors are available.

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 presented in, e.g., [21, 2]. Our intent in this paper is

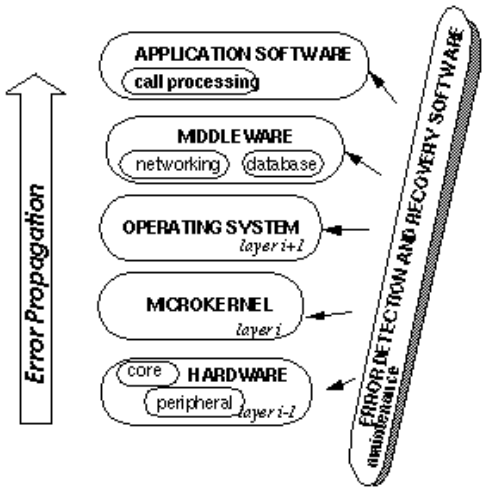

Figure 1. Layered software architecture model.

to optimize with respect to software development and testing, not with respect to software structure. The combination of structure-oriented optimization methods in [21, 2] and the development-oriented methods in this paper can provide powerful tools in software system design.

The remaining sections of this paper are organized as follows: Section 2 specifies the optimal reliability allocation as two related problems: a problem with fixed target failure rate, and a problem with fixed debugging time. The analytical solutions to these two problems are presented for the single application environment in Section 3: Section 3.1 for the exponential distribution, and Section 3.2 for general distributions. Section 4 discusses how the solutions for the problems in multiple application environment can be obtained. Our results are extended in Section 5.1 to consider software failure dependencies, and in Section 5.2 to incorporate fault-tolerant systems. Section 6 proposes the reliability allocation problem specification and solution procedure into a step-by-step framework, and applies it to a case study. Section 7 describes the design and implementation of a software tool for systematic application of the reliability allocation framework. Conclusions are presented in Section 8.

2. Problem Specification

The above case studies can be described by 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. Consider the situation where a set of N software components $C_1, \cdots, C_N$,

can be used in various combinations for different applications. Let there be M such applications $A_1, \cdots, A_M$, and let each application have a pre-specified reliability requirement $R_1(t), \cdots, R_M(t)$. By investing development/testing/debugging time in components, 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 the minimal cost. In what follows, we will characterize the cost in terms of the component testing (including debugging) time. The optimization criterion thus is the minimization of this testing time. Furthermore, to relate the failure rate of components with the amount of testing time, we use reliability growth models.

A variation to the above problem formulation arises if a fixed amount of testing time is available for each application. This requirement may occur because of the constraint on the cost incurred by the component developer and tester. In that case we take as objective function minimization of the failure rate of all the components.

In what follows we will continuously discuss these two variations of the optimization problem, and they will be referred to as the 'fixed failure rate constraint' problem and 'fixed testing budget' problem, respectively.

2.1. Fixed Failure Rate Constraint

In the fixed failure rate case, we assign testing time to components such that the applications meet their reliability requirements, and the testing time is minimized. Throughout this paper, we assume that the failure rates of components relate to the reliability of applications through the exponential relation $R_i(t) = e^{-\delta_i t}$, where δ_i is the sum of failure rates of the components in application $A_i, i = 1, \ldots, M$. Furthermore, we assume that the testing time D_i invested in component i decreases the failure rate λ_i according to some reliability growth model. Note that we assume that 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. In this context, we can formulate the allocation problem as follows.
The objective function is:

$$Minimize \qquad D = D_1 + D_2 + \cdots + D_N,$$

subject to the constraints:

$\sigma_{11}\lambda_1 + \sigma_{12}\lambda_2 + \cdots + \sigma_{1N}\lambda_N \leq \delta_1$ (for application A_1),
$\sigma_{21}\lambda_1 + \sigma_{22}\lambda_2 + \cdots + \sigma_{2N}\lambda_N \leq \delta_2$ (for application A_2),
$\cdots$
$\sigma_{M1}\lambda_1 + \sigma_{M2}\lambda_2 + \cdots + \sigma_{MN}\lambda_N \leq \delta_M$ (for application A_M),
where $\sigma_{ij} = 1$ if A_i uses component C_j, and $\sigma_{ij} = 0$ otherwise. Note $\lambda_1, \ldots, \lambda_N \geq 0$; $D_1, \ldots, D_N \geq 0$; and $\delta_1, \ldots, \delta_M \geq 0$. For the sake of notational simplicity, we assume that the testing times D_i are equally 'important' (costly) among the N components. If this is not the case one can apply weight ω_i to each component testing time D_i in the objective function.

Since a reliability growth curve can be very complex, the objective function is non-linear, and hence this is a general non-linear programming problem. In Section 3.1.1 we consider the closed-form solution for the problem with a single application, and in Section 4 we discuss the numerical solution of the general case. Note also that we assume independence of components with respect to their failure behavior. This assumption may not be appropriate when software components may interact with each other, potentially causing additional failures.

2.2. Fixed Testing Budget

In the fixed testing budget case, we distribute per application a specified amount of testing time over its components, such that 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:

$$Minimize \qquad \lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N,$$

subject to the constraints:

$\sigma_{11}D_1 + \sigma_{12}D_2 + \cdots + \sigma_{1N}D_N \leq d_1$ (for application A_1),
$\sigma_{21}D_1 + \sigma_{22}D_2 + \cdots + \sigma_{2N}D_N \leq d_2$ (for application A_2),
$\cdots$
$\sigma_{M1}D_1 + \sigma_{M2}D_2 + \cdots + \sigma_{MN}D_N \leq d_M$ (for application A_M),
where $d_1, d_2, \cdots, d_M$ are the fixed amount of testing times that are available for components used by applications $A_1, A_2, \cdots, A_M$, respectively. As in the previous problem, all variables are positive. Weight functions for the failure rates λ_i can be introduced in the objective function to reflect their impact.

Of special interest is the single application case. This case 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. In Section 3.1.2 and Section 4, we discuss the case of single and multiple applications, respectively.

3. Solutions for Single Application Environment

When there is only a single application in the system, an explicit solution of the reliability allocation problem can usually be found. In this section we give the general solution for a large class of reliability growth models (see Section 3.2). To explain the solution procedure, we first provide in Section 3.1 the solution assuming an exponential reliability growth model.

3.1. Exponential Reliability Growth Model

The exponential reliability growth model [9, 15] relates the failure rate λ_i (for component i) with the invested testing time D_i through:

$$\lambda_i = \lambda_{i0} e^{-\mu_i D_i}.$$

λ_{i0} is the initial failure rate at time 0, and μ_i is the decay parameter. Over an infinite time interval $\frac{\lambda_{i0}}{\mu_i}$ faults will be found. Note that λ_i is a function of time, although we do not explicitly express it in the notation. This reliability growth model is in common use, and we will now determine the solution for the allocation problem in the single application environment.

3.1.1 Fixed Failure Rate Constraint

The fixed failure rate constraint problem can be formulated in the single application environment, assuming exponential reliability growth curves, as:

$$Minimize \qquad D = \sum_{i=1}^{N} \frac{1}{\mu_i} ln\left(\frac{\lambda_{i0}}{\lambda_i}\right),$$

subject to:

$$\lambda_1 + \lambda_2 + \cdots + \lambda_N \leq \delta.$$

To solve this, one can use the Lagrange method as follows [1]. The optimization problem is equivalent to finding the minimum of:

$$F(\lambda_1, \cdots, \lambda_N) = D + \theta((\lambda_1 + \cdots + \lambda_N) - \delta),$$

where θ is a Lagrange multiplier. The necessary conditions [1] for the minimum to exist are:

1. $\Delta(F(\lambda_1, \cdots, \lambda_N)) = 0$ where Δ is the partial derivative operator,
2. $\theta > 0$,
3. $\lambda_1 + \lambda_2 + \cdots + \lambda_N = \delta$.

If we expand the equation from the first condition and take the partial derivatives w.r.t $\lambda_1, \lambda_2, \cdots, \lambda_N$, we can get the following equations, respectively.

$$\frac{-1}{\mu_1 \lambda_1} + \theta = 0,$$
$$\frac{-1}{\mu_2 \lambda_2} + \theta = 0,$$
$$\cdots$$
$$\frac{-1}{\mu_N \lambda_N} + \theta = 0.$$

Given that the μ and the λ values are all positive, it is clear that $\theta > 0$, satisfying the second condition. From the above equations we also note that $\mu_1\lambda_1 = \mu_2\lambda_2 = \cdots = \mu_N\lambda_N$. Together with this observation and the third condition, we get the following solution for the obtained failure rates.

$$\lambda_1 = \frac{\delta}{1 + \frac{\mu_1}{\mu_2} + \cdots + \frac{\mu_1}{\mu_N}},$$
$$\lambda_2 = \frac{\mu_1}{\mu_2}\lambda_1,$$
$$\cdots$$
$$\lambda_N = \frac{\mu_1}{\mu_N}\lambda_1.$$

The testing times that should be allotted to the software components now follows from substituting the above values into the equations for $D_1, D_2, \cdots, D_N$. For example, D_1 is equal to

$$D_1 = \frac{1}{\mu_1} \ln\left[\frac{\lambda_{10}}{\frac{\delta}{1+\frac{\mu_1}{\mu_2}+\cdots+\frac{\mu_1}{\mu_N}}}\right].$$

Note that D_1 is positive if $\lambda_1 \geq \lambda_{10}$. To assure that no impossible solutions arise, we present in Section 3.2 a procedure that checks for validity conditions and guarantees that the optimal solution follows a valid strategy.

Example 1: Consider an example where a system has three components C_1, C_2, and C_3 and one application A which uses all the components. All the three components have an initial failure rate of 5 failures/year ($\lambda_{10} = \lambda_{20} = \lambda_{30} = 5/yr$). Assume that the application requirement states that the failure rate of the application (δ) cannot be more than $6/yr$. Also assume that all the μ values are the same and equals 1. In this case, we note that $\lambda_1 = \lambda_2 = \lambda_3 = 2/yr$ and $D_1 = D_2 = D_3 = \ln(2.5)$. That is, 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 components' failure rates are brought down to the same value, provides a solution that meets the application requirement with minimum testing time spent; the testing time spent on each component is the same.

Example 2: Now consider an example where the three components have initial failure rates which are different (say, $\lambda_{10} = 5/yr$, $\lambda_{20} = 6/yr$, $\lambda_{30} = 7/yr$). Assume that $\delta = 6/yr$. Also assume that the μ values are all identical and equal to 1. In this case, again we find that the required λ values are the same and equal 2. Here again, an *average testing policy* provides a solution that meets the application requirement with minimum testing time spent. But now, the testing time spent on each component is different because the initial failure rates are different. In this case, $D_1 = \ln(\frac{5}{2})$, $D_2 = \ln(\frac{6}{2})$ and $D_3 = \ln(\frac{7}{2})$. The testing time spent on each component is now proportional to the logarithm of the initial failure rate.

Example 3: Let us now assume that the three components have initial failure rates which are the same ($\lambda_{10} = \lambda_{20} = \lambda_{30} = 5/yr$), and that the μ values are different (say, $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 3$). Again, assume that $\delta = 6$. Now, computation tells us that to minimize the testing time, we require $\lambda_1 = \frac{6}{1.834} = 3.273$, $\lambda_2 = \frac{3}{1.834} = 1.636$ and $\lambda_3 = \frac{2}{1.834} = 1.091$. The optimal testing policy that leads to the above failure rates will require a total testing time of $\ln(\frac{5\times 1.834}{6}) + \frac{1}{2}\ln(\frac{5\times 1.834}{3}) + \frac{1}{3}\ln(\frac{5\times 1.834}{2}) = 1.49$. An *average testing policy* which assigns $\lambda_1 = \lambda_2 = \lambda_3 = 2$ leads to a total testing time of $\ln(\frac{5}{2}) \times 1.834 = 1.68$ which is more than that of the optimal testing policy.

3.1.2 Fixed Testing Budget

The fixed testing budget problem can be formulated in the single application environment, assuming exponential reliability growth curves, as:

$$Minimize \qquad \lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N,$$

subject to the constraint

$$D_1 + D_2 + \cdots + D_N \leq D.$$

Again, this can be solved using the Lagrange method, which uses that the optimization problem is equivalent to finding the minimum of

$$F(D_1, \cdots, D_N) = \lambda + \theta((D_1 + D_2 + \cdots + D_N) - D),$$

where θ is the Lagrange multiplier. Again, the necessary conditions for a minimum to exist are:

1. $\Delta(F(D_1, D_2, \cdots, D_N)) = 0$ where Δ is the partial derivative operator,
2. $\theta > 0$,
3. $D_1 + D_2 + \cdots + D_N = D$.

If we expand the equation from the first condition and take partial derivatives with respect to D_i, we get:

$$\lambda_{i0}(-\mu_i e^{-\mu_i D_i}) + \theta = 0.$$

Given that μ_i is positive, we note that θ is positive and hence the second condition is satisfied. Thus, from the above equation, we get

$$\lambda_{10}(-\mu_1 e^{-\mu_1 D_1}) = \lambda_{20}(-\mu_2 e^{-\mu_2 D_2})$$

$$= \cdots = \lambda_{N0}(-\mu_N e^{-\mu_N D_N}).$$

Using the above equation together with the constraint that $D_1 + D_2 + \cdots + D_N = D$, we get

$$D_1 = \frac{D - [\frac{1}{\mu_2}\ln(\frac{\lambda_{20}\mu_2}{\lambda_{10}\mu_1}) + \frac{1}{\mu_3}\ln(\frac{\lambda_{30}\mu_3}{\lambda_{10}\mu_1}) + \cdots + \frac{1}{\mu_N}\ln(\frac{\lambda_{N0}\mu_N}{\lambda_{10}\mu_1})]}{1 + \frac{\mu_1}{\mu_2} + \cdots + \frac{\mu_1}{\mu_N}},$$

$$D_2 = \frac{1}{\mu_2}\ln\left(\frac{\lambda_{20}\mu_2}{\lambda_{10}\mu_1}\right) + \frac{\mu_1}{\mu_2}D_1,$$

$$\cdots$$

$$D_N = \frac{1}{\mu_N}\ln\left(\frac{\lambda_{N0}\mu_N}{\lambda_{10}\mu_1}\right) + \frac{\mu_1}{\mu_N}D_1.$$

The above equations determine how the testing times should be allocated to the different components. The minimum λ value that is obtained follows directly from the values for the testing times.

It is interesting to note that only if the initial failure rates λ_{i0}'s and the μ_i values are the same, an *average testing policy* where the available testing time is equally divided among the components will provide an optimal solution. If either the λ_{i0}'s or the μ_i values are different, then we need to compute the above expressions 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 ($\lambda_{10} = \lambda_{20} = \lambda_{30} = 5/yr$), and the μ values are different (say, $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 3$). Assume that the available testing time is 1yr. Then, D_1 computes to 0.1567, D_2 computes to 0.4249 and D_3 computes to 0.4184. The optimized failure rate evaluates to $\lambda_1 = 4.27/yr$, $\lambda_2 = 2.14/yr$ and $\lambda_3 = 1.43/yr$ for a total failure rate of $7.84/yr$. If an average allocation policy is used, then $D_1 = D_2 = D_3 = 0.333$ and the failure rates evaluate to $\lambda_1 = 3.58/yr$, $\lambda_2 = 2.57/yr$ and $\lambda_3 = 1.84/yr$ for a total failure rate of $7.99/yr$ which is worse than the optimal allocation. Let us try another allocation. Assume an allocation where $D_1 = 0.4$, $D_2 = 0.4$ and $D_3 = 0.2$. In this case, we note that $\lambda_1 = 3.35/yr$, $\lambda_2 = 2.25/yr$ and $\lambda_3 = 2.74/yr$ for a total failure rate of $8.34/yr$.

3.2. General Reliability Growth Models

In this section we provide the procedure to obtain the closed-form solution for generic reliability growth model. The only restriction to the growth models is with respect to the 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 given in [12] to general continuous distributions.

We consider the fixed failure rate constraint case. Let the relation between the failure rate and the testing time be given by some function f, that is:

$$D_i = f_i(\lambda_i).$$

Now, 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 $\lambda_i = \lambda_{i0}$. That is:

$$abs(\frac{d}{d\lambda_i} f_i(\lambda_i)|_{\lambda_i=\lambda_{i0}}) > abs(\frac{d}{d\lambda_{i+1}} f_{i+1}(\lambda_{i+1})|_{\lambda_{i+1}=\lambda_{(i+1)0}}),$$

for $i = 1, 2, \ldots N-1$. Using this ordering, the procedure to obtain the closed-form solution is as follows:

1. $K = N$;
2. For $i = 1$ to K
   express λ_i as a function $g_i(\lambda_K)$ of λ_K, by equating $\frac{d}{d\lambda_i} f_i(\lambda_i) = \frac{d}{d\lambda_K} f_K(\lambda_K)$;
3. Solve λ_K from $\sum_{i=1}^{K} g_i(\lambda_K) = \delta$;
4. If $\lambda_K > \lambda_{K0}$ then
   $K = K - 1$ and goto step 2;
   else
   For $i = 1$ to K
   Compute λ_i from $g_i(\lambda_K)$ and the solution of λ_K in step 3;

The important feature in the 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 $\lambda_i > \lambda_{i0}$ for some component). If the first derivatives $\frac{d}{d\lambda_i} f_i(\lambda_i)$ are less than zero for all i, and the second derivatives $\frac{d}{d\lambda_i^2} f_i(\lambda_i)$ are greater than zero for all i, then it can be shown that the component ranked lowest according to the derivatives at time zero can be discarded. This happens in step 4 of the algorithm. In other words, the sufficient conditions on the derivatives say that the failure rate decreases over time, and that the rate of decrease gets smaller if time increases. Note that if the derivatives of the growth curve are less regular, specific conditions must be established to determine which components should not be assigned testing time.

The above procedure can be similarly formulated for the fixed testing budget problem. We will not do so here.

Pareto Growth Model As an illustration, let us consider the Pareto distribution and consider the fixed failure rate constraint problem. The failure rate is given by $\lambda_i(t) = \epsilon_{i0}(\epsilon_{i1} + t)^{-\epsilon_{i2}}$, where ϵ_{i0}, ϵ_{i1} and ϵ_{i2} are constants. Hence, we have for the testing time:

$$D_i = \left(\frac{\epsilon_{i0}}{\lambda_i}\right)^{\epsilon_{i2}} - \epsilon_{i1}.$$

The Pareto class of failure-rate distributions is useful because it is a generalization of the exponential, Weibull and gamma classes [11, 14].

One can show that the first derivative is less than zero, and the second derivative is greater than zero, provided ϵ_{i0}, ϵ_{i1} and ϵ_{i2} are all positive. Hence, taking the partial derivative of D_i w.r.t. λ_i, we get in step 2 of the algorithm ($K = N$ in the first iteration):

$$-\frac{\epsilon_{i0}^{\frac{1}{\epsilon_{i2}}}}{\epsilon_{i2}} \lambda_i^{-(\frac{1}{\epsilon_{i2}}+1)} = -\frac{\epsilon_{K0}^{\frac{1}{\epsilon_{K2}}}}{\epsilon_{K2}} \lambda_K^{-(\frac{1}{\epsilon_{K2}}+1)}.$$

In step 3, we have to equate the total failure rate to δ. In this case, we have to do that numerically, since a closed-form expression using the Pareto distribution is too intricate. As soon as we obtain a possible solution for λ_K, we compute the individual failure rate using the relationship

$$\lambda_i = \left(\frac{\epsilon_{i2}}{\epsilon_{K2}} \frac{\epsilon_{K0}^{\frac{1}{\epsilon_{K2}}}}{\epsilon_{i0}^{\frac{1}{\epsilon_{i2}}}} \lambda_K^{-(\frac{1}{\epsilon_{K2}}+1)} \right)^{-(\frac{1}{\frac{1}{\epsilon_{i2}}+1})}.$$

Example 5: Assume the following parameters with Pareto distribution. $\epsilon_{10} = 5$, $\epsilon_{11} = 1$, $\epsilon_{12} = 3$, $\epsilon_{20} = 2$, $\epsilon_{21} = 1$, $\epsilon_{22} = 6$, $\epsilon_{30} = 4$, $\epsilon_{31} = 1$, $\epsilon_{32} = 5$. The optimal policy requires the following failure-rate values: $\lambda_1 = 3.395$, $\lambda_2 = 1.556$, $\lambda_3 = 2.047$. The total testing time is 0.3236.

4. Solutions for Multiple Application Environment

When there are multiple applications in the system, the reliability allocation problem becomes too intricate to solve explicitly. However, in this case its solution can be obtained using non-linear programming software such as AMPL [6]. Let us here show how this procedure works for an example of a 3-component, 3-application system, by specifying and solving it using the RAT tool presented in Section 7.

Example 6: There are three components C_1, C_2 and C_3 which can be used to build three applications A_1, A_2 and A_3. A_1 is built using C_1 and C_2, A_2 is built using C_2 and C_3 and A_3 is built using C_1, C_2, and C_3. Thus, there are multiple

applications, each with a failure-rate constraint. The fixed failure rate constraint problem can thus be formulated as:

$$Minimize \qquad D = D_1 + D_2 + \cdots + D_N,$$

under the constraints:
$\lambda_1 + \lambda_2 \leq \delta_1$ (for application A_1),
$\lambda_2 + \lambda_3 \leq \delta_2$ (for application A_2),
$\lambda_1 + \lambda_2 + \lambda_3 \leq \delta_3$ (for application A_3).

The fixed testing budget problem can be formulated as:

$$Minimize \qquad \lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N,$$

under the constraints:
$D_1 + D_2 \leq d_1$,
$D_2 + D_3 \leq d_2$,
$D_1 + D_2 + D_3 \leq d_3$.

Assume the parameters from Example 3 where the initial failure rates for the three components are the same ($\lambda_{10} = \lambda_{20} = \lambda_{30} = 5/yr$), and the μ values are different (say, $\mu_1 = 1$, $\mu_2 = 2$, $\mu_3 = 3$). Assume that the failure-rate requirements for the three applications are $\delta_1 = 6$, $\delta_2 = 5$, $\delta_3 = 7$. Modeling this as a non-linear optimization problem with multiple constraints in AMPL and using the MINOS solver, we get the following result: $\lambda_1 = 3.818$, $\lambda_2 = 1.909$ and $\lambda_3 = 1.273$. The total testing time for this failure-rate allocation evaluates to 1.207 yrs; (the individual testing times can be computed using the failure-rate allocation for each component). It is interesting to note that the failure-rate constraint for A_3 is strictly satisfied; for A_1 with the failure-rate requirement of $6/yr$, it is not strictly satisfied ($\lambda_1 + \lambda_2 = 5.73$), similarly for A_2 whose requirement is $5/yr$ ($\lambda_2 + \lambda_3 = 3.18$). Now consider an *average* testing policy where the constraint for the application A_3 is strictly satisfied without violating the other constraints. That is, $\lambda_1 = 2.333$, $\lambda_2 = 2.333$ and $\lambda_3 = 2.333$. The total testing time based on this average testing policy evaluates to 1.39 yrs, much larger than that obtained with the optimal testing policy.

5. Software Failure Dependencies and Fault-Tolerant Systems

The basic reliability allocation problem formulation can be extended in various ways. Here we discuss two extensions, software failure dependencies in Section 5.1 and fault tolerance aspects in Section 5.2.

5.1. Software Failure Dependencies

In the above discussions we assume software components fail independently. In reality, this may not be the case. For example, the feature interaction problem[10] describes many incidents where independently developed software components interact with each other unexpectedly, thus causing unanticipated failures. We incorporate this extra failure incidence by introducing *pair-wise* failure rates. Specifically, λ_{ij} represents the failure rate due to the interaction of components C_i and C_j, where $i \leq j$.

The constraints of the original problem are then modified as:

$\sigma_{11}\lambda_1 + \sigma_{12}\lambda_2 + \cdots + \sigma_{1N}\lambda_N + \sum_{\{\forall(i,j)|\sigma_{1i}=1,\sigma_{1j}=1\}} \lambda_{ij} \leq \delta_1 \; (for\ application\ A_1)$,

$\sigma_{21}\lambda_1 + \sigma_{22}\lambda_2 + \cdots + \sigma_{2N}\lambda_N + \sum_{\{\forall(i,j)|\sigma_{2i}=1,\sigma_{2j}=1\}} \lambda_{ij} \leq \delta_2 \; (for\ application\ A_2)$,

$\cdots$

$\sigma_{M1}\lambda_1 + \sigma_{M2}\lambda_2 + \cdots + \sigma_{MN}\lambda_N + \sum_{\{\forall(i,j)|\sigma_{Mi}=1,\sigma_{Mj}=1\}} \lambda_{ij} \leq \delta_M \; (for\ application\ A_M)$.

Subject to the above constraint we need to minimize

$$D = D_1 + D_2 + \cdots + D_N.$$

The fixed testing-time problem can be obtained by adding the pair-wise failure rates in the objective function.

5.2. Fault-Tolerant Systems

In the situation where the system possesses fault tolerant attributes, we can introduce *coverage factors* [18] into the original problem. Coverage is defined as the conditional probability that when a fault is activated, it will be detected and recovered without causing system failure. With c_i denoting the coverage measure for the component C_i, we can reformulate the fixed failure rate constraint case, using $\rho_i = 1 - c_i$, as:

$$Minimize \qquad D = D_1 + D_2 + \cdots + D_N,$$

subject to:
$\sigma_{11}\rho_1\lambda_1 + \sigma_{12}\rho_2\lambda_2 + \cdots + \sigma_{1N}\rho_N\lambda_N \leq \delta_1$ (for application A_1),
$\sigma_{21}\rho_1\lambda_1 + \sigma_{22}\rho_2\lambda_2 + \cdots + \sigma_{2N}\rho_N\lambda_N \leq \delta_2$ (for application A_2),
$\cdots$
$\sigma_{M1}\rho_1\lambda_1 + \sigma_{M2}\rho_2\lambda_2 + \cdots + \sigma_{MN}\rho_N\lambda_N \leq \delta_M$ (for application A_M).

6. Reliability Allocation Solution Framework

We have discussed the reliability allocation problem in terms of two constraints: fixed failure rates or fixed testing budgets. We also discussed the problem to account for

component interactions. The fault tolerant attributes in the system to tolerate component failures are also incorporated. In this section we describe a framework for specifying and solving a general reliability allocation problem, and describe how this procedure is applied to a specific application.

6.1. The Problem Specification and Solution Procedure

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

1. Determine if it is a fixed failure rate constraint problem or a fixed testing budget problem.
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 with each other. 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 non-linear programming problem with appropriate parameters.
8. If the solution is analytically available, obtain it. Otherwise, use the reliability allocation tool (see Section 7), based on the mathematical programming tool AMPL and solver MINOS, to obtain the results.

In the following sub-section we examine a case study where a required reliability allocation problem is specified. We illustrate how the above procedure is applied to the project to obtain numerical solutions for various scenarios.

6.2. A Hypothetical Example

Let us consider a distributed software architecture that is used for switching telephone calls. Different call types will exercise different software modules, and we break up the system in components such that reliability growth models are available for all components. Of course, prerequisite to our analysis is the availability of reliable growth models, but the example will clearly show that it is beneficial to make decisions based on such models.

Table 1 shows the data input in this example. Neither the example nor the data corresponds to existing systems or numbers. We consider 4 types of calls (two types of standard calls, and two type of 1-800 calls), and 5 components (basic processing, scheduling, call processing, and two signal processing modules). The terms 'in' and 'not in' in Table 1 denote which components the applications use. For instance, 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 the table.

With the results obtained for this example, we want to show two things: the necessity to use mathematical optimization techniques to establish an optimal allocation scheme, and the importance of selecting and parameterizing adequate reliability growth models.

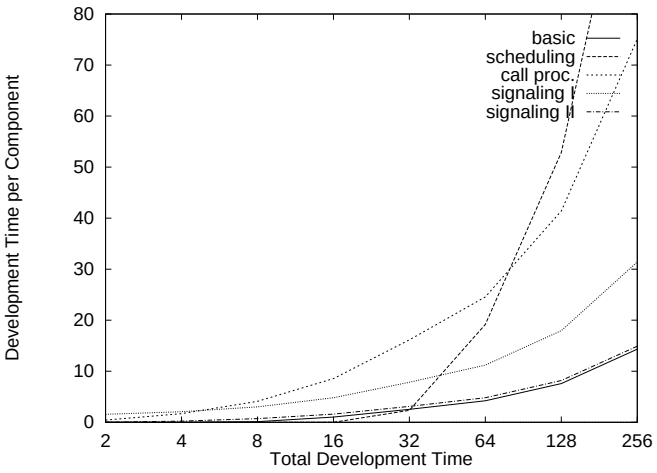

Figure 2. Total available testing time versus the optimal allocation for the components.

Figure 2 shows the total amount of testing time available, versus the time allocated to the individual components. Following the framework in Section 6.1 we solved it as a fixed testing budget problem with multiple applications. We assumed, however, that the testing time is shared by all applications, that is, we consider the special case mentioned in Section 2.2 where the constraints map to a single constraint. Furthermore, applications are weighted based on their relative frequency of occurrence given in Table 1 (the RAT tool described in Section 7 automatically converts this to weights on the component failure rates in the objective function.) Using weights we thus include parts of the operational profile (see, e.g., Chapter 5 in [11]) in the model (see Table 1 for the relative frequencies of the different call applications). We obtained solutions for the testing time ranging from 2 to 256, and assumed no failure dependency or explicit fault-tolerant mechanisms. We input the problem

Component	λ_{i0}	μ_i	standard I	standard 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

Table 1. System components with corresponding parameters growth curve, and applications that use the component.

in the RAT tool, and solved it using AMPL and MINOS.

Figure 2 shows very clearly the dependence of the optimal schedule on the total testing time. For instance, while the scheduling component should not be assigned debugging time if a small budget is available, it takes the largest chunk if the testing budget is large.

The irregular assignment of testing time to individual components in Figure 2 cannot be obtained easily by means other than mathematical modeling. With back-of-the-envelop calculations, one cannot expect to get such precise results, and one would be bound to make inefficient decisions.

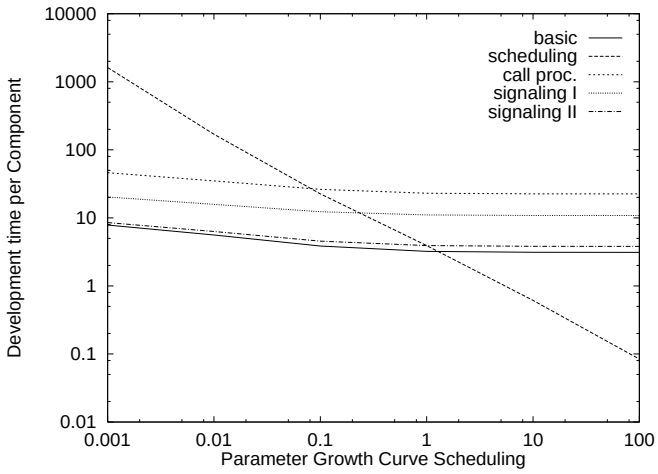

Figure 3. Parameter μ in growth curve of scheduling software, versus the optimal allocation for all components.

Figure 3 shows the parameter λ of the reliability growth curve corresponding to the scheduling software, versus the allocation of testing time to the components. In this case, we took the allowed failure rates per application to be 4, and solved the fixed failure rate constraint problem.

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

The 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 are bound to be sub-optimal.

7. RAT: The Reliability Allocation Tool

We have designed and built a reliability allocation tool (RAT) with a GUI based on a Java Applet. The tool allows multiple applications to be specified and allows 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. Figure 4 shows the GUI. The tool chooses the optimization criteria, where optimizing failure rate implies that the constraint is fixed testing budget and optimizing testing time implies that the constraint is fixed failure rate. Components can be specified in the field named "Components" and applications can be specified in the field named "Applications." The reliability growth distribution can be chosen for each components independently; parameters for these distributions can be specified in the box named "Parameters." At present exponential and Pareto distributions are allowed, but we plan to extend the options to specifying other distributions. In case of a fixed-failure rate constraint, the allowed failure rate for the applications can be specified in the field named "Allowed Failure Rate"; similarly, if testing time is fixed, then this can be specified in the field named "Allowed Debugging Time." Information about the components that have been specified and the applications that have been input are shown in two separate areas.

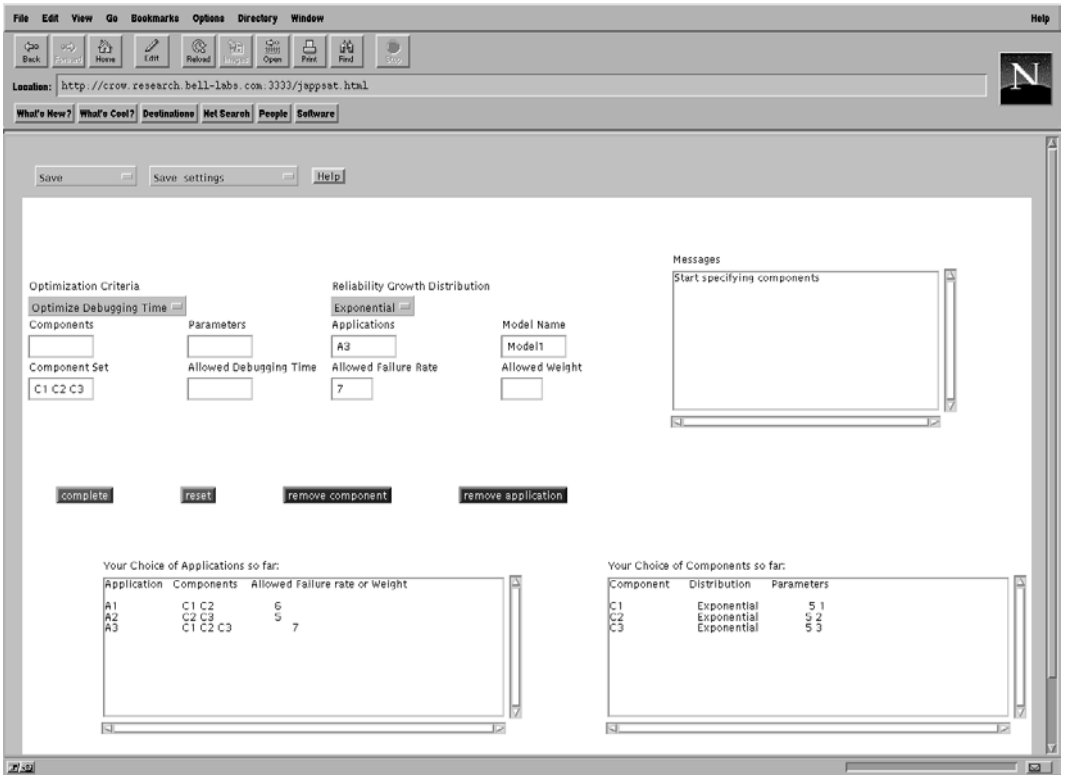

Figure 4. The reliability allocation tool.

As an example of the use of this tool, we consider the fixed-failure rate problem with multiple applications that we considered in Example 6. Figure 4 shows the interface after the components have been chosen and the configurations of the applications have been specified. In Example 6, there were three components C_1, C_2 and C_3, each of which follows an exponential reliability growth model with $\lambda = 5$ and μ values of 1, 2 and 3 respectively. This is shown in the area titled "Your Choice of Components so far." Three applications are configured where application A_1 uses components C_1, C_2, application A_2 uses C_2,C_3 and application A_3 uses C_1, C_2 and C_3. The applications have a failure-rate requirement of 6, 5 and 7 respectively. This is shown in the area titled "Your Choice of Applications so far." This model when solved produces the result as shown in the "Message" area in Figure 5. As presented in Example 6, the results show that the failure rates of C_1, C_2 and C_3 should be brought down to 3.818, 1.909 and 1.273 respectively. This will minimize the total testing time used, while at the same time satisfying the failure rate requirements of all the applications. The "Message" area also shows the testing times that need to be spent on each of the components to bring the failure rates of the components down to the above values. It is seen that the total testing time for the three components is as computed in Example 6.

8. Conclusions

We consider the software component reliability allocation problem for a system with single or multiple applications, each with a pre-specified reliability requirement. The system testing activity is formulated as a combinatorial optimization problem for overall system failure rate or testing time and cost. The relation between failure rates of components and cost to decrease this rate is modeled by various types of reliability growth curves.

We achieve closed-form solutions to problems with only one single application in the system, and we describe how to solve the multiple application problem using non-linear programming techniques. We also examine the interactions between the components in the system, and include inter-component failure dependencies into our modeling formula. In addition to regular systems, we extend the technique to address fault-tolerant systems. We further develop a procedure for a systematic approach to the reliability allocation problem, and describe its application in a case study. Finally, we present the design and implementation of a reliability allocation tool for an easy specification of the problem and an automatic application of the technique.

The presented methodology gives the basic solution approach to the optimization of testing schedules, subject to re-

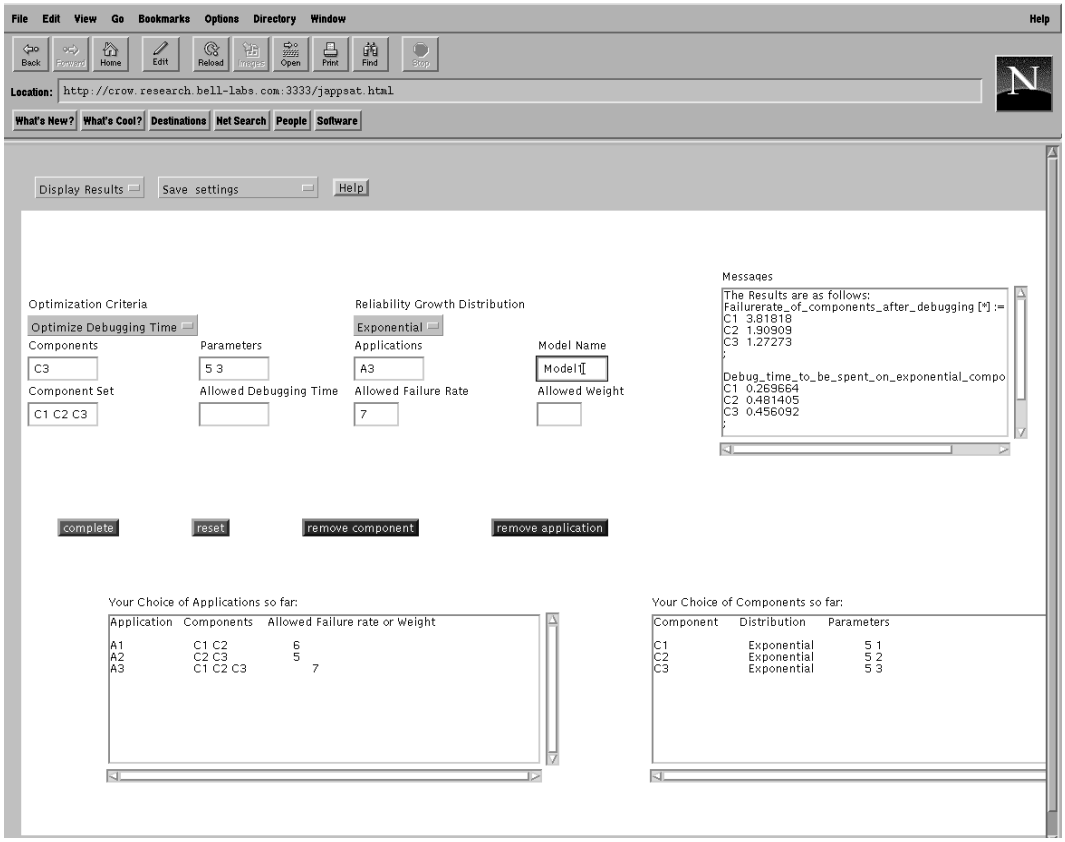

Figure 5. Display of reliability allocation results.

liability 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 will further improve the planning opportunities in software design and testing.

9. Acknowledgement

We wish to thank Chandra M. Kintala of Lucent Bell Labs. 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.), Chapter 11, Marcel Dekker Inc., New York, 1981.

[2] O. Berman and N. Ashrafi,"Optimization Models for Reliability of Modular Software Systems," *IEEE Transactions on Software Reliability*, vol. 19, no. 11, November 1993, pp. 1119–1123.

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

[4] D.W. Coit and A.E. Smith, "Reliability Optimization of Series-Parallel Systems Using A Genetic Algorithm," *IEEE Transactions on Reliability*, vol. 45, 1996.

[5] A.K. Dhingra, "Optimal Apportionment of Reliability and Redundancy in Series Systems Under Multiple Objectives," *IEEE Transactions on Reliability*, vol. 41, no. 4, December 1992, pp.576–582.

[6] R. Fourer et. al., "AMPL: A Modeling Language For Mathematical Programming," The Scientific Press, 1993.

[7] D.E. Fyffe, W.W. Hines, and N.K. Lee, "System Reliability Allocation and a Computational Algorithm," *IEEE Transactions on Reliability*, vol. R-17, 1968, pp. 64–69.

[8] P.M. Ghare and R.E. Taylor, "Optimal Redundancy for Reliability in Series System," *Operational Research*, vol. 17, 1969, pp. 838–847.

[9] A.L. Goel and K. Okumoto, "Time-Dependent Error Detection Rate Model for Software and other Performance Measures," *IEEE Transactions on Reliability*, vol. R-28, no. 3, August 1979, pp.206–211.

[10] N. Griffeth and Y.-J. Lin (ed.), *IEEE Communications Magazine*, Special Issue on Feature Interactions in Telecommunications Systems, August 1993.

[11] M. Lyu (ed.), "Handbook of Software Reliability Engineering," McGraw-Hill and IEEE Computer Society Press, 1996.

[12] 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," *Proceedings of the 7th International Symposium on Software Reliability Engineering*, October/November 1996, pp. 289–298.

[13] K.B. Misra and U. Sharma, "An Efficient Algorithm to Solve Integer Programming Problems Arising in System Reliability Design," *IEEE Transactions on Reliability*, vol. R-40, 1991, pp. 81–91.

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

[15] J. Musa, "Validity of Execution-Time Theory of Software Reliability," *IEEE Transactions on Reliability*, vol. R-28, no. 3, August 1979, pp. 181–191.

[16] Y. Nakagawa and S. Miyazaki, "Surrogate Constraints Algorithm for Reliability Optimization Problems with Two Constraints," IEEE Transactions on Reliability, R-30, 1981, pp. 175–181.

[17] L. Painton and J. Campbell, "Genetic Algorithms in Optimization of System Reliability," *IEEE Transactions on Reliability*, vol. 44, 1995, pp. 172–178.

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

[19] F.A. Tillman, C.L. Hwang, and W. Kuo, "Determining Component Reliability and Redundancy for Optimum System Reliability," *IEEE Transactions on Reliability*, vol. R-26, 1977, pp. 162–165.

[20] 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 Transaction on Software Engineering*, vol. 15, no. 3, March 1989, pp. 345–355.

[21] F. Zahedi and N. Ashrafi, "Software Reliability Allocation Based on Structure, Utility, Price, and Cost," *IEEE Transactions on Software Engineering*, vol. 17, no. 4, April 1991, pp. 345–355.