Artificial Systems for Software Engineering Studies

John H. Rowland*
University of Wyoming

Abstract
This paper describes techniques for the automatic generation of large artificial software systems which can be used for laboratory studies of testing and integration strategies, reliability models, and so forth. A prototype generator is described which produces code for such systems by constructing a large number of nearly identical modules. This generator has been used to construct a family of systems which in theory can be made arbitrarily large. Several experiments were conducted to explore the sensitivity of the Jelinski-Moranda model to violations of the assumption that all defects have equal probability of being discovered.

Index Terms-Artificial systems, software testing, software reliability, software engineering.

1 Introduction
A number of recent papers describe experiments or simulations designed to gain insight into software engineering issues. For example, Giammo [8] uses a simulation to test the effectiveness of a reliability model which relaxes the common failure rate assumption. Miller and Sofer [14] use Monte Carlo simulation of Poisson processes to explore the accuracy of certain least square completely monotone curves as estimators for the current failure rate of programs. Other experiments have involved studies of testing strategies. Duran and Ntafos [7] compare random strategies with partition strategies, White and Sahay [18] explore path strategies for testing predicates, while Howden [11] compares certain functional and structural techniques when applied to actual programs with known defects. Additional experiments and techniques for data collec-

*This work was carried out while the author was on a sabbatical leave at the Georgia Institute of Technology.
sible to perform analytical calculations to measure system attributes.

A prototype generator has been written to produce code for experiments which require the actual system rather than simulation or analytical computations. Incidentally, the code generator can also provide a means for checking on the validity of simulations or analytical computations. Let us first describe a sample system and then the architecture for software tools which can be used to expedite this type of experimentation. Some sample experiments involving reliability model assumptions will be described later.

2 Sample System

In order to demonstrate the feasibility of this approach, the prototype code generator was used to produce a simple class of artificial systems which in theory could be arbitrarily large. In practice, of course, the size is limited by machine space and time constraints. The sample systems consist of three module types: 1) a control module, 2) internal modules, and 3) leaf modules. The internal and leaf modules correspond to internal and leaf nodes, respectively, on the system hierarchy chart. The modules of a given type all have the same code except for a limited number of items. These items are specified on a script which gives the details of the particular system being generated.

The input to the control module consists of a control vector \( z \) and a computational vector \( y \). After reading \( z \) and \( y \) the control module executes a five-way case statement which calls another module with \( z \) and \( y \) as parameters. The variables which are contained in the control module script include: 1) the name of the control module, 2) the component of \( z \) which controls the five-way case statement, and 3) the list of up to five modules which can be called by the control module.

The \( i \)th internal module performs two matrix multiplications and assigns the results to \( z \) and \( y \):

\[
\begin{align*}
x & := A(i) \times y \\
y & := B(i) \times y
\end{align*}
\]

where \( A(i) \) and \( B(i) \) are four by four integer matrices. It then executes a five-way case statement similar to the one performed by the control module. In addition to the script parameters listed for the control module, there are two additional ones which determine \( A(i) \) and \( B(i) \) from a list of possible matrices. The vector \( z \) is reduced (mod 5) for ease in writing the case statement. This could also be done to \( y \) to eliminate the possibility of arithmetic overflow. The matrix multiplications are performed in integer arithmetic. The internal modules return control to the calling program after executing the case statement.

The leaf modules are similar to the internal modules except that they return to the calling program immediately after performing the matrix multiplications. The module name and two matrix indices are contained on the script record corresponding to a leaf module.

Properties of the Systems

Some properties possessed by the sample systems will be described here. The control vectors \( z \) determine the control flow of the system. The operations on the computational vectors \( y \) are independent of those on the control vectors except for the control flow. For this reason, once the control path has been determined, the output values for the \( y \) vectors are uniquely determined by their values on a basis. Thus, four linearly independent samples of \( y \) along a path are a sufficient test for the computational matrices.

Termination can be guaranteed for the systems by requiring the modules with the highest indices to be leaves and requiring that each interior module only call modules with a higher index.

A matrix is said to be non-singular mod (5) if it has an inverse within the system of arithmetic mod(5). If the vectors \( y \) are reduced mod (5) after each matrix multiplication and if all of the computational matrices are non-singular mod (5), then for a given control path the problem of computing the \( y \) input to produce a given \( y \) output has a solution which can be computed using a modified form of Gaussian elimination on the appropriate system of equations. This modification avoids division by using multiplication by inverses mod (5) in places where division would ordinarily occur [16].

A given path in the system might or might not be feasible. An algorithm for determining the feasibility of a given path can again be based on Gaussian elimination. Let \( 0, i_1, \ldots, i_m \) be a path through the system and let \( A(0), A(i_1), \ldots, A(i_m) \) be the corresponding control matrices. Let \( j_0, j_1, \ldots, j_m-1 \) be the coordinates of the control vector \( z \) used to determine calls from the corresponding modules. Let \( P(k) \) be the partial matrix product

\[
A(i_k) \times \ldots \times A(i_1) \times A(0)
\]
where \( A(0) = I \), the identity. Then the feasibility of the path is determined from the feasibility of the system of equations

\[
Cz = b
\]

where for \( k = 0, 1, \ldots, m - 1 \) the \( k \)-th row of \( C \) taken to be the \( j_k \)-th row of \( P(k) \) and \( b_k \) is the index required to have the case statement select module \( i_{k+1} \). The modified form of Gaussian elimination mentioned above can be used to determine the feasibility of the given path.

3 Architecture

The software components which are useful for conducting experiments include a code generator, an oracle, a defect seeder, and a simulator. These components will now be described.

The Code Generator

The artificial system will be constructed by replicating a limited number of standard modules which differ only by variations in certain parameters associated with each module type. The input to the code generator consists of a set of templates and a script. There is a template for each module type which consists of the code for the module type with special symbols marking positions to be filled with parameters which can vary from module to module. The script contains the symbols to be inserted at these positions. The generator produces code by carrying out the following steps:

1. Scan a template and record the positions of the parameters to be inserted from the script.
2. Read the parameter values from the script which correspond to one module and insert these values in the template.
3. Output the code for the current module.
4. Repeat 2 and 3 for all modules of the type represented by the given template.
5. Repeat 1 - 4 for each template type.

For example, the systems described above contain three types of modules: A control module, internal modules, and leaves. The generator produces first the control module, then a number of internal modules, and finally the leaves.

Note that the generator is independent of the artificial system and the language in which the code is written. Of course the templates are language dependent, but the scripts might or might not be language dependent, depending on the meaning assigned to the parameters.

The Oracle

The oracle has the capability of determining the system output for any given input. An oracle can be constructed for a family of systems if the functions performed by the systems are sufficiently simple. The basic approach is through composition of the functions represented by the individual modules. For example, suppose \( z \) is an input vector which causes modules 1, 2, \ldots, \( k \) to be invoked and that this causes transformations of the form

\[
x := f_i(z), \quad i = 1, 2, \ldots, k
\]

Then the value of \( z \) after execution would be

\[
f_k \circ \cdots \circ f_2 \circ f_1(z)
\]

where \( \circ \) represents the composition of functions. One of the keys to this approach is to keep the functions \( f_i \) simple; otherwise, the task of constructing an oracle would be as large as constructing a large real system. The oracle for the sample systems determines the control path and then computes a succession of matrix products.

A separate oracle must be written for each family of systems. The oracle is not essential for all experiments but in many situations speeds up the process and serves as a check on the system generator and other features of the experiment.

Defect seeder

The defect seeder has the capability of inserting or deleting defects in the script for the artificial system. This can be done according to specified probability distributions or by following a defect script.

Simulator

The simulator makes use of the oracle to produce system output directly from the script. This avoids the frequent compilation of system code which would otherwise be required every time a defect is seeded. The exact nature of the experiment determines whether or not a simulator is appropriate.
4 Sample Experiments

We will describe some experiments in which artificial systems were used to study the relationship between the hazard rate and the number of defects in a system. The hazard rate is defined in terms of the system's reliability. Let $T$ be a random variable representing the time to first failure of the system. The reliability, $R(t)$, is defined to be the probability that the system will not fail in the time interval $[0,t]$. The hazard rate $z$ is then defined by

$$z(t) = f(t) / R(t),$$

where $f$ is the probability density function for $T$. The probabilistic interpretation is that for a small time increment $\Delta t$, $z(t)\Delta t$ represents the conditional probability that the system will fail in the interval $(t, t + \Delta t)$ given that it has not failed in $[0,t]$. Further discussion of this concept is given in [2], [13], [9], and [10].

The hazard rate plays a central role in the formulation of several software reliability models [9] [10]. These models are based on assumptions about the general form of the relationship between the number of defects in a system and the hazard rate. The parameters of this relationship, including the number of defects, are then estimated from data obtained during tests of the system. The experimental results show that several factors other than the number of defects can drastically affect the shape of the hazard curve.

In the Jelinski-Moranda reliability model [5] [9] the hazard is assumed to be constant during the interval of time between the removal of one defect and the discovery of the next. Under this assumption $T$ has an exponential distribution and the hazard rate is the reciprocal of the mean time to failure (MTTF). The Jelinski-Moranda model also assumes that the hazard rate is proportional to the number of errors remaining in the system. The mean time to failure was measured for a number of different configurations of artificial systems and testing strategies. We will now describe those experiments.

Experiment 1. Non-equally-likely defects. The assumption is made in the Jelinski-Moranda model that each defect in the system is equally likely to cause a failure during testing. This experiment determines the shape of the hazard curve under conditions where this assumption is invalidated by having calls to some modules be more probable than others. The system whose call graph is shown in Figure 1 was used for this experiment.

![Call Graph for System 1](image)

Figure 1. Call Graph for System 1.

The coordinates of $x$ and $y$ were chosen from a uniform distribution on the integers $0, 1, 2, 3, 4$. The system was constructed so that one fifth of the calls from the control module were to module 1 and four-fifths of the calls were to module 2. Similarly, one-fifth of the calls from module 2 were to module 8 and four-fifths were to module 9. The remainder of the calls were uniformly distributed. A defect was seeded in each module by choosing the wrong computational matrix from a list of five matrices. The first coordinate of $x$ controlled the calls from the control module, the second coordinate controlled the calls from modules 1 and 2, and the third coordinate controlled the calls from modules 3 and 4. The control matrices were all taken to be the identity matrix; hence $x$ was not changed during the computations.

The relationship between the hazard rate $z$ and the number of defects was determined both analytically and experimentally for two different strategies of defect removal. In the first strategy, which we will refer to as the $LR$ strategy, the module defects were removed from left to right; that is, in the order $1,3,10,11,12,13,14,4,15,16,17,18,19,5,6,7,2,8,9$. For the second $RL$ strategy the defects were removed in the right to left order: $2,9,8,1,7,6,5,4,19,18,17,16,15,3,14,13,12,11,10$. The experimental values were determined by simulating the system with uniform random input and measuring the time to first failure. The time to first failure was taken to be the number of tests required to produce the first incorrect output vector. These times were averaged over a relatively large number ($n$) of trials to obtain an estimate for the mean time to failure which is the reciprocal of $z$. The value of $n = 500$ was used except for the smaller hazard values where $n$ was lowered to avoid excessive simulation times. For these runs the value $n = 100$ was used for
$0.04 \leq z \leq 0.08$ and $n = 30$ was used for $z < 0.04$. The analytical computations will be described later.

The continuous curve in the upper portion of Figure 2 represents the analytical hazard rate as a function of the number of defects removed under the left to right (LR) strategy. The individual points represent the experimentally measured values. The lower curve and points give the same information for the right to left (RL) strategy.

![Figure 2. Hazard Rate for Experiment 1.](image)

Table I compares the actual number of defects in the system to estimates obtained from the Jelinski-Moranda model. These estimates were computed using the following procedure. Let $N$ be the total number of errors in the system and let $t_i$ be the time at which the $i$th defect is removed. According to this model the hazard rate $z$ is given by

$$z(t) = \phi \cdot |N - (i - 1)|, \quad t_{i-1} \leq t < t_i$$

where $\phi$ is a constant of proportionality. Let $\alpha = N \cdot \phi$ and $\beta = -\phi$. The defect estimates were obtained by fitting a least squares line to the measured values of $z(t_0), z(t_1), \ldots, z(t_{i-1})$ to determine $\alpha$ and $\beta$ and then setting $N = -\alpha / \beta$. The maximum likelihood estimates for $N$ described in Goel [10] were also computed using the measured mttr's as the time between failures. The latter values were in reasonable agreement with the least squares estimates and are not shown here.

<table>
<thead>
<tr>
<th>Defects Removed</th>
<th>LR Strategy</th>
<th>RL Strategy</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-499</td>
<td>19</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-20585500</td>
<td>3</td>
<td>19</td>
</tr>
<tr>
<td>3</td>
<td>390</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>4</td>
<td>222</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>203</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>201</td>
<td>7</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>151</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>134</td>
<td>11</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>71</td>
<td>14</td>
<td>19</td>
</tr>
</tbody>
</table>

Table I. Jelinski-Moranda Estimates for System 1.

Experiment 2. Top-down vs. Bottom-up Defect Removal. This experiment demonstrates that the order of defect removal can have a significant effect on the shape of the hazard curve in hierarchical systems. The experiment was performed by simulating a system containing 780 modules in addition to the control module. There were five modules on the first level, 25 modules on the second, 125 modules on the third, and 625 leaf modules on the fourth level. Each module except for the leaves could call five modules on the next level and each module except the control could be called by one and only one other module.

The single control matrix was again taken to be the identity so that $z$ was unchanged by the system. The first coordinate of $z$ was used to determine calls from the control module, the second coordinate for level 1, the third coordinate for level 2, and the fourth coordinate for level 3 modules. The coordinates of $z$ and $y$ were again chosen uniformly from the integers 0, 1, 2, 3, 4. There is a unique path to each leaf of the system and all paths are equally probable.

There were five non-singular computational matrices $B(1), B(2), \ldots, B(5)$. The matrix $B(i)$ was assigned to module $j$ if $i$ was congruent to $j \mod (5)$. Errors were seeded into modules whose index was congruent to 1 mod (5) by assigning them $B(2)$ rather than $B(1)$. The top solid curve in Figure 3 shows the theoretical hazard as a function of the number of defects removed for a bottom-up defect removal strategy (776,771,766,etc.), while the lower solid curve shows the hazard for a top-down strategy (1,6,11,etc.). As before, the measured values are shown with asterisks. The value $n = 500$ was used except for $0.04 < z < 0.1$ where $n$ was taken to be 100 and $z < 0.04$ where $n = 10$ was used. Table II shows selected Jelinski-Moranda least-squares estimates of the number of defects in the system compared to the actual number.
Defects Removed Experiment 2

<table>
<thead>
<tr>
<th>Defects Removed</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Least Squares</td>
<td>Actual</td>
</tr>
<tr>
<td>Top-down</td>
<td>Bottom-up</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>50</td>
<td>63</td>
</tr>
<tr>
<td>100</td>
<td>92</td>
</tr>
<tr>
<td>125</td>
<td>112</td>
</tr>
<tr>
<td>150</td>
<td>127</td>
</tr>
<tr>
<td>155</td>
<td>134</td>
</tr>
</tbody>
</table>

Table II. J-M Estimates for Systems 2, 3.

Experiment 3. Coincidental Correctness. This experiment was used to investigate the effect of coincidental correctness on the Jelinski-Moranda estimates. A test of a system is said to produce coincidental correctness when it causes execution of defective code, but produces the correct output anyway. The call graph was identical to the one used in Experiment 2. There were six computational matrices $B(1), B(2), \ldots, B(6)$. The matrices $B(2), B(3), B(4)$ had the value one in the first, second, and third positions of the main diagonal, respectively, and zeros in all other positions. The matrix $B(6)$ was identical to $B(1)$ except for one element. The non-defective system had matrix $B(i)$ assigned to module $j$ whenever $i$ was congruent to $j$ mod (5). Defects were seeded into modules whose index was congruent to 1 mod (5) by assigning $B(6)$ to modules 1, 11, 21, etc. and $B(2)$ to modules 6, 16, 26, etc. The effect of the low rank of $B(2), B(3), B(4),$ and $B(6) - B(1)$ was to produce a considerable amount of coincidental correctness in the tests. The results of this experiment are summarized in Figure 3 and Table II. The diamonds (•) in Figure 3 represents the measured values of the hazard rate for bottom-up defect removal while the circles (o) represent measured values for top-down defect removal.

5 Analysis of Experimental Results

Sample Size Considerations

The mean time to failure was obtained experimentally for all three of the systems described above. It was also computed analytically for systems 1 and 2. The experimental values were obtained by taking the average of a large number of time to failure measurements for each defect configuration. Confidence intervals for these averages will now be determined.

Let $T$ again represent the time to first failure of a system. This will be measured in terms of the number of trials required to discover a defect with random uniform input vectors. Let $q$ be the probability of uncovering a defect on a single trial, $p = 1 - q$, and $P$ be the appropriate probability measure. Then $T$ follows a geometric distribution [15]; that is,

$$P\{T = j\} = p^{j-1}q, \quad j = 1, 2, \ldots$$

The expected value and standard deviation of $T$ are given by $\mu = 1/q$ and $\sigma = \sqrt{p/q}$, respectively.

Let $\bar{T}$ be the average of $n$ independent time to failure measurements. According to the central limit theorem [15], the random variable

$$\frac{T - \mu}{\sigma / \sqrt{n}}$$

is approximately standard normal.
has an approximate normal distribution with zero mean and unit variance. Let $t_\alpha$ be chosen so that the probability of a normal random variable with mean zero and unit variance being in the interval $[-t_\alpha, t_\alpha]$ is $1 - \alpha$. Then

$$P\left(\sqrt{n} \left| T - \mu \right| / \sigma \leq t_\alpha\right) = 1 - \alpha.$$  

It follows that the $1 - \alpha$ confidence interval for the relative error $qT - 1$ is $[-\epsilon, \epsilon]$, where

$$\epsilon = t_\alpha \sqrt{p/n}.$$  

This can also be used as an approximate $1 - \alpha$ confidence interval for the relative error in $1 / T$ as an estimator of $q$ since $qT \approx 1$. With $n = 500$ and $\alpha = 0.05$ this interval ranges from roughly 1% at $p = 0.01$ to 9% at $p = 1.0$. In Experiment 1 the value $n = 100$ was used for $0.04 \leq q \leq 0.08$ and $n = 30$ for $q < 0.04$. This leads to confidence intervals of 19% and 36%, respectively. For the small values of $q$ the absolute error has confidence intervals of width 0.03 in both cases. The confidence interval corresponding to $n = 10$ (used in Experiment 2 for $q < 0.01$) is 62% for the relative error and 0.012 wide for the absolute error.

Comparison with Analytical Calculations

The results of the experiments 1 and 2 have been compared to predicted values which were obtained analytically. The analytic values of the hazard rate were computed using basic combinatorial probability formulas under the assumption that there would be no coincidental correctness. Each leaf in the artificial system has a unique path in the call graph connecting it to the control module. Let $\pi_1, \pi_2, \ldots, \pi_k$ represent these paths and let $D_1, D_2, \ldots, D_k$ represent the corresponding sets of control vectors $z$ which will drive those paths. Following Duran and Ntafos [7] we will let $p_i$ be the probability that $z \in D_i$, and let $\theta_i$ be the conditional probability that a defect is detected given that $z \in D_i$. The overall probability $q$ of detecting a defect is given by

$$q = \sum_{i=1}^{k} p_i \cdot \theta_i.$$  

The assumption of no coincidental correctness would imply that for each path $\theta_i$ is either 0 or 1. In this case $q$ is the sum of the $p_i$'s over the set of indices corresponding to paths which contain defects. (Actually, coincidental correctness does occur when all of the coordinates of $y$ are zero. This event occurs with probability $1/625 = 0.0016$.)

Experiment 1. The system shown in Figure 1 has 15 leaf modules: namely modules 5 through 19. For convenience, we will use $p_5, \ldots, p_{19}$ to represent the probabilities that $z$ will drive modules 5 through 19, respectively. The index $i$ in (1) will then run from 5 to 19 rather than from 1 to $k$. In order to compute $p_i$, for example, note that the corresponding control path $\pi_i$ is through modules 0,1,5, where 0 represents the control module. Now module 1 is called when $x_1 = 0$ and module 5 is called from module 1 when $x_2 = 2$. Thus $p_5$ is the probability that $x_1 = 0$ and $x_2 = 2$. Since the coordinates of $z$ are chosen independently, we can multiply probabilities to obtain $p_5 = 0.2 \cdot 0.2 = 0.04$. Similarly we obtain $p_9 = p_7 = 0.04$. The paths $\pi_8$ and $\pi_9$ are 0,2,8 and 0,2,9, respectively. Now module 2 is called when $x_1 > 0$ and module 8 is called from module 2 when $x_2 = 0$; so $p_8 = 0.8 \cdot 0.2 = 0.16$. Similarly, $p_9 = 0.8 \cdot 0.8 = 0.64$. The other path probabilities are given by $p_{10} = p_{11} = \cdots = p_{19} = 0.2^9 = 0.008$.

Let $q_i$ represent the probability of detecting a defect after $k$ defects have been removed from the system. Initially there was one defect in each module; so $q_i = 1$ for $i = 5, \ldots, 19$, and $q_0 = 1$. Consider first the left to right (LR) order of defect removal. The removal of defects from modules 1 and 3 does not yet affect $q$ because every leaf module still has a defect. Thus $q_1 = q_2 = 1$. When the defect is removed from module 10, $\theta_{10}$ becomes 0 and $q_3 = 1 - p_{10} = 0.992$. The remaining points on the graph shown in Figure 2 can be calculated in a similar fashion.

When defects are removed in the RL order we obtain $q_0 = q_1 = 1$, and $q_2 = 1 - p_9 = 0.36$, and so forth.

The measured values for Experiment 1 were all within 2.4% of the theoretical values for $q > 0.5$, 7.3% for $q > 0.08$, and 21% for $q < 0.08$. The absolute error never exceeded 0.012 for $q < 0.08$.

The theoretical values for the top-down strategy in Experiment 2 were computed by considering the probability of detecting errors at each level. For example, when no defects have been removed, the probability of detecting a defect at level 1 is 1/5. The probability of first detecting a defect at level 2 is the product of the probabilities of not detecting a defect at level 1 (4/5) and of detecting a defect at
level 2 \((1/5)\). Continuing in this fashion we obtain
\[ q_0 = 0.2(1 + 0.8 + (0.8)^2 + (0.8)^3) = 0.5904 . \]

For the top-down strategy the first defect removed is from module 1. Hence
\[ q_1 = 0.2(1 + 0.8 + (0.8)^2) = 0.488 . \]

The values for the bottom-up strategy were obtained by counting the number of paths in the call graph which contain defects. Initially, 369 out of 625 paths contained defective modules. After removing the defect in module 176 there would be 368 defective paths. Hence \( q_1 = 368/625 = 0.5888 \). The hazard decreases by \( 1/625 \) for each level 4 defect removed until module 756 is processed. Removing the defect in this module does not yet lower the hazard because calls to it come from module 151 which is still defective. The remaining values were computed by continuing this type of analysis.

The measured values for Experiment 2 were all within 7.2% of the theoretical values for \( q > 0.04 \), and 53% for \( q < 0.04 \). The absolute error never exceeded 0.01 for \( q < 0.04 \).

Theoretical hazard values were not computed for Experiment 3 due to difficulties in quantifying the effect of the coincidental correctness.

6 Conclusions

This work has demonstrated the feasibility of constructing large artificial systems with known defects. The generator, which is independent of the language used for coding the artificial system, can automatically produce the code from templates and scripts. The scripts for the larger systems were generated from formulas; so the process is not necessarily limited by the size of script that can be created manually. The oracle and simulators, on the other hand, are very specialised and can only be applied to the family of systems used here.

A number of authors have expressed reservations about the assumptions underlying various software reliability models [12] [4, p.280]. We have illustrated here several situations where the Jelinski-Moranda model is very sensitive to violations of the assumption that all defects are equally likely to be executed. According to this model, the graph of the hazard rate vs. defects removed should be a straight line [9] [10]. The curves in Figures 2 and 3 are obviously not straight lines. Perhaps even more significant is the fact that in the absence of any control over the order of defect removal, the hazard rate is not a well-defined function of the number of defects removed. Thus, when applied to systems which resemble those used in the experiments, any of the reliability models which attempt to fit curves to this relationship would appear to be based on a questionable foundation.

Experiment 1 shows that the order of defect removal can have a significant effect on the shape of the hazard curve when portions of the code have a higher probability of being executed than others. In view of the generally accepted view that 90% of the time is spent in 10% of the code [1], it would appear prudent to investigate these probabilities empirically before applying the model to predict the total number of defects.

Experiment 2 shows that equally probable paths are not sufficient to insure the applicability of the Jelinski-Moranda model to hierarchical systems. In this case the higher-level modules tend to be exercised more heavily than the lower-level modules; so top-down defect removal produces a concave upward hazard curve while a bottom-up strategy produces a concave downward curve. The effect of this on the defect estimates was quite severe for our sample system. This would suggest that there is considerable danger in the application of the Jelinski-Moranda model to hierarchical systems. Incidentally, Downs [6] and Bendell and Mellor [4] also report that a concave upward hazard curve is commonly observed in large systems.

Experiment 3 demonstrates that coincidental correctness might introduce another element of confusion into the estimates. In particular, the estimates for the bottom-up strategy bear little resemblance to those obtained in Experiment 2, even though the same modules are defective in both of these experiments.

Further experiments on other reliability models would be instructive. In particular, it would be of interest to construct systems which do satisfy the assumptions of the models as well as more which do not.

7 Acknowledgements

The author would like to thank Dr. Richard A. DeMillo and his staff at the Software Engineering Research Laboratory of the Georgia Institute of Technology for many helpful discussions. He is indebted to Georgia Tech for providing library and computing facilities for this project.
References


