Engineering Practices for Statistical Testing

Jesse H. Poore, *University of Tennessee*
Carmen J. Trammell, *Software Engineering Technology, Inc.*

This article describes the application of statistical science to the testing and evaluation of software and software-intensive systems. Engineering practices are described for statistical testing based on a usage model, which is an engineering formalism that represents the use of a system in a specific environment or situation, or for a specific customer class. Engineering practices for statistical testing are based on a view of software use as a stochastic process and of software testing as a problem amenable to statistical solution.

When a population is too large for exhaustive study, as is the case for all possible uses of a software system, a statistically correct sample must be drawn as a basis for inferences about the population. In statistical testing of software, testing is treated as an engineering problem to be solved by statistical methods. Figure 1 shows the parallel between a classical statistical design and statistical software testing.

Under a statistical protocol, the environment of use can be modeled, and a statistically valid statement can be made about the expected operational performance of the software based on its test performance.

Statistical testing refers to the application of statistical science to testing software-intensive systems. It begins with characterizing all possible scenarios of use, includes analytical advice on design for testability, and ends with random testing to support estimates of the reliability of the system in field use.

A usage model is a characterization of all possible scenarios of software use at a given level of abstraction. Usage models can be constructed before code is written, and the model-building process can lead to improvements in the software specification that enhance testability.

A test case is any traversal of the usage model. A random test case is a traversal of the usage model based on state transitions that are randomly selected from a usage probability distribution.

Certification means attaining reliability and confidence goals for an environment of use following a protocol of demonstration. This protocol must be well defined, open to evaluation, and repeatable. As with any costly testing program, it is important that low-quality software be rejected (for example, when selecting commercial-off-the-shelf software), or returned to development (for revision and verification) quickly and inexpensively, and that testing continue only if progress toward certification of the software is being made.

Engineering practice refers to procedures that can be applied to problems of a recognizable type to achieve predictable and repeatable results. Engineering practices are derived from an appropriate science base, but the theoretical details of the science are organized, packaged, and often automated to be unobtrusive during application. Engineering practices are designed to get work done rapidly and correctly. The statistical testing process involves the six steps depicted in Figure 2. The engineering practices for each step are described in the succeeding sections.

Operational Usage Modeling

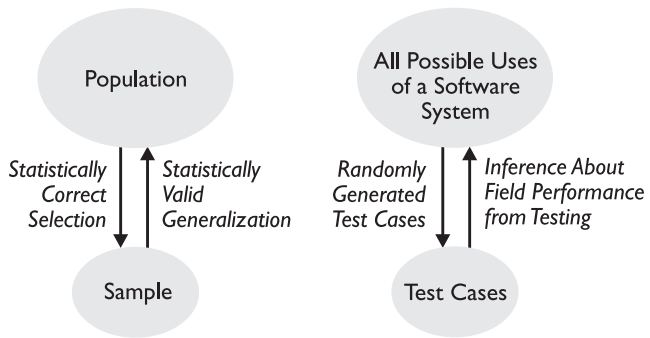
Building Model Structure

Usage models characterize the infinite population of scenarios of use. Usage models are built from specifications, user guides, or even existing systems. The user might be a human, a hardware device, another software system, or some combination. More than one model might be constructed for a single system if there is more than one environment of interest.

The basic task in model building is to identify the states of use of the system and the possible transitions among states of use. This information is encoded into highly structured Markov chains [1] in the form of directed graphs and stochastic matrices. Every possible scenario of use, at the chosen level of abstraction, is represented by the model. Thus, every possible scenario of use is represented in the analysis, traceable on the model, and potentially generated from the model as a test case. Figure 3 portrays a simple usage model as a directed graph with transition probabilities on the arcs.

Models should be designed in a standard form that consists of connected submodels with a single entry and single exit. States and arcs can be expanded like macros. Submodels

Figure 1. *Parallel between a statistical design and software testing.*



of canonical form can be collapsed to states or arcs. This permits model validation, specification analysis, test planning, and test case generation to occur on various levels of abstraction. The structure of the usage models should be reviewed with the specification writers, the real or prospective users, the developers, and the testers. Users and specification writers are essential to represent the application domain and the work-flow of the application. Developers get an early opportunity to see how the system will be used and look ahead to implementation strategies that take account of use and work-flow. Testers are typically the usage model designers and therefore get an early opportunity to plan certification and to define and automate the test environment.

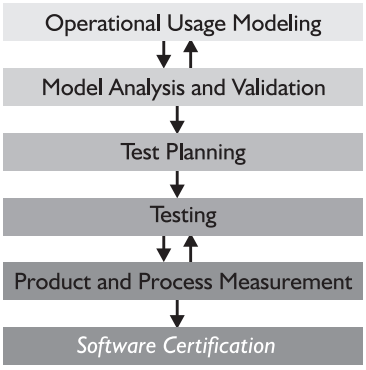
Most usage modeling experience to date is with embedded real-time systems, application program interfaces, and graphical user interfaces. Models as small as 20 states and 100 arcs have proven highly useful. Typical models are on the order of 500 states and 2,000 arcs; large models of more than 2,000 states and 20,000 arcs are in use. Even the largest models developed to date are small in comparison to similar mathematical models used in other fields of science and engineering and are manageable with available tool support.

Assigning Transition Probabilities

Transition probabilities among states in a usage model come from historical or projected usage data for the application. Because transition probabilities represent classes of users, environments of use, or special usage situations, there may be several sets of probabilities for a single model structure. Moreover, as the system progresses through the lifecycle, the probability set may change several times based on maturation of system use and availability of more information.

When extensive field data for similar or predecessor systems exists, a probability value may be known for every arc of the model. For new systems, one might stipulate expected practice based on user interviews, user guides, and training programs. This is a reasonable starting point but should be open to revision as new information becomes available.

Figure 2. *The statistical testing process.*



Generating Transition Probabilities

An alternative to the direct assignment of transition probabilities just discussed is to generate them as the solution to a system of equations [2]. Usage models can be represented by a system of constraints (written as equations or inequalities in terms of the transition probabilities as variables). The matrix of transition probabilities can be generated as the solution to the system. In general, three forms of constraints are used to define a model:

- Structural constraints are so named because they define model structure: the states themselves and both possible and impossible transitions among the usage states.
- Usage constraints represent information about known or expected patterns of system use.
- Management constraints reflect controls on the testing process to enforce budget, schedule, or policy decisions.

Probability values can be related to each other by a function to represent what is known about the relationship without overstating the data and knowledge. Most usage models can be defined with extremely simple constraints.

Engineering Practice for Operational Usage Modeling

Step 1. Identify the system boundary and all hardware, software, and human users of the software and the stimuli they can send the software.

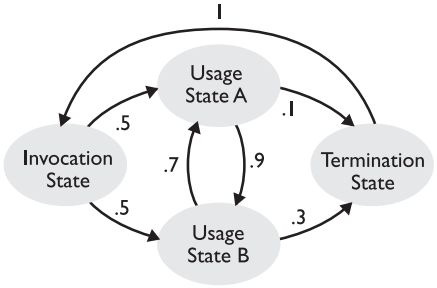


Figure 3. *Markov chain usage model.*

Step 2. Define the structure of the usage model in terms of the possible sequencing of stimuli. Identify any areas where the software specification will result in excessive (as opposed to essential) complexity and cost in system development. Make recommendations for possible simplification.

Step 3. Define the important environments of use for the software, e.g., routine use, hazardous use, malicious use, maximum capacity use, and determine the number of environments to be modeled. Continue the process for each model.

Step 4. Define the transition probabilities of the usage model.

Model Analysis and Validation

Long-Run Characteristics of the Usage Model

A Markov chain is a thoroughly studied mathematical model for which a standard set of statistics exists. In this case, the standard statistics calculated for the usage model have important interpretations for resource allocation, safety analysis, test planning, and field support. Statistics that are routinely calculated from the model and used for these purposes include the following:

Long-run occupancy of each state – the usage profile as a percentage of time spent in each state.

Occurrence probability – probability of occurrence of each state in a random use of the software.

Occurrence frequency – expected number of occurrences of each state in a random use of the software.

First occurrence – for each state, the expected number of uses of the software before it will first occur.

Expected sequence length – the expected number of state transitions in a random use of the software; the average length of a use case or test case.

Analytical results are studied during model validation, and surprises are not uncommon. Parts of systems thought to be unimportant might get surprisingly heavy use, while parts that consume a large amount of the development budget might see little use. Since a usage model is based on the software specification rather than the code, it can be done early in the lifecycle to inform the development process as well as testing and certification.

Engineering Practice for Model Analysis and Validation

Step 1. Generate the standard analytical results for the model. Interpret analytical values in terms of the specification and expected usage to validate their correctness or reasonableness.

Step 2. Change the model structure or constraints if necessary. Changes to the structure may be needed to correctly represent the specification; changes to constraints may be needed to correctly represent usage or test management issues.

Step 3. If the model has been changed, repeat Steps 1 and 2.

Step 4. Generate some test cases and confirm that they look realistic; if not, return to Step 2.

Step 5. Use the model and its implications to inform development activities such as performance planning, correctness verification, safety analysis, and test planning.

Test Planning

Crafted (Nonrandom) Test Cases

There are compelling reasons for creating special, nonrandom test cases. Such testing can remove uncertainty about how the system will perform in various circumstances and can contribute to effectiveness and control over all testing, both crafted and random.

Following are types of nonrandom testing that may be useful prior to random testing.

Model coverage tests. Using just the structure of the model, a graph-theoretic algorithm generates the minimal sequence of test events (least cost sequence) to cover all arcs (and therefore all states). If it is practical to conduct this test, it is a good first step in that it will confirm that the testers know how to conduct testing and evaluate the results for every state of use and every possible transition.

Mandatory tests. Any specific test sequences that are required on contractual, policy, moral, or ethical grounds can be mapped onto the model and run.

(Nonrandom) regression tests. Regression test suites can be mapped to the model. This is an effective way to discover the redundancy in the test suite and assess its omissions.

Critical but unlikely use. Critical states, transitions, and subpaths that would have low likelihood of arising in a random sample can be identified from the model and tested by crafted cases.

Importance tests. Importance sampling can be implemented by adding management constraints and an objective function that will produce the transition probabilities that will emphasize the "value" in the sampling process.

Partition testing. The usage model can be used to identify and define partitions to gain sampling efficiency.

Random Test Cases

Random test cases may be automatically generated from the usage model. Each test case is a "random walk" through the model, from the initial state to the terminal state. Test cases may be generated as scripts for human testers or as input sequences for automated testing. Post-processing of test cases often further facilitates human or automated evaluation. One may generate as large a set of test cases as the budget and schedule will bear and establish bounds on test outcomes before incurring the cost of performing the tests.

Engineering Practice for Test Planning

Random testing should begin only after all crafted testing has been completed.

Step 1. Using the expected test case length derived during model analysis, estimate and generate the number of random test cases that can be run within the schedule and budget.

Step 2. Define the best-case scenario. Assume that no failures occur in random testing, and determine the values of product quality and process sufficiency that can be achieved by running the number of test cases generated in Step 1. (These measures are described in the "Product and Process Measurement" section.)

Step 3. Define the worst-case scenario. Assume some profile of failures and construct a failure log based on the profile. (This may be done for several scenarios.) Again, determine the values of product and process certification measures under the scenario. The comparison of these values with actual certification goals will reveal how much bad news the budget and schedule can absorb.

Step 4. Analyze the coverage of model states, arcs, and paths that will occur.

Step 5. Analysis might show that testing as planned and budgeted cannot, even in the case of no failures, satisfy requirements for model coverage or demonstrable reliability. Given this fact, one may choose to either revise goals or revise plans.

Testing

It is essential to the integrity of the certification process to maintain experimental control throughout random testing. Experimental control refers to complying with the assumptions associated with a statistical protocol. Results must be evaluated consistently. The team must ensure a common understanding of all test materials and policies so that consistent test decisions are

made. Other steps to ensure experimental control are given in [3, Chap. 17].

Engineering Practice for Testing

Step 1. Hold the specification and oracle constant for each version of the software that is tested.

Step 2. Sustain the conditions in the environment throughout testing.

Step 3. Monitor the performance of human testers to prevent "drift."

Step 4. Run test cases in the order in which they are generated.

Step 5. Schedule regular communication among testers for discussion of matters that may affect test judgment.

Step 6. Log all failures.

Step 7. Maintain at least two testing chains, i.e., testing records encoded as Markov chains, one for the current version of the software and one for the history of testing across all versions. The current-version testing chain will be used for certification and stopping decisions. The historical testing chain will be used to study the development and testing processes.

Step 8. If one or more failures occur during testing of the current version, a decision must be made about whether to stop testing. Many factors may be involved, including the nature of the failures, schedules, and organizational policies. Monitor reliability, confidence, and convergence of the testing chain to the usage model to guide stopping decisions.

Step 9. If no failures are seen during testing, base stopping decisions on reliability, confidence, and convergence measures together with remaining schedules and budgets.

Product and Process Measurement

The usage model from which the test cases are generated is called the "usage chain." A chain of initially identical structure is developed to record actual testing experience, called the "testing chain." The progress of testing is monitored by tracking measures calculated from these two chains.

Product Measures

A reliability measure is calculated from the testing chain along with confidence intervals. This reliability is defined strictly in terms of the failure experience recorded in the testing chain; there are no other mathematical assumptions. This definition of reliability is applicable whenever testing has revealed one or more failures. (When testing reveals no failures, distributional models should be used, e.g., [3, Chap. 5; 4].)

Process Measures

An information theoretic comparison of the usage and testing chains is computed to assess the degree to which the testing experience has become representative of expected field use. Its graph will have a terrace-like appearance of declines and plateaus. The trend in the measure reveals the rate at which the usage and testing chains are becoming indistinguishable. As the two converge, it becomes less likely that new information will be gained by further testing.

Certification

The certification process involves ongoing evaluation of the merits of continued testing. Stopping criteria are based on reliability, confidence, and uncertainty remaining. Decisions to continue testing are based on an assessment that the goals of testing can still be realized within the schedule and budget remaining.

In most cases, users of statistical testing methods release a version of the software in which no failures have been observed. Reliability estimates such as those in [3, Chap. 5; 4] are recommended in this case.

Software is sometimes released with known faults. If the test data includes failures, reliability and confidence may be calculated from the testing chain. The reliability measure computed in this manner reflects all aspects of the sequences tested, including the probability weighting defined by the usage model.

Certification is always relative to a protocol, and the protocol includes the entire testing process and all work products. An independent audit of testing must be possible to confirm correctness of reports. An independent repetition of the protocol should produce the same conclusions to within acceptable statistical variation.

Conclusions

The model construction and validation process is an investment in understanding how the system will be used. Several calculations flow directly from the usage models without further assumptions that quantify the size and complexity of the testing problem. For many practitioners, this quantitative, defensible characterization of size and complexity provides insights that are overwhelming. Practitioners have variously slashed requirements, pruned user options, erected firewalls, extended schedules, extended budgets, initiated test automation efforts, and decimated reliability goals. If one believes that the usage model accurately captures the capability described in the specifications and that the probabilities represent the intended environment of use, the calculations and conclusions based on them are inescapable.

Work in progress is focused on composition of usage models from components, synthesizing information about the whole from the components, and combining testing information across the product lifecycle. ◆

About the Authors

Jesse H. Poore is professor of computer science at the University of Tennessee. He works with industrial and government sponsors on practical software engineering problems. He served as assistant to the president for information technology and professor of computer science at the Georgia Institute of Technology. In 1983, he was executive director of the Committee on Science and Technology in the U.S. House of Representatives. From 1971 to 1980, he was associate professor of mathematics and director of the Computing Center at Florida State University. He holds a doctorate in information and computer science from Georgia Tech.

Department of Computer Science
University of Tennessee, 107 Ayres Hall

Knoxville, TN 37996-1301
Voice: 423-974-5784
Fax: 423-974-4404
E-mail: poore@cs.utk.edu

Carmen J. Trammell is a software consultant with Software Engineering Technology, Inc. She was formerly research assistant professor and manager of the Software Quality Research Laboratory in the Department of Computer Science at the University of Tennessee. She has held technical and management positions in software projects through Oak Ridge National Laboratory, Martin Marietta Energy Systems, and Software Engineering Technology and currently works with industry and government on software engineering process definition and improvement. She has done software development and testing for the U.S. Army and the U.S. Navy, academic teaching on military bases overseas, industrial training for Department of Defense (DoD) contractors, and research and development under contract to the DoD Software Technology for Adaptable, Reliable Systems Program. She has a master's degree in computer science and holds a doctorate in psychology from the University of Tennessee.

Software Engineering Technology, Inc.
5516 Lonas Road, Suite 110
Knoxville, TN 37909
Voice: 423-450-5151 ext. 240
Fax: 423-450-9110
E-mail: trammell@toolset.com
Internet: http://www.toolset.com

References

1. Kemeny, J.G. and J.L. Snell, *Finite Markov Chains*, Van Nostrand, Princeton, N.J., 1960.
2. Walton, G.H., *Generating Transition Probabilities for Markov Chain Usage Models*, Department of Computer Science, University of Tennessee, Knoxville, Tenn., 1995.
3. Poore, J.H. and C.J. Trammell, *Cleanroom Software Engineering: A Reader*, Oxford, England, Blackwell Publishers, 1996.
4. Miller, K.W., et al., "Estimating the Probability of Failure When Testing Reveals No Failures," *IEEE Transactions on Software Engineering*, January 1992.

Coming Events

Tenth Annual Software Technology Conference (STC '98)

Dates: April 19-23, 1998
Location: Salt Palace Convention Center, Salt Lake City, Utah
Co-hosts: Ogden Air Logistics Center commander and the Software Technology Support Center
Contact: Dana Dovenbarger
Voice: 801-775-4932 DSN 777-7411
E-mail: dovenbar@oodis01.hill.af.mil

IEEE Computer Society International Conference on Computer Languages 1998

Dates: May 14-16, 1998
Location: Loyola University Chicago, Chicago, Ill.
Sponsors: IEEE Computer Society Technical Committee on Computer Languages in cooperation with the Association for Computing Machinery Special Interest Group on Programming Languages
Contact: Internet: http://www.math.luc.edu/iccl98/

7th IEEE North Atlantic Test Workshop

Dates: May 28-29, 1998
Location: West Greenwich, R.I.
Subject: Provides a forum for discussions on the latest issues relating to higher quality, more economical, and more efficient testing methods and designs. The workshop will focus on "Reliability and Testing Issues for the 21st Century."
Sponsors: IEEE Computer Society, Test Technology Technical Committee, and the University of Rhode Island
Contact: Jim Monzel
Voice: 802-769-6428
Fax: 802-769-7509
E-mail: jmonzel@vnet.ibm.com

Quality Week '98, Conference "Countdown to 2000"

Dates: May 26-29, 1998
Location: San Francisco, Calif.
Subject: Focuses on advances in software test technology, quality control, risk management, software safety, and test automation, software analysis methodologies, and advanced software quality processes.
Contact: Rita Bral
Voice: 800-942-SOFT (800-942-7638)
Fax: 415-957-0730
E-mail: qw@soft.com
Internet: http://www.soft.com/QualWeek/QW98

Conference on Risk Management Software Engineering Institute (SEI)

Dates: June 8-10, 1998
Location: Crystal Gateway Marriott, Crystal City, Va.
Sponsor: Software Engineering Institute
Contact: SEI Customer Relations
Voice: 412-268-5800
Fax: 412-268-5758
E-mail: customer-relations@sei.cmu.edu