JUMBL: A Tool for Model-Based Statistical Testing

S. J. Prowell
The University of Tennessee
E-mail: sprowell@cs.utk.edu

Abstract

Statistical testing of software based on a usage model is a cost-effective and efficient means to make inferences about software quality. In order to apply this method, a usage model is developed and analyzed to validate its fitness for use in testing. The model may then be used to generate test cases representing expected usage, and to reason about system reliability given the performance on the set of tests. The J Usage Model Builder Library (JUMBL) is a Java class library and set of command-line tools for working with usage models. The JUMBL supports construction and analysis of models, generation of test cases, automated execution of tests, and analysis of testing results.

1. Motivation

Statistical testing of software based on a usage model is a cost-effective and efficient means to make inferences about software quality. To apply this method, a model of system use (a *usage model*) is developed and analyzed to validate its fitness for use in testing. A usage model is a specification of the system's *use*, not its *behavior*. In other words, the model represents what the user is likely to do next, not the internal states of the system. As such it is often possible to construct relatively simple usage models even for highly complicated systems. These models represent a population of uses from which a sample is drawn. The sample defines a set of test cases. When executed, the results of the test cases allow reasoning about the entire population. Thus testing is explicitly treated as a statistical experiment. A large case study is presented in [11].

The usage models considered here are finite-state, time homogeneous Markov chains [5, 14], which can be thought of as deterministic finite automata with probabilistic transitions. An example model is shown in fig. 1. This model describes the operation of a simple telephone. A "use" (or test) of the phone is any path from the state On Hook, called the *source*, to the state Exit, called the *sink*. One use of the phone according to this model is "incoming call," "lift receiver," and "hang up." The model may be analyzed, used to generate tests based on the probabilities or structure, and used for population inference.

The process of statistical testing based on a usage model can be divided into five phases:

1. Usage Modeling, in which one or more usage models are constructed to represent the population of uses.
2. Model Analysis and Validation, in which usage models are analyzed to determine their properties, which are then compared against known or assumed properties of field use and test constraints.
3. Test Planning, in which test cases are generated and test automation is planned and designed.
4. Test Execution, in which the generated tests are executed against the system under test and the results are recorded.
5. Product and Process Measurement, in which the results of the test are analyzed to determine expected reliability.

Throughout this process there are opportunities for automation. For this reason, the J Usage Model Builder Library (JUMBL) has been developed by the Software Quality Research Laboratory of the University of Tennessee. The JUMBL is a Java class library which supports all phases of statistical testing and which provides a command line interface for working with Markov chain usage models.

Two factors constrained the development of the JUMBL:

- Researchers needed a flexible platform for exploring new computations and methods related to Markov chain usage models.
- The lab's industrial partners needed a toolkit for industrial application of the methods being developed in the lab.

To support the first goal, a flexible object-oriented design was chosen for the JUMBL. The Java programming language was chosen primarily because of its multi-platform

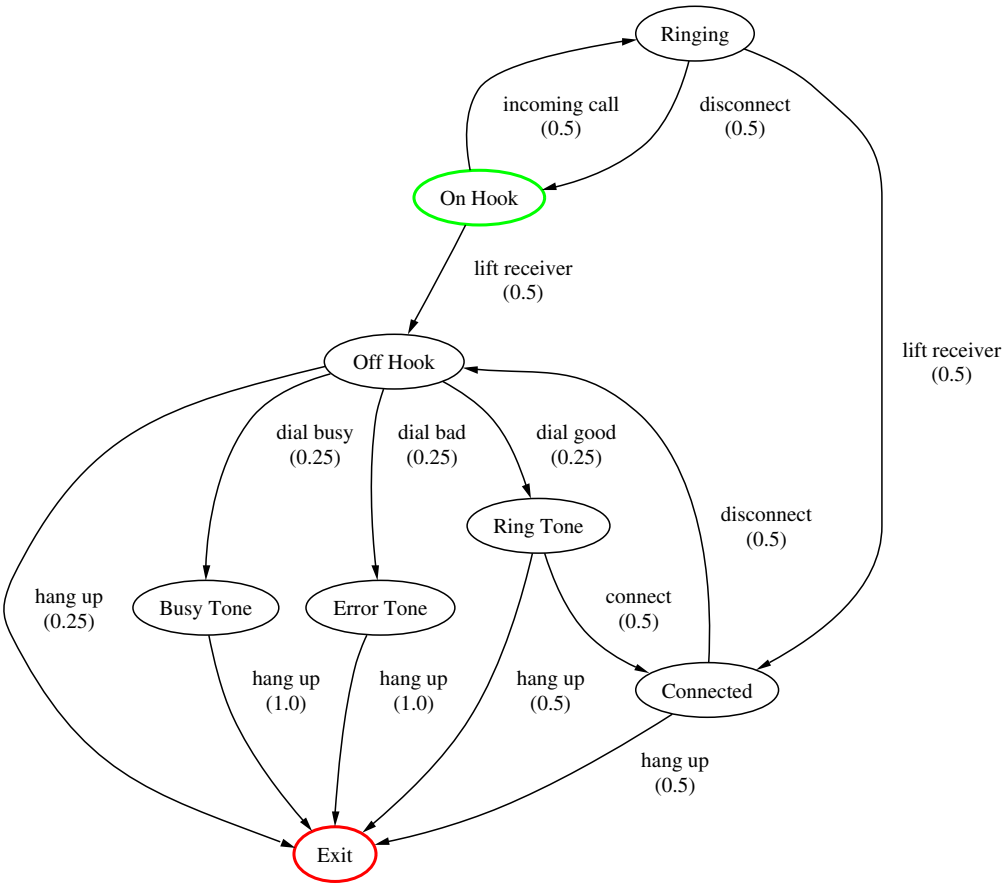

Figure 1. Telephone Model

capability, as the expected user base included Windows, Linux, and UNIX machines. To support the second goal, core features of each public version of the JUMBL are identified and designed under careful peer review.

The remaining sections of this paper discuss the support the JUMBL currently offers for each phase of statistical testing.

2. Model Construction

Markov chain usage models consist of states representing states of use, and arcs labeled with usage events also called *stimuli*. A probability distribution is associated with the outgoing arcs from each state. Thus one must specify both the structure of the usage model and its probability distribution. The JUMBL does not provide a usage model editor, but instead allows users to develop their usage models in a variety of formats and allows users to convert among the different representations. Users may develop their usage models in a spreadsheet, in a text editor, or in graphical editors.

The most common format for developing usage models is The Model Language (TML), described in [10], which is a simple language developed specifically for specifying Markov chain usage models. The TML representation of the phone model from fig. 1 is shown in fig. 2. The TML language supports developing usage models as compositions of other models to simplify specifying the structure of large models. For example one might create a model of each user dialog presented by a software system and then create a top-level model which links these together. TML allows specifying the probabilities as constraints along with simple objective functions to simplify specifying the probability distribution for a usage model. For example one might specify that one arc is twice as likely as another, and then leave all other arcs unconstrained. The JUMBL then chooses probabilities which honor the given constraints. Additionally, multiple distributions (as constraints) can be stored in a single model, and automated testing information (called *labels*) can be attached to parts of the model as necessary.

In addition to TML standard formats supported by the JUMBL include comma-separated value (CSV), the DOT language used by the Graphviz tools [2], GML as used by Graphlet [4], and Model Markup Language (MML). MML is an XML extension language created specifically for representing Markov chain usage models.

3. Model Analysis

The analysis of a usage model reveals the relative time spent in each state, the number of times a particular stimulus will occur in testing, and how long it will take to run a test.

```
// Choose probabilities such that
// state entropy is maximized.
($ emax $)
model phone
    source [On Hook]
         "incoming call" [Ringing]
         "lift receiver" [Off Hook]
     [Busy Tone]
         "hang up"       [Exit]
     [Connected]
         "disconnect"    [Off Hook]
         "hang up"       [Exit]
     [Error Tone]
         "hang up"       [Exit]
     [Off Hook]
         "dial bad"      [Error Tone]
         "dial busy"     [Busy Tone]
         "dial good"     [Ring Tone]
         "hang up"       [Exit]
     [Ring Tone]
         "connect"       [Connected]
         "hang up"       [Exit]
     [Ringing]
         "disconnect"    [On Hook]
         "lift receiver" [Connected]
end
```

Figure 2. Phone Model as TML

The JUMBL creates a comprehensive analytical report of model and use statistics which can be used to validate model correctness, plan for testing, investigate expected use, and compare different modeling approaches.

Model Statistics

Node Count	8 nodes
Arc Count	14 arcs
Stimulus Count	8 stimuli
Expected Test Case Length	4 events
Test Case Length Variance	3.111 events
Transition Matrix Density (Nonzeros)	0.234375 (15 nonzeros)
Undirected Graph Cyclomatic Number	8

Figure 3. Model Statistics

The analytical report produced by the JUMBL is divided into four sections.

- The model statistics section, shown in fig. 3, presents some overall statistics about the model, including its size and the expected length of a test case.
- The node statistics section, shown in fig. 4 gives statistics for each node. These include the fraction of time one spends in the node (out of all nodes) for a large

Node Statistics

Node	Occupancy	Probability of Occurrence	Mean Occurrence / Variance (visits per case)		Mean First Passage / Variance (events)	
[On Hook]	0.266666667	1	1.333	0.444444444	3.75	2.021
[Error Tone]	44.4444444E-3	0.222222222	0.222222222	0.172839506	20.5	395.75
[Ringing]	0.133333333	0.5	0.666666667	0.666666667	5	32.933
[Ring Tone]	44.4444444E-3	0.208333333	0.222222222	0.202469136	21	418
[Connected]	88.8888889E-3	0.416666667	0.444444444	0.30617284	9	69.2
[Off Hook]	0.177777778	0.833333333	0.888888889	0.217283951	3	8
[Busy Tone]	44.4444444E-3	0.222222222	0.222222222	0.172839506	20.5	395.75
[Exit]	0.2	1	1	0	4	3.111

Figure 4. Node Statistics

number of tests, the probability that the node is visited in a randomly-selected test case, and the expected number of times the node will be visited in a test case.

- The stimulus statistics section, shown in fig. 5 gives statistics for each stimulus. These include the fraction of time one spends in the stimulus (out of all stimuli) for a large number of tests, and the expected number of times the stimulus will occur in a test case.
- The arc statistics section, shown in fig. 6, reproduces the entire model and shows the probabilities chosen by the JUMBL for each arc, the fraction of time spent in the arc (out of all arcs) for a large number of tests, the probability that the arc is visited in a randomly-selected test case, and the expected number of times the arc will be visited in a test case.

From the analysis of the phone model we can expect only 42% of the uses generated to result in a connected phone call (this is the probability that state Connected occurs). To increase this, we would increase the relative weight of arcs leading to Connected, and decrease the relative weight of arcs leading to other states. Thus setting arc probabilities becomes an iterative process of proposing, evaluating, and revising constraints. This process is described in detail in [9].

4. Test Generation

The JUMBL can generate test cases in five ways:

- A collection of test cases can be generated which cover the model with the minimum cost.
- Test cases can be generated by random sampling with replacement.
- Test cases can be generated in order by probability.
- There is an interactive test case editor for creating test cases by hand.
- Test cases can be created by interleaving the events of other test cases.

Test cases generated from the model must match the functionality of the system; if the test case says to select a particular menu item, the item must be available at that time. One can gain confidence that the model and delivered system match one another by running tests which visit every arc of the usage model. Such a test can be constructed by the JUMBL. The JUMBL uses the Chinese postman algorithm [3] to construct a collection of test cases which cover all arcs of the model. A nonnegative cost is associated with each arc of the model. The algorithm minimizes the sum of these costs and thus the length of the resulting test cases. Running this minimum-cost coverage set gives a measure of assurance that the usage model and the system as delivered match; as a result, the minimum-cost coverage set is typically run before running any other tests generated from the model.

Random test cases are created by randomly choosing outgoing arcs in accordance with a selected probability distribution on the model. For example, if arc x is twice as likely as arc y, then the selection algorithm will tend to select arc x twice as often as arc y. Most models contain loops, and therefore represent an infinite population of tests from which random tests may be drawn.

Test cases can instead be drawn in order by their probability of occurrence. The test case with the greatest probability is generated first, the next-most-likely is generated next, etc. This allows testing those cases which are expected to occur most often in field use.

Stimulus Statistics

Stimulus	Occupancy	Mean Occurrence (visits per case)
hang up	0.25	1
disconnect	0.138888889	0.555555556
lift receiver	0.25	1
connect	27.7777778E-3	0.111111111
incoming call	0.166666667	0.666666667
dial busy	55.5555556E-3	0.222222222
dial good	55.5555556E-3	0.222222222
dial bad	55.5555556E-3	0.222222222

Figure 5. Stimulus Statistics

Arc Statistics

Arc	Probability	Occupancy	Probability of Occurrence	Mean Occurrence / Variance (visits per case)	
[On Hook]					
"incoming call"	0.5	0.166666667	0.5	0.666666667	0.222222222
"lift receiver"	0.5	0.166666667	0.666666667	0.666666667	0.222222222
[Error Tone]					
"hang up"	1	55.5555556E-3	0.222222222	0.222222222	0.172839506
[Ringing]					
"disconnect"	0.5	83.3333333E-3	0.25	0.333333333	0.333333333
"lift receiver"	0.5	83.3333333E-3	0.333333333	0.333333333	0.333333333

...

Figure 6. Arc Statistics

Figure 7. Model Editor

The JUMBL also provides an interactive test case editor, shown in fig. 7. This editor allows creating new test cases from models, or editing existing test cases. Users can create a test case which visits some desired part of the model, then let the editor finish the test case by randomly selecting events.

It is often useful to multiplex test cases. For example, a network switch may take input from several sources. Each kind of source can be modeled, and test cases generated. These test cases can be randomly multiplexed to simulate incoming data from several different streams. Alternately, some user interfaces consist of multiple modeless dialogs. Writing a single usage model for such a system would require a very large model. Instead, one model can be written for each modeless dialog, and generated test cases multiplexed to create a single test case which exercises the dialogs.

As with usage models, the JUMBL supports multiple test case formats. Test Case Markup Language (TCML) is an XML extension allowing users to create test cases directly, or to import from or export to other tools.

5. Test Execution

Since there are many different execution platforms, there are consequently many different test environments. Each test environment poses unique challenges which test automation must address. The JUMBL therefore takes a "least common denominator" approach to test automation.

Arbitrary information, such as commands for an automated test harness, C source code, shell script commands, or even written instructions to human testers can be attached to the model, its states, and its arcs. By writing this information in the sequence defined in the test case, a test case can be transformed into an executable script and run against the system under test. This arbitrary information is called a "label." The model of fig. 2 with C source code labels as fig. 8. Using these labels, a test case can be converted into a compilable and executable program.

The JUMBL's label system was chosen because it is the most general means for supporting automated testing. A common approach among those using the JUMBL is to develop a library of functions, with one function for each usage event. The function executes the event (sends data, clicks a button, etc.) and then records the system's response to the event.

Each test event must be recorded as either a success (the system performs as required) or a failure (the system does not perform as required). This is known as the *oracle* problem. There are several approaches to this problem:

- Limit testing to known cases. The response can then be checked against the known correct response.

```
// A sin-
gle use of the phone is all events from
// lifting the receiver until the re-
ceiver is
// replaced.
($ emax $)
model phone
a:|$ #include "phonetest.h"
  |$ int main() {
  |$     // Temporary phone number.
  |$     int n = 0;
  |$
  |$     // Initialize the test.
  |$     initialize_test();
source [On Hook]
    "lift receiver"
    a:|$     // Lift the receiver.
      |$     lift($t);
    [Off Hook]
    "incoming call"
    a:|$     // Call this phone.
      |$     call($t);
    [Ringing]
[Off Hook]
    "hang up"
    a:|$     // Hang this phone up.
      |$     hangup($t);
    [Exit]
    "dial busy"
    a:|$     // Lo-
cate a busy phone and call it.
      |$     n=findbusy();
      |$     dial($t, n);
    [Busy Tone]
    "dial good"
    a:|$     // Lo-
cate a good phone and call it.
      |$     n=findgood();
      |$     dial($t, n);
    [Ring Tone]
...
[Exit]
a:|$     Finished with the test.
  |$     close_test();
  |$ }
end
```

Figure 8. Phone Model with Labels

- Use an inverse. For example, the implementation of a transform such as a matrix inverse or Fourier transform might be checked by applying the inverse transformation and then comparing the result to the original data.
- Use self-checking data. This approach requires augmenting the system to produce checksums or other additional data which can be checked for consistency with recorded responses. [1] presents a general approach for implementing abstract data types in software.
- Use an alternate implementation.
- Let a human decide. In the event that there is no alternate implementation, or when there is no practical means to quickly determine success or failure, it might be best to let a human decide.

The JUMBL does not directly support any of these approaches, but through the use of labels it can be made to indirectly support them. For example, a label might include the instructions to execute a command, record the response, and check the response against the specification for correctness.

The JUMBL supports recording success and failure for each event in a test case, as well as recording that the test case was stopped prior to reaching the last event.

6. Test Analysis

Statistical testing can be viewed as an experiment to quantify the reliability of the system given a usage profile. This raises two important questions. What is the quality of the system, and how much testing is sufficient to correctly assess the quality of the system? The first question can be answered by computing the system's projected reliability. The second question can be answered by looking at various *stopping criteria*.

The JUMBL uses a reliability model proposed by [7] and extended to Markov chain usage models in [12]. The reliability model used is Bayesian, and the JUMBL allows setting parameters on the model to specify prior information. This prior information is taken into account when generating the reliability estimates. The JUMBL produces individual stimulus reliabilities, a single-event reliability, and a source-to-sink single use reliability based on recorded testing experience. This model generates useful reliability estimates both when failures have been observed, and when no failures have been observed.

The JUMBL computes reliabilities using two methods. First, reliabilities are computed as if every recorded event passed. These are called *optimum* reliabilities, and they can be computed whether or not any actual testing has taken place. This allows users to plan the number of test cases necessary to reach a reliability goal, assuming all are run without failure. Second, reliabilities are computed based on recorded successes and failures. These are simply reported as the stimulus, arc, or system reliability.

One may use the reliability estimate, with prior information, as a stopping criterion [6]. The JUMBL computes an associated variance for each reliability measure, and this could also be used as a stopping criterion. The subject of when to stop testing when using Markov chain usage models has been studied, however, and the use of a measure called the Kullback-Liebler information number, also called the discriminant, has been proposed [13]. This number approaches zero as testing experience grows to resemble expected use. Thus if one runs tests unlike expected use, the discriminant will grow. As one runs tests which closely resemble expected use, the discriminant will shrink. The JUMBL computes and reports the discriminant based on tests generated (optimum) and on success and failure records.

The discriminant is an information-theoretic number based on the difference between two entropies. To make the discriminant comparable across models, a special version of the discriminant is computed which is normalized by dividing by the model's source entropy. This measure is called the *relative* discriminant, and is also reported by the JUMBL.

The JUMBL divides the test analysis into several sections. Fig. 9 shows the JUMBL's model coverage report. Stimulus reliabilities are shown in fig. 10. Note that where failures are reported, the optimum reliability is greater than the "true" projected reliability. As with the model analysis, the model is reproduced and reliabilities are given for each arc. A portion of the arc report is shown in fig. 11. Finally, a comprehensive model reliability and stopping criteria report is shown in fig. 12.

Model Statistics

Node Count	8 nodes
Arc Count	14 arcs
Stimulus Count	8 stimuli
Test Cases Recorded	205 cases
Nodes Generated	8 nodes / 8 nodes (1)
Arcs Generated	14 arcs / 14 arcs (1)
Stimuli Generated	8 stimuli / 8 stimuli (1)
Nodes Executed	8 nodes / 8 nodes (1)
Arcs Executed	14 arcs / 14 arcs (1)
Stimuli Executed	8 stimuli / 8 stimuli (1)

Figure 9. Model Test Report

Stimulus Statistics

Stimulus	Generated	Executed	Failed	Reliability / Variance		Optimum Reliability / Variance		Prior Successes / Failures	
connect	77	77	0	0.987341772	156.224964E-6	0.987341772	156.224964E-6	1	1
hang up	205	204	0	0.976635514	106.132962E-6	0.976744186	105.161949E-6	5	5
disconnect	308	308	1	0.990384615	30.4246933E-6	0.993589744	20.34877E-6	2	2
lift receiver	205	205	1	0.985645933	67.3715607E-6	0.990430622	45.1324047E-6	2	2
incoming call	279	279	1	0.992882562	25.0595028E-6	0.996441281	12.5746609E-6	1	1
dial good	109	109	0	0.990990991	79.7129176E-6	0.990990991	79.7129176E-6	1	1
dial busy	42	42	0	0.977272727	493.572084E-6	0.977272727	493.572084E-6	1	1
dial bad	38	38	0	0.975	594.512195E-6	0.975	594.512195E-6	1	1

Figure 10. Stimulus Test Statistics

Arc Statistics

Arc	Probability	Generated	Executed	Failed	Reliability / Variance		Optimum Reliability / Variance		Prior Successes / Failures	
[On Hook]										
"incoming call"	0.5	279	279	1	0.992882562	25.0595028E-6	0.996441281	12.5746609E-6	1	1
"lift receiver"	0.5	126	126	0	0.9921875	60.0888748E-6	0.9921875	60.0888748E-6	1	1
[Ring Tone]										
"connect"	0.5	77	77	0	0.987341772	156.224964E-6	0.987341772	156.224964E-6	1	1
"hang up"	0.5	32	32	0	0.970588235	815.620366E-6	0.970588235	815.620366E-6	1	1

...

Figure 11. Arc Test Statistics

Reliabilities

Single Event Reliability	0.984835969
Single Event Variance	15.2760188E-6
Single Event Optimum Reliability	0.986893113
Single Event Optimum Variance	13.7076688E-6
Single Use Reliability	0.940801133
Single Use Variance	4.79341123E-3
Single Use Optimum Reliability	0.948593993
Single Use Optimum Variance	3.74007245E-3
Arc Source Entropy	0.888888926 bits
Kullback Discrimination	92.2717127E-3 bits
Relative Kullback Discrimination	10.381 %
Optimum Kullback Discrimination	91.7464673E-3 bits
Optimum Relative Kullback Discrimination	10.321 %

Figure 12. Model Reliability Report

7. Future

At present, the JUMBL provides support for every phase of statistical testing. Models can be constructed from libraries of common components using a language such as TML and then analyzed to determine their stochastic properties. The JUMBL can generate tests in various ways and convert test cases into automated test information. The results of testing can be analyzed to determine expected system performance in the field and plan for maintenance.

The JUMBL does not directly support construction of models or automated execution of test cases. For model construction there are many excellent text and graph editors, and there is little reason to develop a new editor. For automated execution domain-specific needs must be met, and no single automation system can hope to address all domains. The JUMBL thus provides a simple means to link it with other automated test tools.

Planned enhancements to the JUMBL include improvements to the user interface, and improved reporting of analytical results. Additionally, new input / output filters, and special templates for graphical editors are under development to facilitate constructing usage models.

While the JUMBL provides means to generate tests, and supports the use of prior information in its reliability model, it does not provide a nice means to integrate test records generated in different ways and against different distributions. This problem was studied in [8], and future versions of the JUMBL may address this issue.

References

[1] S. Antoy and D. Hamlet, “Automatically Checking an Implementation Against its Formal Specification,” IEEE Transactions on Software Engineering, v. 26, n. 1, January 2000, pp. 55-69.

[2] E.R. Gansner and S.C. North, “An Open Graph Visualization System and Its Applications to Software Engineering,” Software—Practice and Experience, v. 30, n. 11, September 2000, pp. 1203-1233.

[3] A.M. Gibbons, Algorithmic Graph Theory, Cambridge: Cambridge University Press, 1985.

[4] M. Himsolt, “Graphlet: Design and Implementation of a Graph Editor,” Software—Practice and Experience, v. 30, n. 11, September 2000, pp. 1303-1324.

[5] J.G. Kemmeny and J.L. Snell, Finite Markov Chains, New York, NY: Springer-Verlag, 1976.

[6] B. Littlewood and D. Wright, “Some Conservative Stopping Rules for the Operational Testing of Safety-Critical Software,” IEEE Transactions on Software Engineering, v. 23, n. 11, November 1997, pp. 673-683.

[7] K.W. Miller, L.J. Morell, R.E. Noonan, S.K. Park, D.M. Nicol, B.W. Murriel, and J.M. Voas, “Estimating the Probability of Failure When Testing Reveals No Failures,” IEEE Transactions on Software Engineering, v. 18, n. 1, January 1992, pp. 33-44.

[8] J.M. Morales, “Test Planning for Software Reuse,” M.S. Thesis, Knoxville, TN: The University of Tennessee, May 1997.

[9] J.H Poore, G.H. Walton, J.A. Whittaker, “A Constraint-Based Approach to the Representation of Software Usage Models,” Information and Software Technology, v. 42, n. 12, 1 September 2000, pp. 825-833.

[10] S.J. Prowell, “TML: A Description Language for Markov Chain Usage Models,” Information and Software Technology, v. 42, n. 12, 1 September 2000, pp. 835-844.

[11] S.J. Prowell, C.J. Trammell, R.C. Linger, J.H. Poore, Cleanroom Software Engineering: Technology and Process, Reading, MA: Addison-Wesley, 1999.

[12] K.D. Sayre, “Improved Techniques for Software Testing Based on Markov Chain Usage Models,” Ph.D. Dissertation, Knoxville, TN: The University of Tennessee, December 1999.

[13] K.D. Sayre and J.H. Poore, “Stopping Criteria for Statistical Testing,” Information and Software Technology, v. 42, n. 12, 1 September 2000, pp. 851-857.

[14] J.A. Whittaker and J.A. Poore, “Markov Analysis of Software Specifications,” ACM Transactions on Software Engineering and Methodology, January 1993, pp. 93-106.