An Automated Testing Environment to support Operational Profiles of Software Intensive Systems

Robert S. Oshana
Raytheon Systems Company
oshana@raytheon.com
(972)344-7083

Abstract:

Raytheon Systems Company is a defense electronics company that has been actively engaged in a software process improvement effort at the organizational and program levels for over a decade. The company uses the Software Engineering Institute's (SEI) Capability Maturity Model (CMM) as a basis for their improvement efforts and has been formally assessed at level 3 (defined).

The project is a real-time embedded software application using a heterogeneous computer architecture consisting of the two fundamental computing environments;

- PowerPC single board computer (SBC) using Ada for embedded command and control,
- Digital signal processors (DSP) using the C programming language for a primarily signal processing algorithm based application. There are significant real-time constraints within the signal processing application.

The program follows a tailored version of DoD-STD-2167A and MIL-STD-498 documentation standards. The project is built upon an Integrated Product Team (IPT) structure. There is a fully defined software development process as well as system engineering and hardware development processes.

Our testing approach is based on the concepts of software testing based on statistical principles. The statistical testing approach to software treats the software like a statistical experiment. A statistical subset of all possible software uses is first generated. Performance on this subset is used to form conclusions about operational performance based on the usage model developed. The expected operational use is represented in a usage model of the software. Test cases are then randomly generated from the usage model. These tests are executed in an operational environment. Failures are interpreted according to mathematical and statistical models.

This paper will focus on the successes and issues associated with developing a statistical testing environment for an industrial software project. The paper will also describe how both statistical testing based on software models and traditional testing based on unit and other functional tests can be combined into an effective approach to testing large software intensive systems.

A STATISTICAL EXPERIMENT

One approach to software testing is to treat the testing process like a statistical experiment. There are several components of a statistical experiment (Figure 1);

1. The *population* which is the set of items that we are attempting to make a statement about.
2. The *strata* which are the useful subsets of the population.
3. The *sample* which is the subset of the population actually used in the experiment.
4. The *inference* which is the process of estimating the population statistics based on the results from the sample.

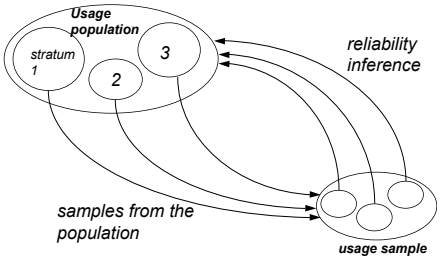


Figure 1. Sampling a population

Software use can be characterized as being stochastic. A *stochastic* process is a random process or experiment that takes place in stages. The outcome of any preceding experiment does not affect subsequent experiments. Stochastic processes can be used to model system state as a function of time. Short term or long term behavior can be studied using stochastic processes. The type of model we will use to model software behavior is the Markov chain. A *Markov chain* encodes the input domain as a set of states which represent usage history of the software from the users point of view. Arcs are used to connect the states and represent the transitions caused by the various stimuli to the system. These stimuli can be generated from hardware, human interface, other software, and so on. Finally, transition probabilities are assigned to the arcs and represent how a typical user is likely to apply stimuli to the system. A Markov chain of this type is a discrete time, finite state machine. A model of this sort can be represented as a directed graph (Figure 2) or a transition matrix (Figure 3).

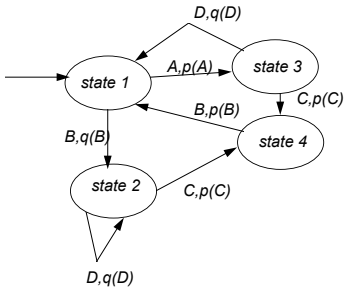
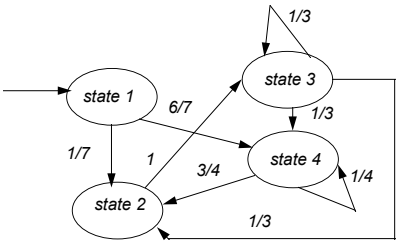


Figure 2. A Markov chain in directed graph format



A Markov chain in digraph format

$$T = \begin{array}{c} \\ \text{state 1} \\ \text{state 2} \\ \text{state 3} \\ \text{state 4} \end{array} \begin{array}{c} \begin{array}{cccc} \text{state 1} & \text{state 2} & \text{state 3} & \text{state 4} \end{array} \\ \begin{bmatrix} 0 & 1/7 & 0 & 6/7 \\ 0 & 0 & 1 & 0 \\ 0 & 1/3 & 1/3 & 1/3 \\ 0 & 3/4 & 0 & 1/4 \end{bmatrix} \end{array}$$

all rows sum to 1

A Markov chain in transition matrix format

Figure 3 Transition matrix and digraph form of a Markov chain

The common properties of Markov chains are;

1. *time homogeneous*; this means the probabilities on the arcs do not change with time.
2. *finite state*; this means the model has only a finite number of states.
3. *discrete parameter*; the state transition probabilities are discrete and not continuous.

Markov chains offer several advantages to modeling software systems:

1. The state diagram form is a common and familiar tool for modeling.
2. There are graph theoretic modeling techniques available.
3. Mathematical analysis is available for these models.
4. The models generalize well and sequences through the model can be made to look more like typical use of the software system.

There are several steps for developing Markov models for software systems;

1. Develop the usage profiles
2. Define the usage probabilities
3. Conduct the statistical test
4. Analyze the results and update the models as necessary

DEVELOPING THE USAGE PROFILES

Software systems can have multiple users or classes of users. Each of these classes of users can potentially use the system differently. The first step in developing

usage models is to determine what requires testing. This is referred to as stratifying the input domain. There are two types of stratification; user level stratification and usage level stratification. User stratification refers to who or what can stimulate the system. Usage stratification refers to what the system can do under test. In other words, user level forces you to think about all the various types of users and how they can use the system (which may be different) and usage level refers to all the functionality that the system is capable of providing.

Different modes of operation are also considered when developing a stratification plan. A finite state machine of software usage is developed based on the operational states of the software system. The test developer must understand what the software is intended to do and how it is to be used. No knowledge about how the software is designed or constructed is required to do this. Each usage condition should have a model that represents conditions under which the software is used. In general, the expected usage is modeled but other usage conditions may also be of interest. Usage should be characterized in whatever terms are important in the testing context. For example, if it is very important to test for special purposes such as a hazardous condition or malicious use, a model should be constructed representing this scenario.

DEFINE THE USAGE PROBABILITIES

Once the user and usage models have been developed the arc probabilities are assigned. These probability estimates are based on;

1. user data collected from existing systems
2. talking to or observing the user or users
3. prototyping and/or trial analysis
4. domain experts

There are three approaches to defining the usage probabilities;

1. *Uninformed approach*. In the uninformed approach uniform probabilities are assigned across the exit arcs for each state. This approach maximizes the "entropy" which is a measure of statistical uncertainty. The higher the entropy, the less representative the test sequences are to the model itself. This approach is useful when no other information is available.

2. *Informed approach*. The informed approach is used when some actual user sequences are available (from prototypes, prior versions, etc.). These estimates are driven primarily from field data. This is the best approach to use if data is available.

3. *Intended approach*. The next best approach after the informed approach is the intended approach. In this approach, test sequences are obtained by hypothesizing runs of the software by the various user and usage types. Data to support this approach comes from user data and domain experts.

CONDUCT THE STATISTICAL TEST

Once the usage models have been created and the probabilities have been assigned, the next step is to conduct the experiment. Tests are conducted using a form of Monte Carlo simulation. In this approach, random numbers are generated and used to traverse the model of the software. Using this type of simulation, statistical integrity is preserved. A random path through the software, driven by the usage model, arc probabilities, and the random numbers generated create a statistical experiment on the software. There are tools available to generate test scripts using these techniques. These tools also generate a wealth of statistical data and visual aids to help the testing organization establish reliability estimates and other stopping criteria.

Tests are executed until the required acceptance goals or stopping criteria are established (Figure 4). The testing organization needs to determine what is acceptable and unacceptable for each test executed. Acceptable means the software is ready for use. Unacceptable requires the software to be re-worked. Testing of this sort on software must be a controlled experiment. In order to use the statistical data available when running such an experiment, the same version of software must be used in each of the test cases. A new version of software marks the beginning of a new experiment. The outcomes of the trials must also be consistent. When analyzing the results of each test, pass/fail criteria must be used consistently by the testers and evaluators. Automated pass/fail evaluation (sometimes called oracles) will help make the pass/fail criteria more consistent. The testing organization must also be explicit to ensure experimental integrity.

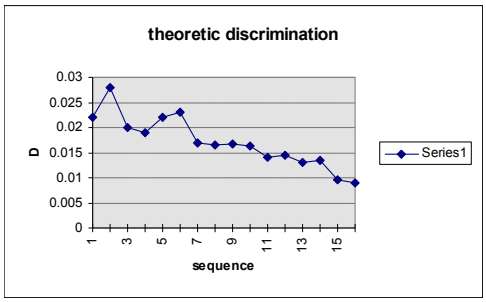


Figure 4. Theoretic discrimination used to determine stopping criteria

There are situations where pure statistical testing cannot be done. Other forms of non-statistical testing may be included because they are required by the customer, required by the contract for the product, or required by law. There are ways, using statistical testing tools, to generate tests scripts that produce the fastest coverage of a given usage model in order to achieve a level of coverage type testing. One effective technique is to perform both non-statistical tests (such as coverage tests) as well as statistical tests in the testing program. If the non-statistical tests are performed prior to the statistical tests, the statistics generated by the statistical testing process will still be valid. Performing non-statistical tests after statistical tests may invalidate the reliability estimates of the software.

Our program has developed an automated testing environment (Figure 5) that uses statistically generated test scripts based on user and usage profiles [1] as well as other operational profiles [2] to drive the testing effort.

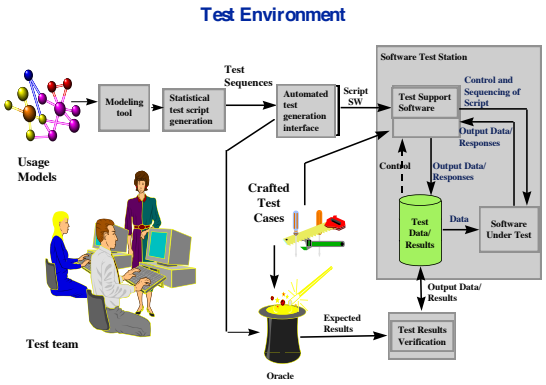


Figure 5. Testing environment automating the execution of statistically generated test scripts

The software components used in the test environment consisted of the following main components (Figure 6);

- Operator test software; also referred to as the "user function". There was a different user function for each of the usage models developed for the software. These programs were written in "C" and generated the message sequences that drove the software under test for each of the test scripts generated from the usage model,
- Labview interface; This software was composed of a number of different Labview "virtual instruments" used to provide an interface to the operator executing the tests,
- Station specific Special Test Equipment (STE) software; this software provides the low level functionality required by each of the different software test stations,
- Common STE software; this software was the Application Programming Interface (API) to the rest of the software. It provided capabilities to watch for certain events occurring in the software under test, log those results, and provide information to the user function software and Labview virtual instruments.

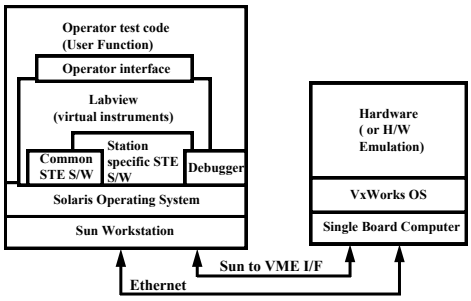


Figure 6. Software components for the testing environment

Once the test has been executed and the input and output events logged in an output file, the data is parsed into several different oracle files (Figure 7). Each of these oracle files

was used for a different pass/fail criteria. "Generic" oracles were used to slice the data in such a way as to determine if the proper high level sequencing of the test was performed correctly. Other oracles were used to slice the data to determine other pass/fail criteria such as;

- Does the data reflect the expected outputs as determined by the system engineering model(s)?,
- Does the raw output data match the expected outputs defined in the algorithm document?,
- Does the output message sequence match what was expected in the Software Requirements Document?

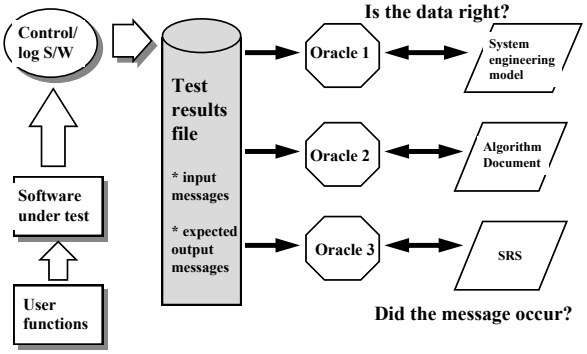


Figure 7. Testing Environment Oracle

TESTING AT THE PROGRAM LEVEL

Software testing at the program level was performed using various levels of statistical testing supplemented with unit and functional testing where required as well as operational profiles testing techniques. Unit and function testing was generally performed for the algorithmic portions of the software. This software involved the more mathematically based functions, with strict real time constraints. Various implementation and optimization strategies were performed to effectively map the algorithms to the processor for optimal real-time performance. We developed an effective model for certification of algorithmically intensive software that included a formal code inspection and correctness verification phase, an optional level of unit and function testing (based on the nature of the algorithms) which included both static and run time analysis, an operational profile phase using real data collected from various user environments, and a final statistical testing phase using usage models developed for different user and usage stratifications (Figure 8).

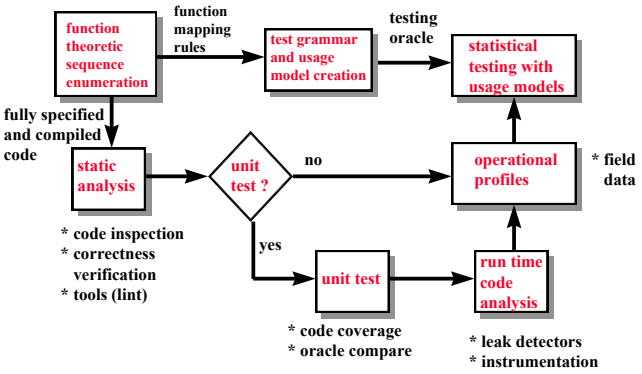


Figure 8. Testing model

LIMITATIONS TO MODELING

There are limitations to modeling software systems that must be understood in order to properly analyze the statistics generated when using statistical techniques such as the ones discussed in this paper. The quote "All models are wrong, but some are useful" has some truth to it. The main point to remember when modeling any type of system using any technique is that all modeling approaches lack, to some extent, the naturalness for representative power. The Markov approach has limitations in handling some of the common issues such as counting and concurrency. But there are ways of getting around some of these issues but it may result in state explosion (larger models). Also, the more abstract the model is, the less confident one should be in the predicted reliability generated by the tools. If one is careful not to blindly accept the statistics without proper analysis, then the statistical techniques discussed in this paper can be very useful in any testing organization and help lead to higher quality software products.

References

1. Musa, John D. "Operational Profiles in Software Reliability Engineering", IEEE Software, March 1993

2. Walton, G.H., J.H.Poore, and C.J.Trammel. "Statistical Testing of Software Based on a Usage Model", Software Practice and Experience, January 1993.