Modular Test Plans for Certification of Software Reliability

Jayant Rajgopal[1] (Senior Member, IIE)
Associate Professor of Industrial Engineering

Mainak Mazumdar (Member, IIE)
Professor of Industrial Engineering

Subba Rao Majety (Student Member, IIE)

Vikram Talada (Student Member, IIE)

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 certifying 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. A test procedure is considered in which only the individual modules are tested and the system is certified if, and only if, no failures are observed. The minimum number of tests required of each module is determined such that the probability of certifying 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 Tests; Sample Size Determination; Mathematical Programming

1. Introduction

It is a well-known fact that the cost of software is far higher than the cost of hardware when considering the total cost of a computer-based system. With the rapid growth in the use of such systems in almost every walk of life, it has become critical to explicitly consider software reliability, especially because the penalty costs associated with software failures can be very significant. Over the last decade or so, planning and evaluating software reliability have become crucial activities when developing large computerized systems. This is especially true in light of the growing emphasis on software that is reusable (as opposed to software that is written for a terminal mission), where it is essential to demonstrate that the system will perform reliably for a variety of end-user applications.

In this paper we address the problem of drawing cost-effective statistical inferences about the reliability of a software system through a dedicated program of testing. Such programs are commonly used during the design and development of a complex hardware system, and this paper adapts the same ideas to the special case of software systems. As proposed by Poore, Mills and Mutchler [1], 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 individual module and the relationship between these modules and the system; in this regard a software system is quite similar to any other system. However, 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.

In this paper we focus on software systems that can be decomposed into a finite number of functionally independent modules.

In testing software, one could assemble the entire system and then test it. However, it is often the case that different organizational entities are assigned responsibility for different pieces of a software system, and it may be more beneficial from the perspective of both cost and time to test these individual modules or components for the purpose of drawing inferences about the overall system. In order to do this, there must exist mathematical models that express software system reliability in terms of the module reliabilities. Cheung [2], who has proposed a reliability model for a software system that expresses the system reliability in terms of the component reliabilities, advocates such testing by concentrating on the modules with the largest values of the *sensitivity coefficients.* It is not clear here how much, or if at all, the less sensitive components should be tested if this approach is used. Poore et al. [1] 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. This procedure does not consider the system configuration any longer once the reliability goal has been allocated. As discussed by Easterling, Mazumdar, Spencer and Diegert [3], this method may lead to estimates of overly conservative sample size requirements for component testing. In this paper we follow the approach of *system-based component testing* that has been used here as well as in an earlier paper by Gal [4]. For a full discussion of system-based component test plans, the reader is referred to the article by Easterling et al. [3].

We focus on *system-based* modular test plans that explicitly take into account the system-level requirements when designing the modular test plans, while simultaneously trying to do so with a minimum number of component tests. In essence, we attempt to answer the following question: "What is the minimum amount of testing required for each module of the system to ensure that there is a very low probability of accepting a system that would be considered definitely unacceptable?" Similar situations have been addressed in detail for various general systems by the authors (e.g., [5], [6]), but this approach has never been applied in particular to software systems with their unique reliability definitions; this paper represents a first effort in this direction. In the remainder of the paper we first discuss and define what is meant by software reliability. This is followed by the development of the actual test plans that result in a two-stage mathematical programming formulation. An algorithm is developed for this formulation and the entire procedure is then illustrated via numerical examples. It is our belief that mathematical programming formulations of the kind discussed here are apt to occur in the context of determination of sample size requirements in many similar situations, and statisticians should become familiar with these techniques.

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 applications, in which case it is the probability that it will correctly

process a randomly chosen application. With the emphasis on reuse this latter viewpoint is generally more useful, and is the one that we consider in this paper. The term *failure* in the context of software reliability implies a result other than what was expected from the software for a set of inputs. Unlike most general systems, software does not fail because of wear and tear. Rather, failure is caused by such things as poor program design, logical errors and incorrect / inadequate coding or instructions; these are referred to as *faults* [7]. In general, the failure probability does not increase with time other than in the sense that the longer an application runs, the higher the probability that it will encounter a fault at some point (if such faults do indeed exist).

There are several approaches for evaluating the reliability of software. In this paper, we use a model originally proposed in [2] for the sort of system we consider. 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. Specifically, software use is typically modeled by some *use distribution* that represents the relative frequency with which the software will be used on a particular type of application. The randomly chosen application referred to in the previous paragraph may be assumed to come from this distribution. Associated with each type of application there is a specific path, or sequence of modules over which control is transferred by the system, and based on the use distribution one could then evaluate the probability that control will transfer from one particular module to another.

The model proposed by Cheung 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 perfect system. Since we consider a system that is imperfect, it is therefore 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 imperfect, 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 successful completion of the program and failure respectively. 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 [8], to show that the reliability R_S of the system can be computed via

$$R_S = \sum_{i=1}^{n} (I_n - \hat{Q})^{-1}_{1i} f_i \qquad (1)$$

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 and $f_i = r_i p_{iS}$ is the probability of successful completion in one step from module i. 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 important thing to note here is that in most applications, the p_{ij} values will be accurately known but the same is not true about the r_i's.

It should be mentioned that this method of computing the reliability expression can be quite inconvenient without the use of software that can perform the requisite symbolic linear algebra, although packages such as *Mathematica* [9], which was used in this instance, make this very easy to do. An alternative approach to computing system

reliability for the Markovian model was proposed by Siegrist [10]. This approach uses conditional probability arguments and dispenses with the need to invert matrices, and using this approach Siegrist 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 test k_i instances of module i using input data drawn randomly from the use distribution, where 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 component i. It is assumed that X_1 , X_2 ,…, X_n are mutually independent random variables. The following rule is then used to certify the overall system: "Infer that the system is reliable as long as $\Sigma_i X_i = 0$." Gal [4] 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 certify 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.

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 $1-r_i$, and mean = $k_i(1-r_i)$. Since k_i is likely to be large, and $1-r_i$ likely to be small, we can approximate the distribution of X_i by a Poisson distribution with parameter $k_i(1-r_i)$. From the assumed independence of the testing of the different modules, it follows that $\sum_i X_i$ follows an approximate Poisson distribution with parameter $\sum_i k_i(1-r_i)$.

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 $k_1+k_2+\ldots+k_n$

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

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

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:

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

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:

$$\text{Minimize } k_1+k_2+\ldots+k_n$$

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

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

In the "inner" stage we fix the values of k_i and solve a problem in r_i, *which* has a linear objective but a nonlinear constraint in addition to simple upper and lower bounds. In the "outer" problem we solve a linear program in the nonnegative integers k_i. The overall idea is thus to find the values of k_i for which $\Sigma_i k_i$ is minimized and for which the inner problem yields an optimum value that is $\ge -ln\,(\beta)$.

4. A Solution Strategy for the Optimization Problem

We now develop 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: Define an initial linear program (LP) with objective given by Minimize $\Sigma_i k_i$, st $k_i\ge 0$. Define an initial nonnegative starting solution $\boldsymbol{k}=[1,1,\ldots,1]$.

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 not as easy to solve. However, the reliability function $R_S(\boldsymbol{r})$ typically involves only polynomial terms and it may be rearranged so that the subproblem is restated as a signomial geometric program [11], for which efficient algorithms have been developed. We have used an efficient adaptation of an algorithm originally developed by Avriel, Dembo and Passy [12], which solves such problems by solving a sequence of *posynomial* geometric programs [13].

Also note that the choice of the vector $\boldsymbol{k}$=[1,1,…,1] is arbitrary and was used only because the solution to the LP defined in Step 0 is given by $\boldsymbol{k}=\boldsymbol{0}$ which is inadequate to generate a vector $\boldsymbol{r}$ in Step 1. One could also start with several initial constraints added to the LP defined in Step 0. For instance, one could calculate for each i=1,2,…,n a vector $\boldsymbol{r}^i$ by setting $r_j^i=1$ for all $j\neq i$ and solving for the value of r_i^i in the equation $R_S(\boldsymbol{r}^i)=R_0$. Clearly, each of these vectors satisfies the constraints of the subproblem specified by the LHS of (4) and thus generates a valid constraint for the main problem. One could thus start with n initial constraints in the LP and obtain the starting vector $\boldsymbol{k}$ in Step 0 by solving this LP.

For the problems 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 by Soistman and Ragsdale [14], and presented in [10]. 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 $0 \leq r_1, r_2, r_3 \leq 1$. The initial vector $\boldsymbol{k}$=[2564.35, 856.78, 287.59] at Step 0, was obtained using the procedure described in the last paragraph of Section 4 (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 instances of the first module, 858

instances of the second, and 289 instances of the third, and the system would be accepted or certified 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 [1]. 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 $R_s(\boldsymbol{r}) \geq R_0$ 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 on request 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, when the vector $\boldsymbol{k}$ is normalized on a 0 to 1 scale we obtain 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 by Poore et al. In other words, the relative testing effort is directly proportional to the degree of sensitivity exhibited by the system reliability towards each module, which is an intuitively appealing result. The actual number of test cases must of course, be determined by the solving the problem formulated herein.

6. Guaranteeing Probability of Type I error

In this paper we have provided a procedure for finding the minimum number of tests required for the different modules such that the probability of certifying a software system whose reliability is less than a specified value is guaranteed to be less than a preassigned amount. The required number of tests is based on the assumption that the software will not be certified if a single failure occurs in the component tests. This requirement may in some situations be considered to be too stringent, for it runs the risk that a highly reliable system may occasionally be denied certification. This risk is the probability of Type I error. It will not be possible in general, to guarantee small values of both probabilities of Type I and Type II errors by requiring that certification can occur only if no failures are observed. In order to guarantee probabilities of both kinds of errors, the certification rule needs to be suitably modified. One possibility is the sum rule [5] which states that the system will be certified if, and only if, $\sum_i X_i \leq m$, where both m and the k_i's will need to be determined. As Poore et al. have observed in [1], now the

required number of test for each module will be larger than in the situation where only the Type I error probability is considered.

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.

REFERENCES

[1] Poore, J. H., Mills, H. D. and Mutchler, D. (1993) Planning and Certifying Software System Reliability. *IEEE Software*, 88-99.

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

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

[4] Gal, S. (1974) Optimal Test Design for Reliability Demonstration. *Operations Research*, **22**, 1236-1242.

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

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

[7] Kuo, W. (1992) Software Reliability. *Maynard's Industrial Engineering Handbook*, 4th edition (W. K. Hodson, Editor-in-Chief), 11.116-11.122.

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

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

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

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

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

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

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

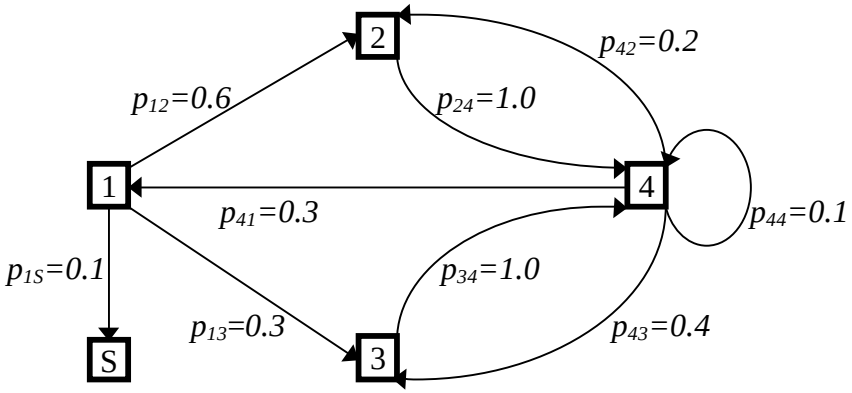
2
1
4
3
S
$p_{12}=0.6$
$p_{24}=1.0$
$p_{42}=0.2$
$p_{41}=0.3$
$p_{44}=0.1$
$p_{1S}=0.1$
$p_{34}=1.0$
$p_{13}=0.3$
$p_{43}=0.4$

FIGURE 1

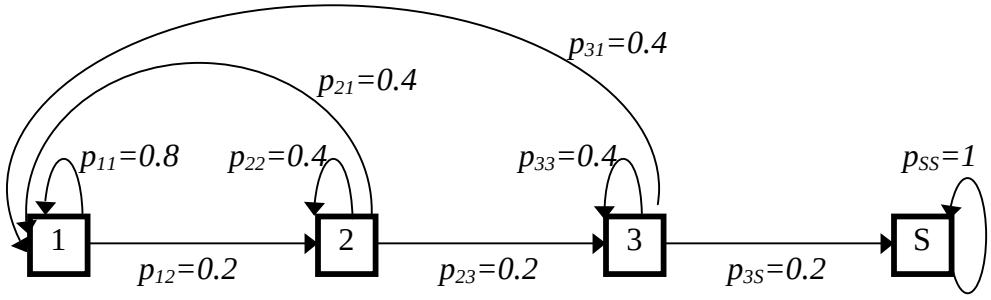

FIGURE 2

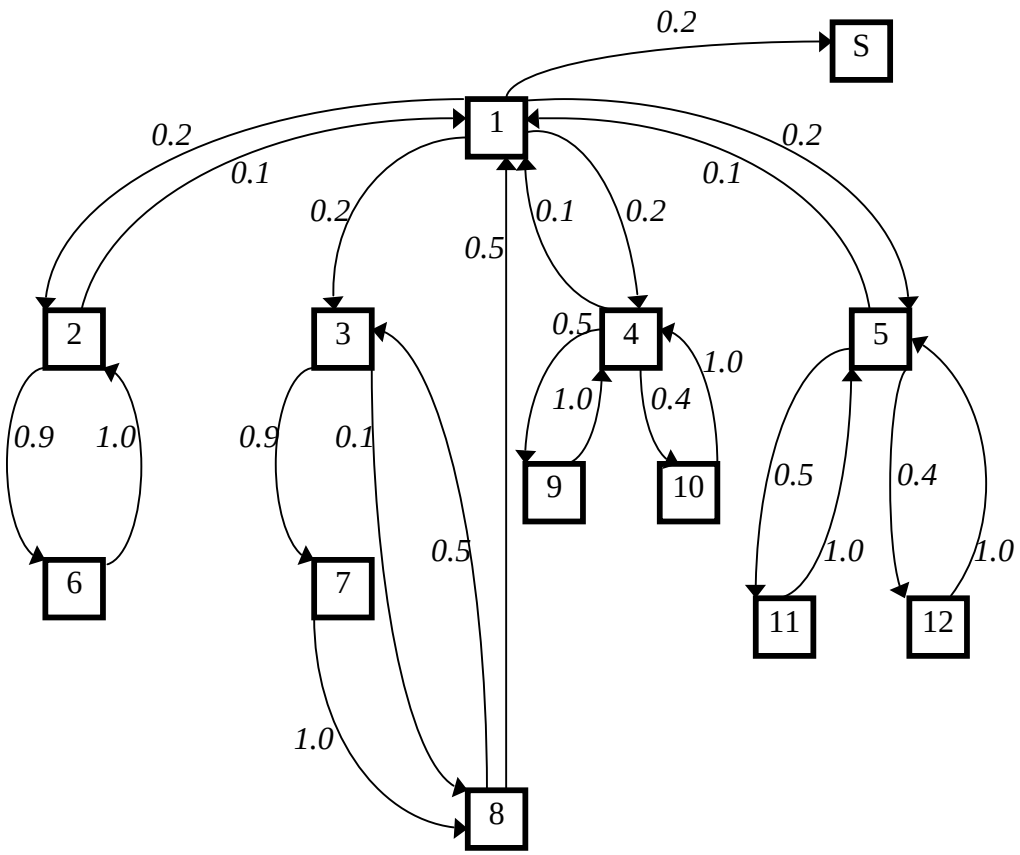
S
1
2
3
4
5
6
7
8
9
10
11
12
0.2
0.2
0.1
0.2
0.1
0.2
0.1
0.2
0.5
0.5
1.0
1.0
0.4
0.9
1.0
0.9
0.1
0.5
0.5
0.4
1.0
1.0
1.0

FIGURE 3

Biographical Sketches

Jayant Rajgopal holds a Ph.D. in Industrial and Management Engineering from the University of Iowa and is currently Associate Professor of Industrial Engineering at the University of Pittsburgh. His research interests are in applied mathematical programming, reliability and production & operations analysis. He is a senior member of IIE, and a member of INFORMS and ASEE. He is a registered professional engineer (IE) in the state of Pennsylvania.

Mainak Mazumdar holds a Ph.D. in Applied Statistics and Probability from Cornell University and is currently Professor of Industrial Engineering at the University of Pittsburgh. Prior to his current appointment he was employed as a research scientist in the Mathematics Department and in the Systems Sciences Department at the Westinghouse R&D Center for fifteen years. His research interests are in reliability and industrial statistics. He is a member of IIE, IEEE and ASA.

Subba Rao Majety is a doctoral student in the Industrial Engineering Department at the University of Pittsburgh. He has Master's degrees in Civil Engineering and Industrial Engineering from the University of Windsor. His current research is focused on the application of integer programming to reliability allocation. He is a student member of IIE and INFORMS.

Vikram Talada is a graduate student in the Industrial Engineering Department at the University of Pittsburgh. He has a Bachelor's degree in Mechanical Engineering from the Indian Institute of Technology, Madras (India). His research interests are in the application of information systems to manufacturing and optimization.