Better Reliability Assessment of Database Based Application Software

M. S. HOSAIN†, M. S. ALAM‡
†Department of Computer Science & Engineering
The University of Asia Pacific
House # 52/1, Road # 3/A, Dhanmondi, Dhaka – 1209
BANGLADESH
‡CSE Dept., Bangladesh University of Engineering & Technology, Dhaka – 1000, Bangladesh

Abstract:- Software systems are becoming increasingly more complex and testing it to a reliable system requires a great deal of effort. Statistical testing promises a solution to this increased testing burden and gives the opportunity to have statistical inferences like reliability, mean time to failure etc. Today there are hundreds of reliability models with more models developed every year. Still there does not exist any model that can be applied in all cases. In this paper we present the feasibility of applying statistical testing based on Markov chain usage model to database based application software and show that statistical testing gives better reliability assessment than random testing.

Key-Words:- Statistical Software Testing, Software Reliability, Stochastic Modeling, Usage Model, Random Testing.

1 Introduction

Software testing is the process of executing software products by using a set of defined inputs to verify that the execution results of the product match the predefined set of outputs [1, 2]. The goal of testing is to prove the software free of bugs and to build a reliable system. But it is impossible to test software exhaustively, i.e., testing the application with every input combination or scenario. Actually testers are *sampling* form the input population of the software under test [3].

Statistical testing is advancing rapidly in response to these needs. By implementing the collection of usage data into operational profiles, developers can utilize well-known statistics to direct the application of testing, thereby reducing redundant testing, focusing testing on portions of the software with the biggest impact on the system, and reducing the amount of testing required overall [2, 4, 5]. These improvements can significantly decrease the amount of resources required for software testing. Statistical testing can also be used to determine when it is time to stop testing a software product, through reliability [6, 7].

Statistical testing is appealing theoretically, but is limited in usage by the kinds of usage models that can be built [8]. Usage modeling based on Markov chains gains its credibility in the literature. Whittaker first proposed Markov chain based statistical testing in [9, 10]. An improved technique for software testing based on Markov chain usage model was also presented in [11]. Also a number of research efforts have been published on software reliability based on this technique [12, 13]. We also have presented our work on software reliability [14], where we have investigated the feasibility of applying Markov chain technique to large and complicated software systems and have measured reliability in terms of probability of executing a randomly selected test case from the usage model. One of the most important shortfalls of statistical testing is the lack of evidence of the effectiveness of statistical testing compared to other methodologies, such as structural testing, random testing etc [8]. The work we present here is an extension of our previous work [14] in the direction of measuring mean time to failure and showing the effectiveness of statistical testing over random testing.

The rest of the paper is organized as follows. Section 2 presents background knowledge on statistical testing, random testing and software reliability, section 3 shows case study results and finally section 4 draws a conclusion.

2 Background Knowledge

Test can be designed from a functional or a structural point of view. Structural testing does look at the implementation details like programming style, control method, source language, database design, coding details etc. In functional testing the program or system is treated as a black box. It is subjected to inputs, and its outputs are verified for conformance to specified behavior [1]. Functional testing takes the user's point of view. Statistical testing is one kind of functional testing which is applied only to large, robust products with few bugs [1] and is described as follows.

2.1 Statistical Testing

In statistical testing test data is randomly selected based on probability distributions defined across the test input domain i.e. according to the usage model of the system. Statistical testing uses an operational profile, the statistical behavior of the system under test, typically modeled as a Markov chain and often referred as usage chain [2, 9, 10, 12]. The Markov chain has two special states, one is "*Un-Invoked*" sate and the other is "*Terminated*" state. The first step of this process is to create a Markov model of software specification and assigning transition probabilities to the exiting arcs of states of the chain. Secondly, generate test cases from the model and execute the test oracle to software. And finally analyze the test results for reliability and other statistical assessment and prediction, and help with decision-making.

Assigning probabilities to arcs is an important area of research of this study because the argument that test should follow user patterns is vital. If this is not the case, then the tests are not a representative sample and all statistical conclusions are invalid. Three types of probability distributions can be used – uninformed, informed and intended [9, 10]. The *uninformed* approach simply labels the arcs out of a node with a uniform distribution. The *informed* approach uses actual user data and the *intended* approach uses the expected usage of the system under test by "a careful and reasonable user [9]" rather than the actual usage by users in the field. Our process uses the intended approach and can be adapted to the informed approach as actual usage data becomes available.

2.2 Random Testing

In random testing, which can be seen as a simpler form of statistical testing, there is an explicit lack of systematic in the choice of test data, so that there is no correlation among different tests [2].

2.3 Software Reliability

There are several ways to define reliability. Reliability can be defined as a function of time, i.e., the software will operate according to specification for a period of time [4, 15, 16]. Another simpler definition is that reliability is the probability that a randomly chosen use (test case) will be processed correctly [10, 16]. Using the definition of latter one, the mean time to failure is the average number of uses between failures [10, 16].

3 Case Study Results

3.1 Usage Markov Model

Fig. 1 shows a simple selection menu of typical software. The example given here is the same as in [14]. The up arrow key and down arrow key moves the cursor from one item to next, and wraps from top to bottom on an up-arrow and from bottom to top on a down-arrow key.

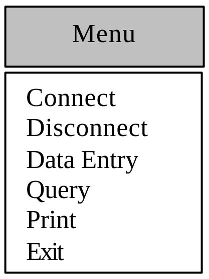

Fig. 1: Selection Menu

The first item "Connect" is used to establish connection to a database server. The connect window has two options, Ok and Cancel. Pressing the "Ok" button establishes a connection to the specified server with proper authentication and the "Cancel" button returns to previous state. Once the connection is established the next four items, Disconnect, Data Entry, Query and Print can be selected to perform their respective functions. If connection is not established, selecting these items give no response. As Connect state, Disconnect state has also Ok and Cancel button to disconnect from database server or not. From the other three options we enter another screen only for Data Entry state for simplicity and assume that the same thing could be done for other states. For data entry state we enter in a new screen, which could insert or update department record to database. The screen is pictured in Fig. 2.

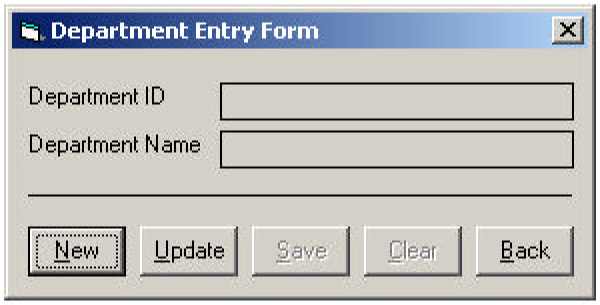

Fig. 2: Department Entry Form First State

Initially "`New`", "`Update`" and "`Back`" button are enabled and the other controls are disabled. Selecting data entry from menu displays this screen and the control focus goes on to "`New`" button. The tab key will shift the focus on the next enabled button, and will rotate right when the focus is on right-most button. If "`New`" button is pressed, "`New`", "`Update`" and "`Back`" buttons will be disabled and the disabled controls will be enabled. In that case the screen will look like Fig. 3. The same thing will happen if "`Update`" button is pressed. Now, if "`Save`" button is pressed, data provided in the text boxes will be updated to database. If "`Clear`" button is pressed the screen will go to its initial state i.e. "`New`" state and "`Back`" button returns to "`Data Entry`" state.

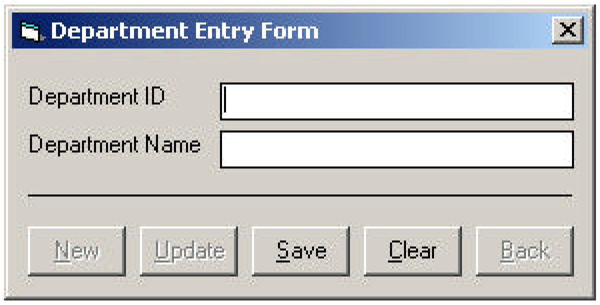

Fig. 3: Department Entry Form Second State

The software behavior is modeled in a Markov chain and is shown in Fig. 4. In the state the CL means *Cursor location* and takes on values CN, DC, DE, QR, PR or Exit for each respective menu item, and CS means *Connection status* and takes on the values Y or N. In addition, we include two states "*Un-invoked*" and "*Terminated*". A path from the initial "*Un-invoked*" state to the final "*Terminated*" state represents a single execution of the software and is known as *sequence*. In order to generate sequence statistically, probability distributions are established over the exit arcs at each state that simulates expected field usage. Table-I lists each transition for the example chain in Fig. 4 with probabilities assigned according to its *intended* use.

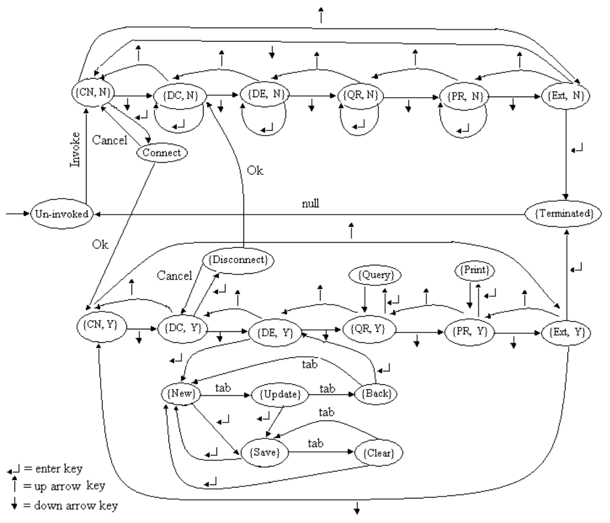

Fig. 4: Usage Markov chain for the software

3.2 Testing Chain

Testing chain T is constructed from Usage chain U. Initially the testing chain is the same as usage chain with all its arc frequencies set to zero. During testing a test case is generated from the usage chain by randomly walking from the "*Un-Invoked*" state to "*Terminated*" state according to its usage probability. The frequencies of the arcs through which the usage chain is traversed are incremented by one. If failure occurs during execution of test a new state labeled f_j is placed in Markov chain exactly as it is ordered in test sequence. The arcs to and from the new state f_j have frequency count 1. If f_j is catastrophic failure, then the run of software *P* is aborted, and the arc form f_j goes to "Terminated"; otherwise, the test sequence can continue, and the arc from f_j goes to the next state in sequence. As the testing chain T evolves it becomes more and more similar to usage chain U. A key point is that the test history T is statistically typical of the usage chain U if and only if convergence is achieved [6].

3.3 Analytical Results

For the convenient of our study we further present briefly the reliability analysis from [14]. When two stochastic processes converge, the numerical value of *log likelihood ratio* [10, 17] known as Kullback *discriminant* tends to zero. The value is computed by the following equation:

$$D\left(U, T\right) = \sum_{ij} p_i p_{ij} \log_2 \frac{p_{ij}}{\hat{p}_{ij}} \quad (1)$$

Where p is the stationary distribution of U, p_{ij} is the probability of a transition from i to j in U, and $\hat{p}_{ij}$ is the corresponding probability in T.

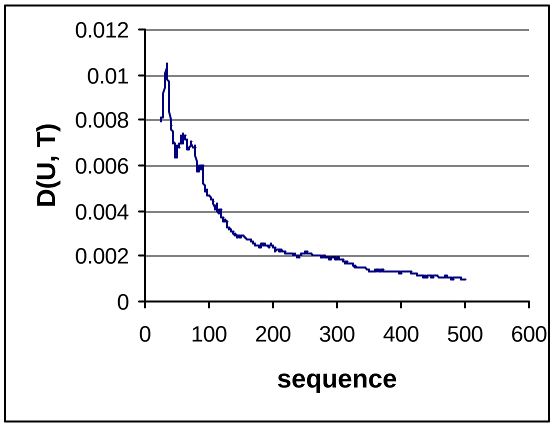

Fig. 5: Plot of *D* (*U*, *T*)

Fig. 5 shows plots of D (U, T). When testing chain grows quite similar to usage chain i.e. the test history reflects the actual usage pattern, the value of D (U, T) becomes very small. Test should stop at this point. Whenever a failure occurs the value of D (U, T) increases significantly so additional tests require minimizing that effect [10, 14].

Reliability is the probability that a randomly chosen test case beginning with "Un-invoked" and ending with the first occurrence of "Terminated" will not contain a failure state. It is measured by the following equation:

$$R_{Un-in,Term} = \hat{p}_{Un-in,Term} + \sum_{j \in t} \hat{p}_{Un-in,j} R_{j,Term} \quad (2)$$

Where t is the set of transient (non-absorbing) states. Fig. 6 depicts a plot of R for 250 sequences. Whenever there is a failure there is a sharper decrease in R, because the failures are probability-weighted according to their location in chain [10].

Table - I: Transition probabilities for the example usage chain

SL#	From state	Trans. Stimuli	To state	Est. Prob.
1	Un-Invoked	I nvoke	{CL=CN, CS=No}	1.00

2	{CL=CN, CS=No}	↓
		↑
		↵
			{CL=DC, CS=No}
			{CL=Ext, CS=No}
			{Connect}
				0.10
				0.10
				0.80

3	{CL=DC, CS=No}	↓
		↑
		↵
			{CL=DE, CS=No}
			{CL=CN, CS=No}
			{CL=DC, CS=No}
				0.33
				0.34
				0.33

4	{CL=DE, CS=No}	↓
		↑
		↵
			{CL=QR, CS=No}
			{CL=DC, CS=No}
			{CL=DE, CS=No}
				0.33
				0.34
				0.33

5	{CL=QR, CS=No}	↓
		↑
		↵
			{CL= R, CS=No}
			{CL= E, CS=No}
			{CL= R, CS=No}
				0.33
				0.34
				0.33

6	{CL=PR, CS=No}	↓
		↑
		↵
			{CL=Ext, CS=No}
			{CL=QR, CS=No}
			{CL=PR, CS=No}
				0.34
				0.33
				0.33

7	{CL=Ext, CS=No}	↓
		↑
		↵
			{CL=CN, CS=No}
			{CL=PR, CS=No}
			{Terminated}
				0.34
				0.33
				0.33

8	{CL=CN, CS=Yes}	↓
		↑
		↵
			{CL=DC, CS=Yes}
			{CL=Ext, CS=Yes}
			{CL=CN, CS=Yes}
				0.50
				0.35
				0.15

9	{CL=DC, CS=Yes}	↓
		↑
		↵
			{CL=DE, CS=Yes}
			{CL=CN, CS=Yes}
			{Disconnect}
				0.70
				0.15
				0.15

10	{CL=DE, CS=Yes}	↓
		↑
		↵
			{CL=QR, CS=Yes}
			{CL=DC, CS=Yes}
			{Data Entry New}
				0.25
				0.25
				0.50

11	{CL=QR, CS=Yes}	↓
		↑
		↵
			{CL=PR, CS=Yes}
			{CL=DE, CS=Yes}
			{Query}
				0.25
				0.25
				0.50

12	{CL=PR, CS=Yes}	↓
		↑
		↵
			{CL=Ext, CS=Yes}
			{CL=QR, CS=Yes}
			{Print}
				0.25
				0.25
				0.50

13	{CL=Ext, CS=Yes}	↓
		↑
		↵
			{CL=CN, CS=Yes}
			{CL=PR, CS=Yes}
			{Terminated}
				0.15
				0.35
				0.50

14	{Connect}	Ok
		Cancel
			{CL=CN, CS=Yes}
			{CL=CN, CS=No}
				0.85
				0.15

15	{Disconnect}	Ok
		Cancel
			{CL=DC, CS=No}
			{CL=DC, CS=Yes}
				0.60
				0.40

16	{Data Entry New}	Tab ↵	{Update} {Save}	0.40 0.60
17	{Update}	Tab ↵	{Back} {Save}	0.50 0.50

18	{Back}	Tab
		↵
			{Data Entry New}
			{CL=DE, CS=Yes}
				0.20
				0.80

19	{Save}	Tab ↵	{Clear} {Data Entry New}	0.20 0.80
20	{Clear}	Tab ↵	{Save} {Data Entry New}	0.50 0.50
21	{Query}	Query Data	{CL=QR, CS=Yes}	1.00
22	{Print}	Print Data	{CL=PR, CS=Yes}	1.00
23	{Terminated}	Null	Un-Invoked	1.00

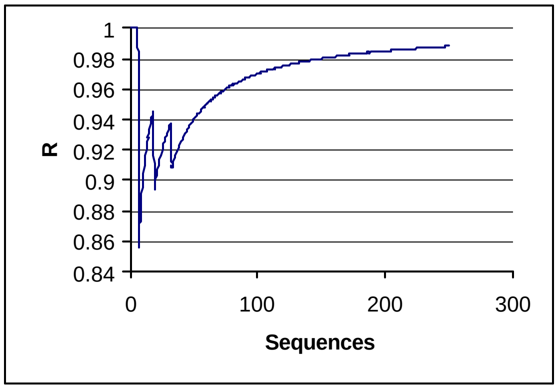

Fig. 6: Plot of R

In this study the mean time to failure is computed as the *expected number of steps between failures*, which is the expected number of state transitions encountered between occurrences of failure states in the testing chain. This value is computed as follows:

$$M = \sum_{i \in f_1,\ldots,f_m} v_i \left(\sum_{j \in u_1,\ldots,u_n} \hat{p}_{ij}(m_j + 1) \right) \quad (3)$$

where v_i is the conditional long-run probability for failure state f_i, given that the process is in a failure state, m_j is the mean number of steps until the first occurrence of any failure state from j, $u_1,\ldots,u_n$ is the set of usage chain states, and $f_1,\ldots,f_m$ is the set of failure states. Fig. 7 is a plot of M for 250 sequences generated from our example software.

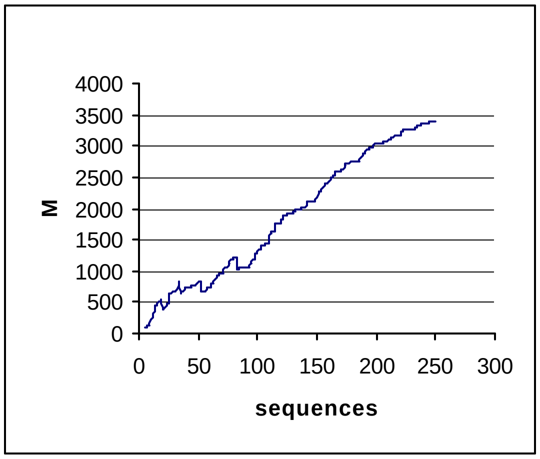

Fig. 7: Plot of *M*

The amount of time spent in any state in the long run is equal to the stationary probability of each state after the test process stops. We compute the stationary probabilities and are shown in Fig. 8.

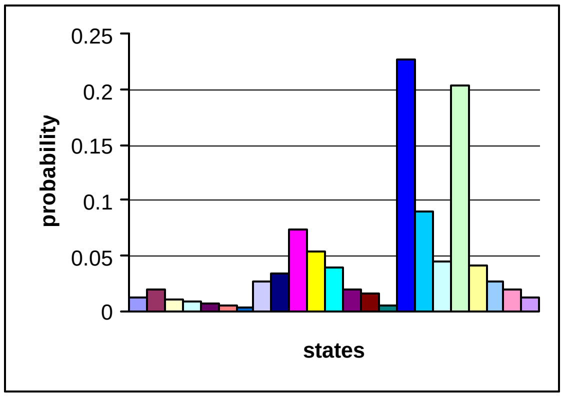

Fig. 8: Plot of stationary probabilities of states of the testing chain

3.4 Test Comparison

One of the major shortfalls of statistical testing is the lack of evidence of the effectiveness of statistical testing compared to other methodologies, such as structural testing [8], random testing etc. We assign equal probabilities to each exiting arcs from a state, generate test cases that represent random testing and measure reliability to compare the test processes.

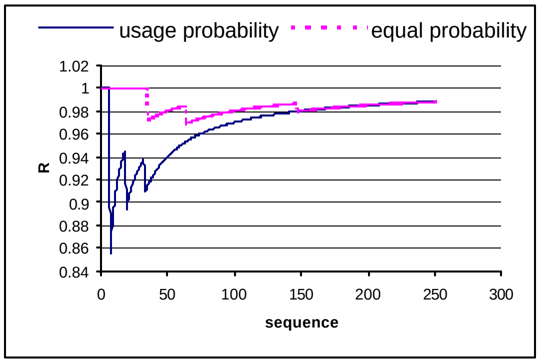

Fig. 9: Plot of R s

From Fig. 9 we find that if the fault lies on the path of heavy usage probability than it reveals early in statistical testing while the fault reveals lately in random testing. If we set the target reliability to 0.98 we see from Fig. 9 that random testing may not reveal one bug. But if the fault lies on the less usage probability path than random testing reveals the fault early than statistical testing but this does not jeopardize our test effort as the same number of bugs are revealed by statistical testing before attaining the desired reliability and this is shown in Fig. 10.

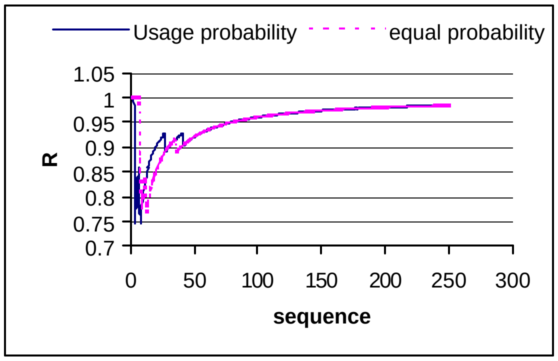

Fig. 10: Plot of R s

4 Conclusion

Statistical testing promises a practical option for software test engineers in the development and analysis of usage models and automatic test input generation. Markov chain usage models can be constructed in a diverse set of application domains. In this paper we have shown its applicability to database based application software. Though our example is a small one our approach proves its viability in measuring reliability for large and complex software systems. We also give evidence of the effectiveness of statistical testing compared to random testing. One important limitation of this approach is that the model becomes complex for large software systems. Further research could be done to represent the model in a concise way.

References:

[1] Boris Beizer, *Software Testing Techniques*, second edition, Boston, Mass.: Int'l Thomason Computer Press, 1990.

[2] HAPPANEN, Pentti, PULKKINEN, Urho, KORHONEN and Jukka, Usage Models in Reliability Assessment of Software-based Systems, STUK-YTO-TR 128, Helsinki 1997, pp. 1-48.

[3] J. A. Whittaker, Stochastic Software Testing, *Annals of Software Engineering*, Vol. 4, 1997, pp. 115-131.

[4] J. D. Musa, Operational Profiles in Software Reliability Engineering, *IEEE Software*, Vol. 10, No. 2, March 1993, pp. 14-32.

[5] Taylor H. M and Karlin S, *An Introduction to Stochastic Modeling*, International Edition, Academic Press, New York, 1984.

[6] J.D. Musa, *Software Reliability Engineering*, McGraw-Hill, New York, 1998.

[7] S. Karlin and H.M. Taylor, *A First Course in Stochastic Processes*, second ed., Academic Press, New York, 1975.

[8] Robert J. Weber, Statistical Software Testing with Parallel Modeling: A Case Study, *Proceedings of the 15th International Symposium on Software Reliability Engineering*, 2004.

[9] J. A. Whittaker and J. H. Poore, Markov Analysis of Software Specifications, *ACM Transactions on Software Engineering Methodology*, Vol. 2, January 1993, pp. 93-106.

[10] J. A. Whittaker and M. G. Thomason, A Markov Chain Model for Statistical Software Testing, *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, October 1994, pp. 812-824.

[11] Kirk Sayre, Improved Techniques for Software Testing Based on Markov Chain Usage Models, Ph.D. dissertation, Dept. of Computer Science, University of Tennessee, Knoxville, USA, December 1999.

[12] Chaitanya Kallepalli and Jeff Tian. Measuring and Modeling Usage and Reliability for Statistical Web Testing, *IEEE transactions on software engineering*, Vol. 27, No. 11, November 2001, pp. 1023-2001.

[13] F. Zhen and C. Peng, A System Test Methodology Based on the Markov Chain Usage Model, *Proceedings of the 8th International Conference on Computer Supported Cooperative Work and Design*, 2003, pp. 160 – 165.

[14] Md. Shazzad Hosain and Md. Shamsul Alam, Software Reliability Using Markov Chain Usage Model, *Proc. of the 3rd International Conference on Electrical & Computer Engineering*, Dhaka, Bangladesh, December 2004.

[15] J.D. Musa, *Software Reliability Engineering*, New York: McGraw-Hill, 1998.

[16] J.H. Poore, Harlan D. Mills and David Mutchler, Planning and Certifying Software System Reliability, *IEEE software*, Vol. 10, No. 1, January 1993, pp. 88-99.

[17] S. Kullback, *Information theory and statistics*, New York: Wiley, 1958.