Optimal Allocation of Testing Resources for Software Reliability
Growth Modeling in Component-Based Software Development

Michael R. Lyu
The Chinese University of Hong Kong

Sampath Rangarajan
Ranch Networks

Aad P. A. van Moorsel
HP Laboratories, Hewlett Packard

Keywords:
reliability allocation
software testing
software reliability engineering
component-based technology

Summary & Conclusions

Component based software development approach has become a trend in integrating modern software systems. To ensure the 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. Efficiency improvement of 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 testing resource allocation problem 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.

We achieve closed-form solutions to the problem for systems with one single application, 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 testing resource allocation problem, and describe its application in a case study of a telecommunications software system. This procedure is automated in a reliability allocation tool for an easy specification of the problem and an automatic application of the technique.
Our methodology gives the basic solution approach to the 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 will further improve the reliability planning accuracy in component-based software development technique.

1 Introduction

1.1 Background

Modern complex software systems are often developed with components supplied by contractors or independent teams under different environments. In particular, component-based software engineering [14, 4] 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 [10, 3, 18], non-linear optimization [24], and heuristic techniques [22, 5]. In [7] the optimal apportionment of reliability and redundancy is considered for multiple objectives using fuzzy optimization techniques. [6] applies a specific reliability growth model for hardware components and determines their optimal testing allocation in order to archive 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 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 [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].

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 [5] and [17]. 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 reliability specification and solution seeking procedure, which has been automated by a software tool, is presented in this paper as an innovative mechanism to handle the difficult yet important testing resource allocation problem.

1.2 Project Applications

Several real projects on Component-based techniques motivate us for this investigation. We describe them as the following case studies.

Distributed Software Systems. Distributed telecommunication systems often serve multiple application types, by executing different software components to meet different reliability requirements. For instance, in telephone switches, 1-800 calls require processing reliability 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 examples in this paper we will make it clear that trustworthy reliability growth curves can help considerably in efficient testing and debugging planning of such systems. Our approach is complementary to [27, 28] which focused on reliability analysis of component-based software systems under different distributed execution scenarios.

Fault-Tolerant Systems. Figure 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 will necessarily include all the layers.

We conjecture that error propagation between layers only occurs in one direction, namely 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 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., if it does not exhibit a 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.

We will show how fault-tolerant mechanisms can be included in the problem formulation for reliability specification and resource allocation, 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., [2, 26]. Our intent in 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, 26] and the development-oriented methods in this paper can provide a powerful mechanism in component-based and object-based software system design.

1.3 Notations

\[ M \] number of software applications
\[ N \] number of software components
\[ A_i \] application \( i \), where \( 1 \leq i \leq M \)
\[ C_j \] component \( j \), where \( 1 \leq j \leq N \)
\[ R_i(t) \] pre-specified reliability requirement for application \( i \)
\[ \delta_i \] sum of failure rates of the components in application \( A_i \)
\[ D_j \] the testing time invested in component \( C_j \)
\[ D \] total testing time invested
\[ \omega_j \] weighting function for \( D_j \) in the total testing time

Figure 1. Layered software architecture model.
\( d_i \)  
fixed amount of testing time allowed for \( A_i \)

\( \lambda_j \)  
failure rate in component \( j \) during testing

\( \lambda_{j0} \)  
initial failure rate in component \( j \) at time \( 0 \)

\( \sigma_{ij} \)  
usage indicator for application \( A_i \) on component \( C_j \)

\( \mu_j \)  
the failure decay parameter for component \( j \) during testing

\( \delta \)  
fixed total failure rate constraint

\( \theta \)  
Lagrange multiplier

\( \Delta \)  
partial derivative operator

\( f_j(\lambda_j) \)  
the function which relates failure rate (\( \lambda_j \)) to testing time

\( abs() \)  
absolute function

\( \epsilon_{j0}, \epsilon_{j1}, \epsilon_{j2} \)  
parameters in the Pareto function with respect to component \( C_j \)

\( c_j \)  
coverage measure for component \( C_j \)

\( \rho_j \)  
\( 1 - c_j \)

### 1.4 Paper Organization

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 4.1 discusses how the solutions for the problems in multiple application environment can be obtained. The results are extended to consider software failure dependencies and to incorporate fault-tolerant systems. Section 5 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.

### 2 Problem Specification

The project case studies in Sections 1 and 2 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 \( C_1, \cdots, C_N \), 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 minimization of the failure rate of all the components as the objective of our optimization problem. Subsequently, we discuss these two variations of the optimization problem, and they are 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 \( \delta_i \) is the sum of failure rates of the components in application \( A_i, i = 1, \ldots, M \). Furthermore, we assume that the testing time \( D_j \) invested in component \( j \) decreases the failure rate \( \lambda_j \) according to some reliability growth model. Note 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:

\[
\text{Minimize } \quad D = D_1 + D_2 + \cdots + D_N,
\]

subject to the constraints:

\[
\sigma_{11} \lambda_1 + \sigma_{12} \lambda_2 + \cdots + \sigma_{1N} \lambda_N \leq \delta_1 \text{ (for application } A_1 \text{)},
\]

\[
\sigma_{21} \lambda_1 + \sigma_{22} \lambda_2 + \cdots + \sigma_{2N} \lambda_N \leq \delta_2 \text{ (for application } A_2 \text{)},
\]

\[
\vdots
\]

\[
\sigma_{M1} \lambda_1 + \sigma_{M2} \lambda_2 + \cdots + \sigma_{MN} \lambda_N \leq \delta_M \text{ (for application } A_M \text{)},
\]

where \( \sigma_{ij} = 1 \) if \( A_i \) uses component \( C_j \), and \( \sigma_{ij} = 0 \) otherwise. (1)

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 \( D_j \) are equally ‘important’ (costly) among the \( N \) components. If this is not the case one can apply weight \( w_j \) to each component testing time \( D_j \) 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 a closed-form solution for the problem with a single application, and in Section 4.1 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:

\[
\text{Minimize} \quad \lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N,
\]

subject to the constraints:

\[
\sigma_1 D_1 + \sigma_2 D_2 + \cdots + \sigma_N D_N \leq d_1 \text{ (for application } A_1), \\
\sigma_2 D_1 + \sigma_2 D_2 + \cdots + \sigma_N D_N \leq d_2 \text{ (for application } A_2), \\
\vdots \\
\sigma_M D_1 + \sigma_M D_2 + \cdots + \sigma_M D_N \leq d_M \text{ (for application } A_M),
\]

where \(d_1, d_2, \ldots, d_M\) are the fixed amount of testing times that are available for components used by applications \(A_1, A_2, \ldots, A_M\), respectively. As in the previous problem, all variables are positive. Weight functions for the failure rates \(\lambda_j\) 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 4.1, we discuss the case of single and multiple applications, respectively.

3 Solutions for Single Application Environment

When there is only one 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 \([11, 20]\) relates the failure rate \(\lambda_j\) (for component \(j\)) with the invested testing time \(D_j\) through:

\[
\lambda_j = \lambda_{j0} e^{-\nu_j D_j}.
\]
$\lambda_0$ is the initial failure rate at time 0, and $\mu_j$ is the decay parameter. Over an infinite time interval $\frac{\lambda_0}{\mu_j}$ faults will be found. Note that $\lambda_j$ is a function of time, although we do not explicitly express it in the notation. With this reliability growth model in common use, we 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:

$$
\text{Minimize } D = \sum_{i=1}^{N} \frac{1}{\mu_j} \ln \left( \frac{\lambda_j}{\lambda_0} \right),
$$

subject to:

$$
\lambda_1 + \lambda_2 + \cdots + \lambda_N \leq \delta. \tag{4}
$$

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

$$
F(\lambda_1, \cdots, \lambda_N) = D + \theta \left( (\lambda_1 + \cdots + \lambda_N) - \delta \right), \tag{5}
$$

where $\theta$ is a Lagrange multiplier, and we get the following solution for the obtained failure rates:

$$
\lambda_1 = \frac{\delta}{1 + \frac{\mu_1}{\mu_2} + \cdots + \frac{\mu_1}{\mu_N}}, \quad \lambda_2 = \frac{\mu_1}{\mu_2} \lambda_1, \quad \cdots \quad \lambda_N = \frac{\mu_1}{\mu_N} \lambda_1. \tag{6}
$$

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

$$
D_1 = \frac{1}{\mu_1} \ln \left[ \frac{\lambda_{10}}{1 + \frac{\mu_1}{\mu_2} + \cdots + \frac{\mu_1}{\mu_N}} \right]. \tag{7}
$$

Note that $D_1$ is negative 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 ($\lambda_{10} = \lambda_{20} = \lambda_{30} = 5/\text{yr}$). Assume that the application requirement states that the failure rate of the application ($\delta$) cannot be more than 6/yr. Also assume that all the $\mu$ values are the same and equals 1. In this case, we note that $\lambda_1 = \lambda_2 = \lambda_3 = 2/\text{yr}$ 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, $\lambda_{10} = 5/\text{yr}$, $\lambda_{20} = 6/\text{yr}$, $\lambda_{30} = 7/\text{yr}$). Assume that $\delta = 6/\text{yr}$. Also assume that the $\mu$ values are all identical and equal to 1. In this case, again we find that the required $\lambda$ 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 ($\lambda_{10} = \lambda_{20} = \lambda_{30} = 5/\text{yr}$), and that the $\mu$ 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 $\ln(\frac{5\times1.834}{6}) + \frac{1}{2}\ln(\frac{5\times1.834}{3}) + \frac{1}{3}\ln(\frac{5\times1.834}{2}) = 1.49$. 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:

$$
\text{Minimize } \lambda = \lambda_1 + \lambda_2 + \cdots + \lambda_N,
$$

subject to the constraint

$$
D_1 + D_2 + \cdots + D_N \leq D. \tag{8}
$$

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

9
\[ F(D_1, \cdots, D_N) = \lambda + \theta((D_1 + D_2 + \cdots + D_N) - D), \]  

where \( \theta \) is the Lagrange multiplier. The resulting solutions for \( \lambda \)'s are

\[
\lambda_{10}(-\mu_1 e^{-\mu_1 D_1}) = \lambda_{20}(-\mu_2 e^{-\mu_2 D_2}) = \cdots = \lambda_{N0}(-\mu_N e^{-\mu_N D_N}),
\]

and the solutions for \( D \)'s are

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

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

\[
\cdots
\]

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

The above equations determine how the testing times should be allocated to the different components. The minimum \( \lambda \) 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 \( \lambda_{j0} \)'s and the \( \mu_j \) 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 \( \lambda_{j0} \)'s or the \( \mu_j \) 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 \( \mu \) 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 [17] 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 $f_j$, that is:

$$D_j = f_j(\lambda_j).$$  \hspace{1cm} (12)

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 $\lambda_j = \lambda_{j0}$. That is:

$$\text{abs}\left(\frac{d}{\lambda_j} f_j(\lambda_j)|_{\lambda_j=\lambda_{j0}}\right) \geq \text{abs}\left(\frac{d}{\lambda_{j+1}} f_{j+1}(\lambda_{j+1})|_{\lambda_{j+1}=\lambda_{j+1}|_{\lambda=\lambda_{j0}}}\right),$$

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

1. $K = N$;
2. For $j = 1$ to $K$
   
   express $\lambda_j$ as a function $g_j(\lambda_K)$ of $\lambda_K$, by equating
   $$\frac{d}{\lambda_j} f_j(\lambda_j) = \frac{d}{\lambda_K} f_K(\lambda_K);$$
3. Solve $\lambda_K$ from $\sum_{j=1}^{K} g_j(\lambda_K) = \delta$;
4. If $\lambda_K > \lambda_{K0}$ then
   
   $K = K - 1$ and goto step 2;
   
   else
   
   For $j = 1$ to $K$
   
   Compute $\lambda_j$ from $g_j(\lambda_K)$ and the
   
   solution of $\lambda_K$ 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 $\lambda_j > \lambda_{j0}$ for some component). If the first derivatives $\frac{d}{\lambda_j} f_j(\lambda_j)$ are less than zero for all $j$, and the second derivatives $\frac{d^2}{\lambda_j^2} f_j(\lambda_j)$ are greater than zero for all $j$, 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, which we will not elaborate here.

**Pareto Growth Model**  
As an illustration, let us consider the Pareto distribution for the fixed failure rate constraint problem. The failure rate is given by 
\[ \lambda_j(t) = \epsilon_j0(\epsilon_j1 + t)^{-\epsilon_j2}, \]
where \( \epsilon_j0, \epsilon_j1 \) and \( \epsilon_j2 \) are constants. Hence, we have for the testing time:

\[ D_j = \left( \frac{\epsilon_j0}{\lambda_j} \right)^{\epsilon_j2} - \epsilon_j1. \]  

(14)

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

One can show that the first derivative is less than zero, and the second derivative is greater than zero, provided \( \epsilon_j0, \epsilon_j1 \) and \( \epsilon_j2 \) are all positive. Hence, taking the partial derivative of \( D_j \) w.r.t. \( \lambda_j \), we get in step 2 of the algorithm (\( K = N \) in the first iteration):

\[ -\frac{1}{\epsilon_j0}\lambda_j\left( \frac{1}{\epsilon_j2} + 1 \right) = -\frac{1}{\epsilon_j0\epsilon_j2\lambda_j K}\left( \frac{1}{\epsilon_j2} + 1 \right). \]  

(15)

In step 3, we have to equate the total failure rate to \( \delta \). 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 \( \lambda_K \), we compute the individual failure rate using the relationship

\[ \lambda_j = \left( \frac{\epsilon_j2 \epsilon_j0 K^2}{\epsilon_j2 \lambda_j K} \right)^{\left( \frac{1}{\epsilon_j2} + 1 \right)} \]  

(16)

**Example 5:**  
Assume the following parameters with Pareto distribution. \( \epsilon_{10} = 5, \epsilon_{11} = 1, \epsilon_{12} = 3, \epsilon_{20} = 2, \epsilon_{21} = 1, \epsilon_{22} = 6, \epsilon_{30} = 4, \epsilon_{31} = 1, \epsilon_{32} = 5 \). For \( \delta = 7 \), which is the sum of failure rates of the components, 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 [8].
4.1 An Example

We demonstrate how our procedure works for an example of a 3-component, 3-application system, by specifying and solving it.

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:

$$\text{Minimize} \quad D = D_1 + D_2 + \cdots + D_N,$$

under the constraints:

$$\lambda_1 + \lambda_2 \leq \delta_1 \text{ (for application } A_1),$$

$$\lambda_2 + \lambda_3 \leq \delta_2 \text{ (for application } A_2),$$

$$\lambda_1 + \lambda_2 + \lambda_3 \leq \delta_3 \text{ (for application } A_3).$$

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 $\mu$ 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.
4.2 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 and fault tolerance aspects.

In the above discussions we assume software components fail independently. In reality, this may 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. We incorporate this extra failure incidence by introducing pair-wise failure rates. Specifically, $\lambda_{ij}$ represents the failure rate due to the interaction of components $C_i$ and $C_j$, where $i \leq j$. 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. We compute these failure rates 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. Note that while failures involving two components may not be neglectable, failures involving three or more components are usually rare [13].

The constraints of the original problem are then modified as:

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

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

$$\cdots$$

$$\sigma_{M1} \lambda_1 + \sigma_{M2} \lambda_2 + \cdots + \sigma_{MN} \lambda_N + \sum_{\{i, j\} | \sigma_{M, j} = 1} \lambda_{ij} \leq \delta_M \quad (\text{for application } A_M). \quad (19)$$

Subject to the above constraint we need to minimize

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

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

In the situation where the system possesses fault tolerant attributes, we can 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 will be detected and recovered without causing system failure. With $c_j$ denoting the coverage measure for the component $C_j$, we can reformulate the fixed failure rate constraint case, using $\rho_j = 1 - c_j$, as:

$$\text{Minimize} \quad D = D_1 + D_2 + \cdots + D_N,$$

subject to:

$$\sigma_{11} \rho_1 \lambda_1 + \sigma_{12} \rho_2 \lambda_2 + \cdots + \sigma_{1N} \rho_N \lambda_N \leq \delta_1 \quad (\text{for application } A_1),$$
\[ \sigma_{21} \rho_1 \lambda_1 + \sigma_{22} \rho_2 \lambda_2 + \cdots + \sigma_{2N} \rho_N \lambda_N \leq \delta_2 \text{ (for application } A_2), \]
\[ \cdots \]
\[ \sigma_{M1} \rho_1 \lambda_1 + \sigma_{M2} \rho_2 \lambda_2 + \cdots + \sigma_{MN} \rho_N \lambda_N \leq \delta_M \text{ (for application } A_M). \] (20)

Note that 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.

5 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 formulate a framework for specifying and solving a general reliability allocation problem, and apply this procedure to a specific application. Finally, we describe a tool to automate the procedure.

5.1 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 mathematical programming tools and solvers 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.
Table 1. System components with corresponding parameters growth curve, and applications that use the component.

<table>
<thead>
<tr>
<th>Component</th>
<th>$\lambda_{j0}$</th>
<th>$\mu_j$</th>
<th>standard I</th>
<th>standard II</th>
<th>1-800 I</th>
<th>1-800 II</th>
</tr>
</thead>
<tbody>
<tr>
<td>basic</td>
<td>10</td>
<td>1.0</td>
<td>in</td>
<td>in</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>scheduling</td>
<td>20</td>
<td>1.0</td>
<td>in</td>
<td>in</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>call proc.</td>
<td>200</td>
<td>0.2</td>
<td>not in</td>
<td>in</td>
<td>in</td>
<td>in</td>
</tr>
<tr>
<td>signal I</td>
<td>200</td>
<td>0.5</td>
<td>in</td>
<td>in</td>
<td>in</td>
<td>not in</td>
</tr>
<tr>
<td>signal II</td>
<td>200</td>
<td>1.0</td>
<td>in</td>
<td>in</td>
<td>not in</td>
<td>in</td>
</tr>
<tr>
<td>frequency</td>
<td>20</td>
<td>0.5</td>
<td>0.3</td>
<td>0.1</td>
<td>0.1</td>
<td></td>
</tr>
</tbody>
</table>

5.2 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 into components such that reliability growth models are available for all components. Of course, prerequisite to our analysis is the availability of reliability 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 $\mu$ and $\lambda$, 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 depicts, with a given total amount of testing time available (shown in the x-axis), the time allocated to test individual components (shown in the y-axis). Following the framework in Section 5.1 we solved it as a fixed testing budget problem with multiple applications. We assume, 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, which can be automatically converted to weights on the component failure rates in the objective function. Using relative frequencies for different call applications we thus include parts of the operational profile (see, e.g., Chapter 5 in [15]) in the model. We obtained solutions for the testing time ranging from 2 to 256, assuming no failure dependency or explicit fault-tolerance mechanisms.

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. Without a systematic approach such as ours, one cannot expect to get such precise results, and one would be bound to make inefficient decisions.

Figure 3 plots changes of the allocation of testing time to the components, while the decay parameter $\mu$ of the reliability growth curve for the scheduling software varies from 0.001 to 100. 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.

5.3 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 component 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. When the model is solved, it will produce results shown in the “Message” area.

The major computational effort required for the RAT tool is on the MINOS solver. For a 5-component system described in Section 5.2, it takes about a few milliseconds to obtain the result in a Sun Untra 5 workstation. As the time requirement only linearly increases 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

We wish to thank Chandra M. Kintala of Lucent Bell Labs. for many of his valuable suggestions and comments for this work. The work described in this paper was substantially supported by a grant from the Research Grants Council of the Hong Kong Special Administrative Region (Project No. CUHK 4193/00E).

References

Figure 4. The reliability allocation tool.


20


AUTHORS

Dr. Michael R. Lyu; Computer Science & Engineering Dep’t; The Chinese Univ. of Hong Kong; Shatin, HONG KONG.

Internet (e-mail): lyu@cse.cuhk.edu.hk

Michael R. Lyu (Senior Member IEEE) received the BS degree in Electrical Engineering from National Taiwan University in 1981, the MS degree in Computer Engineering from University of California, Santa Barbara, in 1985, and the Ph.D. degree in Computer Science from University of California, Los Angeles, in 1988. He is currently an Associate Professor at the Computer Science and Engineering Dep’t 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. Dr. Lyu initiated the first International Symposium on Software Reliability Engineering (ISSRE) in 1990. He is the program chair for ISSRE’96, PRDC’99, and WWW10, and has served in program committees for numerous international conferences. He also serves as General Chair for ISSRE’2001. He is the editor for two book volumes: Software Fault Tolerance, published by Wiley in 1995 and the Handbook of Software Reliability Engineering, published by IEEE and McGraw-Hill in 1996. He is on the editorial board for IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Reliability, and Journal of Information Science and Engineering.

Dr. Sampath Rangarajan; Ranch Networks, Morganville, NJ 07751

Internet (e-mail): sampath@ranchnetworks.com

Sampath Rangarajan received a Ph.D in computer sciences in 1990 and a M.S in electrical and computer engineering in 1987, both from the University of Texas at Austin. Currently, he is VP of Technology at Ranch Networks, a start-up company building 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 in Murray Hill, NJ. Previously, Dr. Rangarajan was an assistant professor in the Department of Electrical and Computer Engineering at Northeastern University in Boston (1992-1996) and a research associate in the University of Maryland Institute for Advanced Computer Studies (UMIACS) (1990-1992). His research interests are in the areas of distributed computing and networking.

Dr. Aad P. A. van Moorsel; Software Technologies Laboratory; Hewlett-Packard Laboratories; Palo Alto, CA 94304 USA.

Internet (e-mail): aad@hpl.hp.com

Aad P. A. van Moorsel received his MS in mathematics from the University of Leiden, The Netherlands (1989) and his PhD degree in computer science from the University of Twente, The Netherlands (1993). He developed
performance, reliability and performability evaluation techniques for telecommunication networks and distributed systems. From 1994 to 1995 he was a research assistant at 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 the area of performability engineering, software fault-tolerance and system management. Since November 1999, he works with HP Laboratories, where he currently heads the E-Services Software Research Department. This department specializes in ground breaking research in manageability and reliability of middleware and Internet applications.