Modular Operational Test Plans for Inferences on Software Reliability Based on a Markov Model

Jayant Rajgopal[1] and *Mainak Mazumdar*

Department of Industrial Engineering
University of Pittsburgh
Pittsburgh, PA 15261

[1] Corresponding Author: Tel. (412) 624 9840, Fax. (412) 624 9831, e-mail:*rajgopal@engrng.pitt.edu*

ABSTRACT

This paper considers the problem of assessing the reliability of a software system that can be decomposed into a finite number of modules. It uses a Markovian model for the transfer of control between modules in order to develop the system reliability expression in terms of the module reliabilities. An operational test procedure is considered in which only the individual modules are tested and the system is considered acceptable if, and only if, no failures are observed. The minimum number of tests required of each module is determined such that the probability of accepting a system whose reliability falls below a specified value R_0 is less than a specified small fraction β. This sample size determination problem is formulated as a two-stage mathematical program and an algorithm is developed for solving this problem. Two examples from the literature are considered to demonstrate the procedure.

Keywords: Software reliability; Modular Operational Tests; Sample Size Determination; Mathematical Programming

Modular Operational Test Plans for Inferences on Software Reliability Based on a Markov Model

It is a commonly recognized fact that software costs are far higher than those of hardware when considering the total cost of a computer-based system. Consequently, over the last several years characterizing, measuring and evaluating software reliability have become crucial activities when developing large computerized systems. In this paper we address one particular aspect, namely operational testing for software reliability (the reader is referred to [1] for a discussion on this) where the objective is to obtain an estimate of the reliability actually achieved. In this type of testing the software is subjected to the same statistical distribution of inputs that one expects it to encounter in operation. Instead of actively looking for failures, operational testing waits for the failures to surface through repeated testing using random samples from the hypothesized statistical distribution of the input values. Software testing models used in practice tend mostly to treat the system as a "monolithic block" [2]. That is, for purposes of testing, the entire system is tested as one unit without paying particular attention to its individual component modules and the logical connections that exist among them. In [2] the authors advocate the use of "atomic" models in which the reliability assessment of the entire system is done based on the testing of its constituent modules or components. They state that such models "allow easy reliability assessment, determination of optimal testing strategies, etc."

In this paper we consider an operational test procedure for the assessment of software reliability that is based on testing of its individual modules. The procedure uses a particular Markov model first proposed in [3] and also considered in [4]. The proposed

plan consists of conducting tests only on the modules that comprise the system, and in order to draw inferences on the system reliability it uses the mathematical model that relates the system reliability to the component reliabilities and the operational profile. In essence, we ask the following question: "assuming that the particular Markov model is an accurate representation of the software system, what is the minimum level of testing that should be performed on each module so that the probability of reaching a wrong conclusion is less than a specified small fraction?" We have previously considered such similar "design" problems in a *hardware* system framework for *component* tests when the objective is to make inference on *system* reliability. This work has covered several different system configurations under several different modes of sampling (e.g. [5], [6], [7]), but this approach has never been applied to software systems. This paper represents a first effort in this direction and an attempt to carry over the ideas that we have developed earlier for *system-based component tests* to the area of software reliability testing.

In summary, this paper addresses operational testing at the module level. It uses a Markov model of the transfer of control between modules in order to compute the system reliability. The approach presented here is based strictly on classical statistics (as opposed to a Bayesian framework) and also assumes that module interactions do not affect system failure. In other words it assumes that there is no failure when transitioning between modules, and that when the software system operates an individual module does not influence the failure of another. In the following sections, we first present a brief discussion of software reliability testing and overview some of the statistical models developed for this purpose that are relevant to the work contained herein. This is

followed by the development of the actual test plans that result in a two-stage mathematical programming formulation. A cutting-plane algorithm [8] is developed for this formulation and the entire procedure is illustrated via numerical examples.

1. SOFTWARE TESTING

As proposed in [4], a software *system* is defined here as a "collection of programs and system files such that the system files are accessed and altered only by the programs in the collection." Each element in this collection will be called a *module* - for instance, a module might be a program, a subprogram, or a file. The performance (and hence the reliability) of the system clearly depends on that of each of the constituent modules, as well as on the relationship between these modules and the system; in this regard a software system is not that much different from any other type of system. However, unlike other systems the actual relationship between system and module reliabilities is quite unique and depends on the specific definition of software reliability as well as on the structure of the overall system. This point is addressed in the next section.

Testing takes place almost continuously throughout the developmental life-cycle of a software system. There are distinct types of tests associated with different phases, e.g., *design-based* tests, *structural* (or white-box) tests, *functional* (or black-box) tests, *debug* tests, *operational* tests, *feature* tests, and *load* tests. This paper focuses on operational testing; the interested reader is referred to [9] or [10] for more details on testing as a part of the software development cycle. A comparison of the operational testing procedure versus debug testing insofar as their ability to deliver reliability in the final product is given in [1]. Although no clear-cut winner emerges in this discussion, the authors point out several advantages that operational testing possesses.

The advantages of modular testing are emphasized in [2] as well as [4]. With software development it is usually the case that most nontrivial systems are developed by a large team of programmers and software engineers, as opposed to a single individual or group. Often, responsibility for different pieces of a software system is assigned to different organizational entities at different locations and at different points in time. While testing of the complete system is essential, it stands to reason that a sound and well-planned program of testing at the module level has the potential to reduce the effort involved in system level testing. Modular testing also allows for more flexibility since it can be done at different times and locations. However, a key point to be kept in mind is that the test plans for the individual modules must be *system-based*, and designed in such a way that statistically valid inferences may be drawn about the overall system from the results of module tests.

Markov models have been presented by Cheung [3] as well as Littlewood [11] to describe the switching that takes place between modules or sub-systems within a software system. In the latter paper, the author also considers the failure caused by the "interfaces" between any two modules, that is, those failures that are introduced when subsystems are put together in a larger system. In [12], the authors investigate the effects of shared use on the reliability of modular software. For this purpose they model the distribution of user demands at the module level by a continuous time Markov chain which is referred to as the operational profile.

In [13] the authors address the problem of estimating software reliability from data on component failures by using a Bayesian framework. While this work is related to ours, it differs in that we approach the problem from the classical statistical perspective

of drawing inferences on system reliability. In particular, we formulate a design problem for the testing of modules in terms of classical statistical concepts related to testing of hypothesis in a strictly non-Bayesian framework. The objective is to use the results from only the modular tests to make inferences on the overall software reliability. For the sake of simplicity, we also do not model the possibility of failures in the interfaces among the modules or module interactions. However, in earlier work on "hardware" systems we have explicitly modeled interface failures [14], and we believe that this extension can be achieved by modifying the software reliability model to include a combination of modular and system testing.

In looking at the test design problem, Cheung [3] develops a *sensitivity coefficient* for each module that captured its effect on the system's reliability. Testing is then advocated by concentrating on the modules with the largest values of these coefficients. However, this work does not present any precise numbers or methodology to derive the exact amount of test effort. Using the same model, the authors in [4] suggest allocating the targeted system reliability goal among the components and then testing the individual components to verify whether the component reliabilities meet the allocated goals at a specified level of confidence. However, this procedure does not consider the system configuration any longer once the reliability goal has been allocated. As discussed in [15], this method may lead to erroneous estimates of sample size requirements for component testing.

To approach the problem of drawing statistically valid inferences about a system based on tests of its constituent modules, one requires a mathematical model that expresses software system reliability in terms of the module reliabilities. Using the model

proposed in [3], we follow the approach of *system-based component testing* that was first described in [16]. For a full discussion of system-based component test plans and the issues involved, the reader is referred to [15].

2. SOFTWARE RELIABILITY

Reliability of a software system can be defined in a fashion similar to that of hardware or any general system. It may be viewed as a function of time, in which case it is the probability of failure-free operation for some specified mission time under specific conditions. Alternatively, it may be viewed from the perspective of general use on a variety of different inputs, in which case it is the probability that it will correctly process a randomly chosen input. With the emphasis on reuse this latter viewpoint is generally more useful, and is the one that we consider in this paper. Given that the precise nature of the application on which some general-purpose software system will be used is not known in advance, one must quantify software use by defining a suitable distribution or *operational profile*. Essentially, one may view this as a description of possible inputs to the software along with the relative frequency with which each input might be encountered, or equivalently, the probability that the input will be encountered in a randomly chosen trial. The operational profile and the program structure allow us to develop a statistical distribution of inputs for each individual module and in the modular testing procedure being considered here we will use inputs that are randomly selected from this distribution.

Associated with each type of input to the system there is a specific path, or sequence of modules over which control is transferred by the system. The model proposed in [3] is based on the assumption that transfer of control between the individual

modules of a software system takes place according to a Markov chain. In essence, the probability p_{ij} that control transfers from module i to another module j is independent of how module i was entered. It is assumed that there are n such modules within the system where module 1 represents the initial state (e.g., this could be the main program called by the user). It is also assumed that when the program is successfully completed, control is transferred to a terminal module S (e.g., this could be the operating system) with probability p_{iS} of being entered from state i. Note that $p_{iS} + \sum_{j=1}^{n} p_{ij} = 1$.

The probabilities defined above represent values for a completely reliable system. Since we consider a system that is presumably not so, it is assumed that each module in the system has faults and that module i has reliability r_i, i.e., P(system fails any time control enters module i) = $(1-r_i)$. To model the system an additional state F is now defined. This state represents program failure and since each module is assumed to be not one hundred percent reliable,, it follows that this state can be entered from any module. Also note that unlike the transient states 1,2,…,n, the states S and F are absorbing states and represent (respectively) successful completion of the program and failure. The Markov chain thus has n+2 states and a transition matrix Q where $q_{ij} = r_i p_{ij}$ for i=1,2,…,n and j=1,2,…,n,S; $q_{iF} = 1-r_i$ for i=1,2,…,n; and $q_{FF}=q_{SS}=1$, with all other $q_{ij}=0$. As an example consider the system given in Figure 1; the transition matrix Q corresponding to this system is given by

$$Q = \begin{bmatrix} 0 & 0.6r_1 & 0.3r_1 & 0 & 0.1r_1 & 1-r_1 \\ 0 & 0 & 0 & r_2 & 0 & 1-r_2 \\ 0 & 0 & 0 & r_3 & 0 & 1-r_3 \\ 0.3r_4 & 0.2r_4 & 0.4r_4 & 0.1r_4 & 0 & 1-r_4 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ S \\ F \end{matrix}$$
$$\begin{matrix} 1 & 2 & 3 & 4 & S & F \end{matrix}$$

Assuming that state 1 represents the initial state (i.e., control is initiated by module 1), one may use a standard method of computing absorption probabilities [17], to show that the reliability R_S of the system can be computed via

$$R_S = \sum_{i=1}^{n} (I_n - \hat{Q})^{-1}_{1i} r_i p_{iS} \tag{1}$$

where I_n is the identity matrix of order n, $\hat{Q}$ is the $(n \times n)$ submatrix of Q obtained by dropping its last two rows and columns. In other words, the reliability of the system is the inner product of the first row of $(I_n - \hat{Q})^{-1}$ and the column of Q corresponding to state S. For the system shown in Figure 1, (1) yields

$$R_S = \frac{0.1r_1 - 0.01r_1r_4 - 0.02r_1r_2r_4 - 0.04r_1r_3r_4}{1 - 0.1r_4 - 0.2r_2r_4 - 0.18r_1r_2r_4 - 0.4r_3r_4 - 0.09r_1r_3r_4}.$$

The assumption made in this work is similar to that in earlier work ([3], [4], [18]); the p_{ij} values are known but the r_i's will in general be unknown. The assumption that the p_{ij} are known is based on the fact that the operational profile provides the probability of encountering a specific input, and the logic of the software provides the specific path of modules through which control is transferred for that input. Thus one may compute the probability that control transfers from module i to module j for some random input. Clearly, this probability depends on the operational profile. Thus one must pay special

attention to the development of the operational profile, and the approach proposed in this paper is accurate as long as the operational profile is accurate. However, this is in general true whenever a statistical approach is used to test software; this point is also discussed at length by Musa et al. [9]. If the operational profile changes over time one would have to recompute the probabilities p_{ij} for the new profile, but the formulae used to compute these would be identical to the ones used initially as long as the software itself is not altered. While it is thus reasonable to assume that the p_{ij}'s could be estimated for a given operation profile, it would not be correct to assume that the reliability values (r_i) will be known for individual modules. Indeed, if these values were known, there would not be any reason to conduct the reliability tests.

It should be mentioned that this method of computing the reliability expression as given in (1) can be quite inconvenient without the use of software that can perform the requisite symbolic linear algebra, although packages such as *Mathematica* [19], which was used in this paper, make this very easy to do. An alternative approach to computing system reliability for the Markovian model was proposed by Siegrist [18]. This approach uses conditional probability arguments and dispenses with the need to invert matrices, and using this approach the author also provided closed-form expressions for certain special types of software systems.

3. A MODULAR TEST PLAN

Given that we can compute the reliability of a software system using the approach described in the preceding section we now develop a procedure for drawing inferences on system reliability by tests on its individual modules. The procedure followed is to run k_i unit tests on module i where each test uses input data drawn randomly from the use

distribution. The values of k_i, i=1,2,…,n are to be determined. Let X_i denote the number of failures observed among the k_i test instances of module i. It is assumed that X_1, X_2 ,…, X_n are mutually independent random variables. The following rule is then used for the overall system: "Infer that the system is reliable as long as $\Sigma_i X_i = 0$." Gal [16] proposed this rule for general systems. It is a reasonable rule for establishing software reliability, because as soon as a failure occurs, presumably every effort will be made thereafter to remove the fault that caused the failure. In designing the test plan the objective is to find the smallest values for k_i (i.e., to minimize test effort) which ensure that the plan will rarely infer as reliable a system whose reliability is below some minimally acceptable value, say R_0. Specifically, we would like to ensure that the probability of the plan accepting a system with reliability $R_S \leq R_0$ to be below some specified small fraction β.

Note that with inputs drawn at random from the use distribution, the proposed test format effectively subjects each module to a series of independent functional tests with inputs that are representative of the frequency of use in the final operational environment. Now suppose that the reliability of module i is r_i as defined in the previous section; for notational convenience we use the vector $\boldsymbol{r} \in \mathrm{R}^n$ to denote the module reliabilities, i.e., $\boldsymbol{r} = [r_1, r_2, \ldots, r_n]$. When the ith module is tested independently k_i different times, the total number of observed failures, X_i, will have a binomial distribution with parameters k_i and $1\text{-}r_i$, and mean = $k_i(1\text{-}r_i)$. Since k_i is likely to be large, and $1\text{-}r_i$ likely to be small, we can approximate the distribution of X_i by a Poisson distribution with parameter $k_i(1\text{-}r_i)$. Since the tests on the modules are independent of each other, it follows that $\sum_i X_i$ follows an approximate Poisson distribution with parameter $\sum_i k_i(1\text{-}r_i)$.

Thus the probability that the proposed test plan infers that the system is reliable is given by $P(\sum_i X_i=0) = \exp\left(-\left(\sum_i k_i(1-r_i)\right)\right)$. The problem of designing a test plan may now be stated as follows:

Minimize $k_1+k_2+\ldots+k_n$

subject to $\exp\left(-\sum_i k_i(1-r_i)\right) \le \beta$ for all $\{\boldsymbol{r}\in R^n \mid \boldsymbol{0}\le \boldsymbol{r} \le \boldsymbol{1},\ R_S(\boldsymbol{r})\le R_0\}$ (2)

$k_i\ge 0$ and integer for $i=1,2,\ldots,n$.

Note that we use $R_S(\boldsymbol{r})$ to denote the reliability of the system as a function of the module reliabilities contained in the vector $\boldsymbol{r}$ and that $R_S(\boldsymbol{r})$ is computed via Equation (1) of the previous section. Also note that there are infinitely many nonnegative vectors $\boldsymbol{r}$ for which $R_S(\boldsymbol{r})\le R_0$ and the constraint specified in (2) must hold for <u>each</u> of these. Thus the optimization problem is an integer linear program with infinitely many constraints (also referred to as a semi-infinite integer linear program). However, not all of these constraints are going to be active at the optimum, so that our interest is in generating only those vectors $\boldsymbol{r}$ that are likely to yield active constraints in the above program. In order to solve the problem we make the observation that (2) is equivalent to solving the following sub-problem:

Minimize $\Sigma_i\, k_i(1-r_i)$, st $\boldsymbol{r}\in R^n$, $\boldsymbol{0}\le \boldsymbol{r} \le \boldsymbol{1}$, $R_S(\boldsymbol{r})\le R_0$ (3)

and requiring that its optimum value be $\ge -ln\,(\beta)$. Thus the overall optimization problem may be rewritten as the following two-stage mathematical program:

Minimize $k_1+k_2+\ldots+k_n$

st $\left\{\text{Minimum } \Sigma_i\, k_i(1-r_i), \text{ st } \boldsymbol{r}\in R^n, \boldsymbol{0}\le \boldsymbol{r} \le \boldsymbol{1},\ R_S(\boldsymbol{r})\le R_0\right\} \ge -ln\,(\beta)$ (4)

$k_i\ge 0$ and integer for $i=1,2,\ldots,n$.

In the "inner" stage we fix the values of k_i and solve a problem in the r_i; this problem has a linear objective but a nonlinear constraint in addition to simple upper and lower bounds on the r_i. In the "outer" problem we solve a linear program in the nonnegative integers k_i. Thus the overall problem is to find from among all the values of $\boldsymbol{k}=[k_1, k_2,\ldots, k_n]$ for which the inner problem yields an optimum value that is $\geq -ln\ (\beta)$, the particular vector for which $\Sigma_i k_i$ is minimized.

4. A SOLUTION STRATEGY FOR THE OPTIMIZATION PROBLEM

We now describe a cutting plane algorithm for solving this problem. In order to simplify the procedure the integer restrictions on k_i will be ignored and the optimum continuous solution will be rounded up to obtain the final values for k_i. Clearly, this solution is guaranteed to be feasible, and if the optimal values are large any resulting deviation from the optimum will be negligible.

STEP 0: For each $i=1,2,\ldots,n$ define a vector $\boldsymbol{r}^i$ by setting all except the i^{th} element of $\boldsymbol{r}^i$ equal to 1 and solving for the value of this i^{th} element in the equation $R_S(\boldsymbol{r}^i)=R_0$. Then use each of these n vectors to define n initial linear constraints of the form $\Sigma_j\ k_j(1-r_j) \geq -ln\ (\beta)$. Define the initial linear program (LP) as Minimize $\Sigma_j k_j$ subject to these n constraints and $k_i \geq 0$. Call the solution $\boldsymbol{k}$.

STEP 1: Solve the (nonlinear) optimization problem given by (3) using the vector $\boldsymbol{k}$ to define its objective (note that instead of the objective in (3), one may equivalently maximize $\Sigma_i k_i r_i$). Suppose the optimum solution is given by $\boldsymbol{r}=[r_1, r_2, \ldots, r_n]$. If $\Sigma_i\ k_i(1-r_i) \geq -ln\ (\beta)$ then stop; the current vector $\boldsymbol{k}$ is optimal. Otherwise proceed to Step 2.

STEP 2: Redefine the current linear program with the extra constraint $\Sigma_i\, k_i(1-r_i) \geq -ln\,(\beta)$. Clearly, this cuts out the current solution vector $\boldsymbol{k}$, which is now infeasible. Solve the linear program and obtain a new solution vector and replace $\boldsymbol{k}$ with this vector. Then go to Step 1.

The linear program in Step 2 is solved efficiently since one could start with the optimum basis from the prior iteration and use the dual simplex method to restore feasibility. The nonlinear program in Step 1 is in general nonconvex and not as easy to solve. However, the reliability function $R_S(\boldsymbol{r})$ involves only polynomial terms and it may be rearranged so that the subproblem is restated as a signomial geometric program [20], for which efficient algorithms have been developed. We have used an efficient adaptation [21] of an algorithm originally described in [22], which solves such problems by solving a sequence of *posynomial* geometric programs.

Note that in Step 0, we effectively assume that all modules except module i are perfect and then find the value of the maximum reliability for module i which would still render the system unreliable (thus, if r_i is above this value the system is reliable and if r_i is below this value then it is not). This procedure is repeated in turn for each i. Clearly, each of these vectors satisfies the constraints of the subproblem specified in the LHS of (4) and thus generates a valid constraint for the main problem. For the problems that we tested the optimal values of $\boldsymbol{k}$ tended to be very close to this initial vector, and one heuristic alternative might be to dispense with the (hard-to-solve) subproblem altogether and adopt this vector $\boldsymbol{k}$. However, more testing with various values of R_0 and β are required to draw any definitive conclusions about the quality of this solution.

5. NUMERICAL ILLUSTRATIONS

Example 1: **C**onsider the sequential system pictured in Figure 2 which was motivated by a radar software system discussed in [23] and presented in [18]. We consider values of 0.95 for R_0 and 0.05 for β. The reliability R_S for this system derived using the procedure in Section 2 yields

$$R_S(r_1, r_2, r_3) = \frac{0.008 r_1 r_2 r_3}{(1-0.8r_1)(1-0.4r_2)(1-0.4r_3) - 0.08 r_1 r_2 (1-0.4r_3) - 0.016 r_1 r_2 r_3}$$

The nonlinear constraint for the subproblem requires that $R_S(\boldsymbol{r}) \leq R_0$. Rearranging terms, this reduces to the constraint

$$0.8r_1 + 0.4r_2 + 0.4r_3 - 0.24r_1r_2 - 0.32r_1r_3 - 0.16r_2r_3 + [0.112+(0.008/R_0)]r_1r_2r_3 \leq 1$$

and in the subproblem we maximize $k_1r_1 + k_2r_2 + k_3r_3$ subject to the above constraint and the restriction that $r_1, r_2, r_3 \in [0,1]$. The vector $\boldsymbol{k}$=[2564.35, 856.78, 287.59] was obtained at Step 0 (the initial LP thus has 3 constraints generated by $\boldsymbol{r}^1$, $\boldsymbol{r}^2$ and $\boldsymbol{r}^3$). The algorithm converged to the optimum solution $\boldsymbol{k}$ = [2564.6, 857.2, 288.8], which is very close to the starting solution. Rounding up, we would thus test 2,565 random inputs on the first module, 858 on the second, and 289 on the third, and the system would be accepted as reliable as long as we observe no failures from these tests.

Example 2: For a larger example, consider the system represented in Figure 3, which is based on an example presented in [4]. This software system has a total of 12 modules with the transition probabilities as given by the numbers alongside the respective arcs. When the procedure of Section 2 was used to compute the reliability of this system and the constraint $R_s(\boldsymbol{r}) \leq R_0$ rearranged, the resulting expression yielded a polynomial of order

10 with a total of 170 terms; the exact expression for system reliability is not reported on account of its size, but is available from the authors. The expression for reliability was computed using *Mathematica*. Once again, the procedure discussed earlier was used to generate 12 constraints for the initial LP and a starting solution $\boldsymbol{k}$ obtained by solving this LP. Upon applying the proposed algorithm it was found that this starting solution was optimal. It yielded (after rounding up) the following vector of required test sizes: $\boldsymbol{k}$ = { 288, 543, 111, 543, 543, 486, 100, 111, 270, 216, 270, 216}.

One interesting aspect of the numbers obtained is their relationship to the sensitivity of the various modules. Poore et al. [1], compute values for these sensitivities from the transition matrix in order to assess the relative importance of each module to system reliability. These values that they obtain, when normalized to a scale from 0 to 1 yield the vector [0.075, 0.147, 0.030, 0.147, 0.147, 0.132, 0.027, 0.030, 0.073, 0.059, 0.073, 0.059]; thus for example, modules 2, 4 and 5 have the greatest impact on system reliability and the system reliability is roughly twice as sensitive to these modules as it is to modules 9 and 11. The vector $\boldsymbol{k}$ shows that modules 2, 4 and 5 need to be tested the most (543 cases) and roughly twice as much as modules 9 and 11. In fact, the vector $\boldsymbol{k}$ when normalized to a scale of 0 to 1 yields the vector [0.078, 0.147, 0.030, 0.147, 0.147, 0.131, 0.027, 0.030, 0.073, 0.058, 0.073, 0.058], which is almost identical to the sensitivity vector provided in [4]. In other words, this example suggests that the relative testing effort may be directly proportional to the degree of sensitivity exhibited by the system reliability towards each module, which is an intuitively appealing result. However, at this stage, this remains only a conjecture that is based on the results for a

few particular systems. In any event, the actual number of test cases must of course, be determined by solving the problem formulated herein.

6. CONCLUSIONS

Using a particular Markov model for software reliability we have provided a procedure for finding the minimum number of test cases required for the different modules such that there is no more than a small prespecified probability of accepting a software system whose reliability is less than a prescribed value. It is worth mentioning that since the required number of tests is based on the assumption that the software will not be accepted if even a single failure occurs in the module tests, the procedure is likely to yield results that are quite stringent in character. Our approach does not account for "module interactions." Differences between the reliability of subsystems in isolation and when operating within the complete system may arise even if no failure occurs during a transition. An extension of the work contained herein will need to investigate this possibility.

Finally, it is also worth mentioning that if the cost of testing were different from one module to the other, this would be easy to incorporate into our solution procedure. The only change would be in the objective function of the linear program, which would now be $\Sigma_i c_i k_i$ instead of $\Sigma_i k_i$ where c_i is the test cost for module i.

ACKNOWLEDGEMENTS

The authors wish to thank Dr. Richard Linger, who first brought this problem to their attention. They would also like to thank the associate editor and three anonymous

referees for several constructive comments and suggestions that helped improve the paper.

REFERENCES

[1] P. G. Frankl, R. G. Hamlet, B. Littlewood and L. Strigini, "Evaluating Testing Methods by Delivered Reliability," *IEEE Transactions on Software Engineering*, **24**, pp. 586-601, 1998.

[2] C. Smidts and D. Sova, "An Architectural Model for Software Reliability Quantification: Sources of Data," *Reliability Engineering and System Safety*, **64**, pp. 279-290, 1999.

[3] R. C. Cheung, "A User-Oriented Software Reliability Model," *IEEE Transactions on Software Engineering*, **SE-6**, pp. 118-125, 1980.

[4] J. H. Poore, H. D. Mills, and D. Mutchler. "Planning and Certifying Software System Reliability," *IEEE Software*, pp. 88-99, January 1993.

[5] J. Rajgopal and M. Mazumdar, "Designing Component Test Plans for Series System Reliability via Mathematical Programming," *Technometrics*, **37**, pp. 195-212, 1995.

[6] J. Rajgopal and M. Mazumdar, "A System Based Component Test Plan for a Series System, with Type-II Censoring," *IEEE Transactions on Reliability*, **45**, pp. 375-378, 1996.

[7] J. Rajgopal and M. Mazumdar, "Minimum Cost Component Test Plans for Evaluating Reliability of a Highly Reliable Parallel System," *Naval Research Logistics*, **44**, pp. 401-418, 1997.

[8] W. I. Zangwill, *Nonlinear Programming: A Unified Approach*, Prentice-Hall, Englewood Cliffs, N.J., 1969.

[9] J. D. Musa, A. Iannino and K. Okumoto, *Software Reliability*, McGraw-Hill Publishing Co., N.Y., 1990.

[10] W. C. Hetzel, *The Complete Guide to Software Testing*, QED Information Sciences, Inc., Wellesley, MA, 1988.

[11] B. Littlewood, "A Reliability Model for Systems with Markov Structure," *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, **24**, pp. 172-177, 1975.

[12] J. F. Meyer, B. Littlewood, and D. R. Wright, "Dependability of Modular Software in a Multiuser Operational Environment," *Proceedings of the Sixth International Symposium on Software Reliability Engineering*, pp. 170-179, 1995.

[13] S. Kuball, J. May and G. Hughes, "Building a System Failure Rate Estimator by Identifying Component Failure Rates," *Proceedings of the Tenth International Symposium on Software Reliability Engineering*, pp. 32-41, 1999.

[14] J. Rajgopal, M. Mazumdar, and S. V. Majety, "Optimum Combined Test Plans for Systems and Components," *IIE Transactions*, **31**, pp. 481-490, 1999.

[15] R. G. Easterling, M. Mazumdar, F. W. Spencer and K. V. Diegert, "System Based Component Test Plans and Operating Characteristics: Binomial Data," *Technometrics*, **33**, pp. 287-298, 1991.

[16] S. Gal, "Optimal Test Design for Reliability Demonstration," *Operations Research*, **22**, pp. 1236-1242, 1974.

[17] E. Cinlar, *Introduction to Stochastic Processes*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1975

[18] K. Siegrist, "Reliability of Systems with Markov Transfer of Control, *IEEE Transactions on Software Engineering*, **14**, pp. 1049-1053, 1988.

[19] S. Wolfram, *The Mathematica Book* (3rd edition), Cambridge University Press and Wolfram Media, Inc., Champaign, Ill., 1996.

[20] C. Beightler and D. T. Phillips, *Applied Geometric Programming*, John Wiley & Sons, Inc., New York, 1976

[21] J. Rajgopal and D. L. Bricker, "An Algorithm for Solving the Posynomial GP Problem, Based on Generalized Programming," Department of Industrial Engineering, University of Pittsburgh, Technical Report No. TR95-10, 1995 (forthcoming in *Computational Optimization and Applications*).

[22] M. Avriel, R. S. Dembo, and U. Passy., "Solution of Generalized Geometric Programs," *International Journal for Numerical Methods in Engineering*, **9**, pp. 149-168, 1975.

[23] E. C. Soistman, and K. B. Ragsdale, "Combined Hardware/Software Reliability Prediction Methodology," Rome Air Development Center Contract Report OR-18-173, Vol. II, 1984.

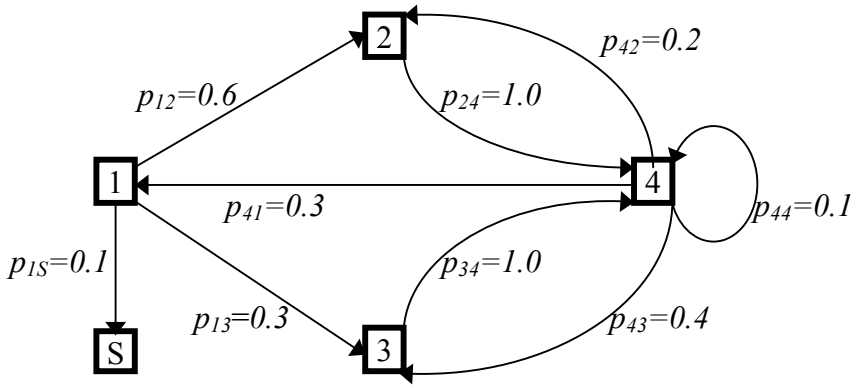
2
$p_{42}=0.2$
$p_{12}=0.6$
$p_{24}=1.0$
1
4
$p_{41}=0.3$
$p_{44}=0.1$
$p_{1S}=0.1$
$p_{34}=1.0$
$p_{13}=0.3$
$p_{43}=0.4$
S
3

FIGURE 1

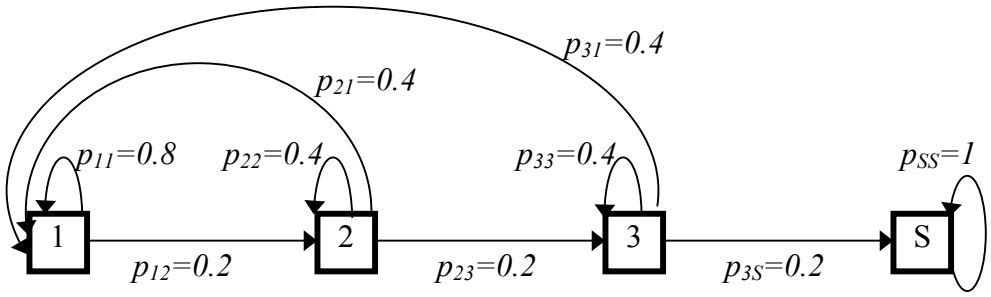

FIGURE 2

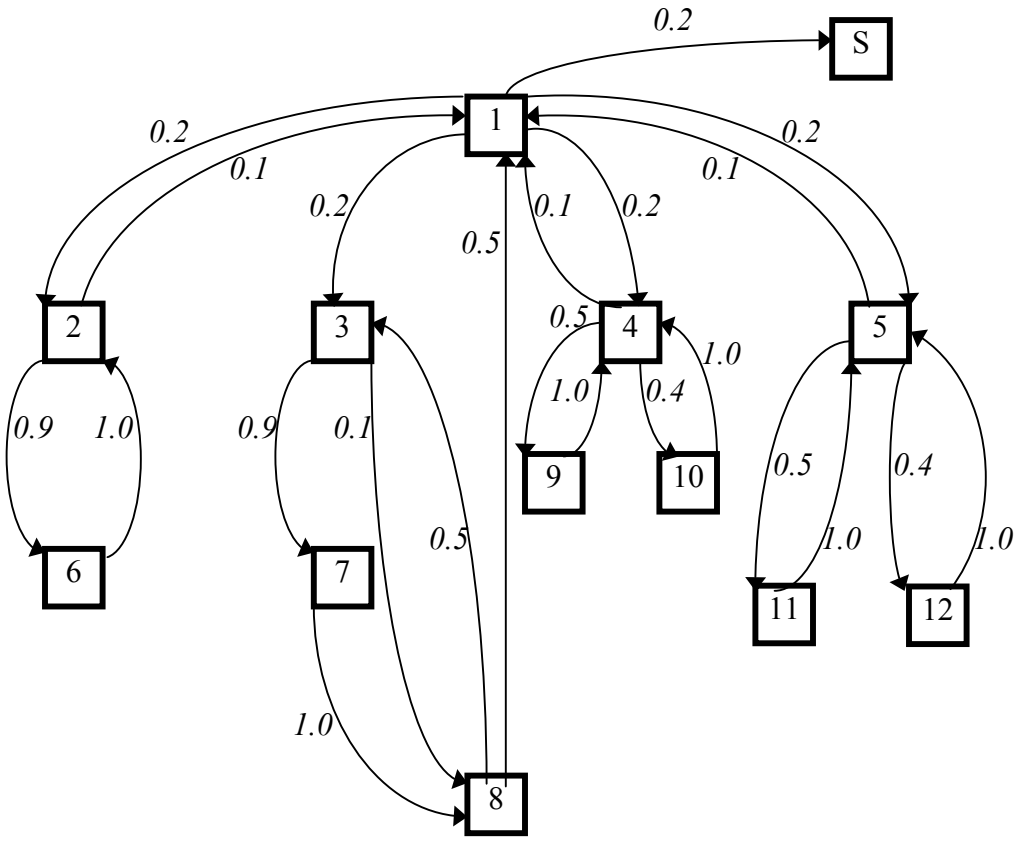
S
1
2
3
4
5
6
7
8
9
10
11
12
0.2
0.2
0.1
0.2
0.1
0.2
0.1
0.2
0.5
0.5
1.0
1.0
0.4
0.9
1.0
0.9
0.1
0.5
0.5
0.4
1.0
1.0
1.0

FIGURE 3

Captions for Figures

Figure 1: A modular software system with transition probabilities.

Figure 2: Software system for Example 1

Figure 3: Software system for Example 2