Software Testing using Markov Chain Usage Model

Raj Kumari
Research Scholar, Department of Computer Science, Mewar University, Chittorgarh, Rajasthan, India.

Abstract
Model base testing is the application of model base design and optionally also executing artifact to perform Software testing. Model can be used to represent the desired behavior of a system under test (SUT) or represent testing strategies and a test environment. A model describing SUT is usually an abstract, partial presentation of the system under test desired behavior. Test cases derived from such a model are functional test on the same level of abstraction as the model. The test cases are collectively known as abstract test suite. Software testing is to be an expensive yet integral of the software development life cycle (SDLC). Testing enable the developer to verify the product and provide the mean for and user to validate the system according to their needs. The choice of software modeling methods depends on the types of testing to be performed on the system. Testing are performed during the designed phase of a software project. The goal of all software testing is to improve the quality of software product by uncovering as many faults as possible within the allocated time.

Keywords: Markovian process, Kullback discriminant, MCUM, SUT, SDLC, MBT

Introduction
There is an abundance of testing styles in the discipline of software engineering. These have come to be used by the industry as solutions to address software quality. The popularization of object orientation and models in software engineering, there has been a growth in black box testing techniques that are collectively dubbed model-based testing.

Model-based testing or MBT
MBT has as its roots applications in hardware testing; most notably telephone switches, and recently has spread to a wide variety of software domains. The wealth of published work portraying case studies in both academic and industrial settings is a sign of growing interest in this style of testing. In the software testing area, aiming to provide information about the system’s reliability, the statistical testing of software sprung up. The technique combines elements of statistics to software test techniques, allowing the collection of measures about the system.

Markovian process
The markov chains bring several limitations, both in modeling and computing terms. This system can be represented as a markovian process when the time spent in each state appears exponentially distributed. A collection of states is associated to this markovian process. This system can assume only one state at any time. On the other hand, a recent formalism called stochastic automata networks shows several characteristics that may compensate the limitations show by the preceding formalism.

Usage models
In the software testing area, there are several techniques of test case generation that aim to find, in the program, fail that lead to correction of errors in its codification. The main techniques are the function testing, that aims to test the system behavior based on its specification, and structural testing that aims to test the application based in its source code. It can be used in many states of software life cycle i.e. to refine the specification, complexity evaluation, conduct the verification efforts, identification of the determined events frequency, planning of testing schedules and infer about the software reliability.

Testing process
Current statistical testing processes assume that enough testing will be performed choose long run statistical effects come into play. This may require a great deal of testing, therefore making some results primarily applicable to software organizations with extensive test automation. Many of the results presented here are concerned with situations where not enough testing will be performed to justify an assumption of long run statistical effects. The typical process followed when performing statistical testing is presented. An overview of new results will be provided in conjunction with discussion of the stage of the testing process impacted by the new results (indicated by shaded boxes).

1. Get Specification - Some form of a specification detailing the correct behavior of the software is needed to develop the usage model. The correct behavior of the software may be defined by a formal specification, requirements documentation, user’s manual, or a predecessor system.

2. Create Model Structure - The states of use and arcs connecting the states are identified. In current practice this stage is manual, i.e., the structure of the usage model cannot be derived automatically from a specification or other artifact. However, work is in progress to show how to derive the model structure from a sequence-based specification of the software.

3. Set Probabilities - The state transition probabilities of the usage model are assigned. The probabilities may be assigned manually or they may be represented as a set of constraints and automatically calculated to satisfy some testing goal.

4. Check and Analyze Model - Analytical results are calculated to aid in test planning and verifying that the model properly
represents the expected use of the software. The following new analytical results will be presented. A means for calculating the expected number of test cases needed to cover a state or arc with associated variance will be presented. The variance of this statistic is a newly derived result. A lower bound on the expected number of test cases needed to cover all states or arcs with associated variance will be presented. The calculation for the probability of an arc appearing in a test case will be given and checking the method of software.

5. **Execute Non-Random Tests** - Non-random test cases are crafted or generated from the usage model and then executed on the software under test. Examples of non-random tests currently being used are hand crafted tests, test cases generated in order of probability of occurrence, and test cases generated to cover all arcs in the usage model in the Introduction minimum number of testing steps. Non-random tests may be run to satisfy contractual obligations, explore a particular use of the software, help validate the usage model and test facility, or determine whether the software is stable enough for full scale testing. Current practice does not include use of non-random tests in reliability estimation.

6. **Execute Random Tests** - Test cases are generated randomly from the usage model and executed on the software under test. The test cases may be run automatically or by hand. Methods for partitioning the test cases, allocating testing effort to blocks in the partition so as to minimize reliability estimator variance, and generation of test cases from particular blocks in the partition are new results.

7. **Reliability estimation** - The testing record containing information as to which tests were run and where failures were observed in the test cases is used to estimate the reliability of the software. Currently, the reliability is estimated using the Whittaker reliability model or the binomial reliability estimator. In the Whittaker model the testing record is stored as a testing chain. The testing chain is a Markov chain that contains all states from the usage model that have been visited during testing. The probabilities of the outgoing arcs of a state in the testing chain are calculated by normalizing the number of times each outgoing arc of a state has been visited during testing. Every time a unique error is encountered during testing a new state, a failure state, and arcs to and from the failure state are added to the testing chain. The reliability is computed as the probability of going from (Invoke) to (Terminate) in the testing chain without entering a failure state.

8. **Decision to stop Testing** - The testing record is evaluated to determine whether testing should continue or stop. Currently two methods are used to measure the degree to which testing matches the expected use of the software, the Euclidean distance between the usage chain and the testing chain and the Kullback discriminant of the usage chain to the testing chain. The Euclidean distance can always be calculated. However, the Euclidean distance is not always an accurate measure of the similarity of the usage chain and the testing chain.

8.1 **Kullback discriminant**

The Kullback discriminant provides no direct information about the stability of the testing record, i.e., about whether the testing chain has converged and is unlikely to change in the future. A specific definition of convergence and a means for assessing the convergence of the testing chain will be presented. The Kullback discriminant provides a more accurate indication of the degree to which the testing chain matches the usage chain but the Kullback discriminant is not defined until all arcs in the usage model have been visited at least once during testing. In many cases it may not be feasible to continue testing until all arcs in the usage model have been covered, constraining the usefulness of the Kullback discriminant as a stopping criterion.

9. **Stop and Report Results** - After testing is finished the test results may be used for a number of purposes, such as deciding whether to release the product, evaluating Introduction page 12 whether the software development process is under control, or evaluating the performance of a new piece of technology used in the product.

**Figure – Testing Process**

**Conclusions**

A testing organization now has considerable ability to tailor the reliability estimation to the situation in order to make testing more efficient. Use the arc-based reliability models to take advantage of pre-test reliability information. Testing savings can be realized if accurate pre-test reliability information is available. Use partition testing techniques in conjunction with usage models and arc-based reliability models to increase testing efficiency. The distributions of various random variables based on the usage model (sequence length, number of sequences to cover all states or arcs, etc.) should be studied. Knowledge of the distribution underlying these random variables will allow for increased accuracy in test planning and model validation. A formal relationship between the similarity of the testing chain and the usage chain and the estimated reliability should be tabulated. Field evidence shows that a high degree of similarity
between the testing chain and the usage chain indicates that the reliability estimated from the testing experience is accurate. However, a more formal relationship is needed. Industrial models increase in size, the time needed to calculate analytical results from the model will impede work flow in the use of statistical testing.

References

Author: Raj Kumari Ph.D Research Scholar Mewar University, Chittorgarh, Rajasthan (India).

Published following papers:

“A NEW APPROACH FOR CONGESTION CONTROL IN WIRELESS NETWORK AND ANALIZE IT FOR TCP” Published By International Journal of Application or Innovation in Engineering & Management (IJAIEM) Volume 1, Issue 2, October 2012 ISSN 2319 – 4847