Analytical Comparison of Manual and Automatic Software Testing Using Markov Chain Model

Raj Kumari, Dr. Roshan Lal Hiranwal, Dr. Ashish Shah

Abstract: Model based 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. A Markov chain (discrete-time Markov chain or DTMC), named after Andrey Markov, a Russian mathematician is a random process that undergoes transitions from one state to another on a state space. It must possess a property that is usually characterized as "memorylessness". The probability distribution of the next state depends only on the current state and not on the sequence of events that preceded it. This specific kind of "memorylessness" is called the Markov property.

Index Terms— Markov Chain Model, MarkovDG, Software under Test, Discrete Time Markov Chain, Memorylessness, Test Cases

1 INTRODUCTION

A discrete-time random process involves a system which is in a certain state at each step, with the state changing randomly between steps. The benefit of using this concept is that it saves lots of the efforts done by the software tester to generate the input sequence for testing the software. It also helps in saving time and costs of hiring the expert.

As the Markov Chain Model based on state spaces that change from one state to another depending on present state not on past state. For example, Markov chain model of baby's behavior include "playing", "eating", "sleeping", and "crying" as states.

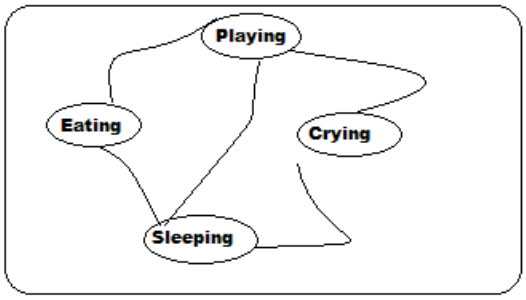


Figure 1: Depicts the baby's behavior of states.

Markov chain model tells us probability of transitioning from one state to another state that a baby currently crying will engaged into playing, eating or sleeping.

- *Raj Kumari, Research Scholar, Department of Computer Science, Mewar University, Chittorgarh, Rajasthan, India, E-mail:rajsihag83@gmail.com*
- *Dr. Roshan Lal Hiranwal Visiting Faculty, Mewar University, Chittorgarh and Associate Professor, Govt. P.G College Karnal(Haryana),India, E-mail:roshanhiranwal@gmail.com*
- *Dr. Ashish Shah, Visiting Faculty, Mewar University, Chittorgarh, Rajasthan, India, E-mail:asheesh.shah@gmail.com*

Since the system changes randomly, it is generally impossible to predict with certainty the state of a Markov chain at a given point in the future. However, the statistical properties of the system's future can be predicted. In many applications, it is these statistical properties that are important.

Similarly Markov Chain model can be used for Software Testing without knowing about previous state. Here in our case states may refer to the specific functions or actions performed by the software application. Markov chain model use the concept of "memorylessness" such that it does not bother about previous state whether it is in whatever state. The main idea behind choosing the Markov chain model is that memorylessness. According to Markov Property, Researcher trying in the automation of software testing, that it is not necessary to bother about previous output received. In the simple word the test cases generated on the basis of random inputs that does not links with the previous state or output received.

The actual idea is that to design the test cases for testing a software application, tester may follows and cares about previous sequence of states so that he can test the software application effectively, which is very time consuming and requires lots of experience and knowledge about the software application.
With the Markov Chain Model's concept of "memorylessness", researcher wants to develop a tool that generates the test cases with the concept of Markov Property. The Test Data in test cases generated randomly within the domain of the input.

2 UTILITY OF PROPOSED MODEL

Testing a software application is one of the most important but time consuming task. In software testing, Test Cases are

drawn from the sample of possible uses according to the sample distribution. The Markov Chain Model works with finite state machines. In similar fashion, states of use of software are represented as states in Markov Chain Model.

Manual Testing requires lots of mind exercise whether some tools can also requires to do mind exercise to test an application. Our tool MarkovDG generates test cases for all the classes from the domain randomly so that each and every perspective of the application should be tested and find out more bugs so that application should fulfill the organization needs.

These are the following benefits for using MarkovDG:

- Saves 50% of test planning. Test Planning is a plan about how to execute the test to find the errors and bugs in an application. This tool based on requirements of the application to be tested given by the user to the tool generate the test cases with the property of "memorylessness".

- Identify the bugs within the domain. Domain of the application is the possible inputs that can be input to the application. MarkovDG has the capability to identify the bugs within the domain.

- Identify the gaps in requirements. By identifying the errors and bugs our research helps to find the 'discrepancies' in the application.

3 OPERATING MARKOVDG

MarkovDG is developed to generate the data with the concept of memorylessness based on the Markov Chain Model. Markov Chain Model demonstrates that the data is randomly used to input for testing of the software. Analytical results associated with Markov chains facilitate informative analysis of the sequences. The test input sequences generated from the chain and applied to the software are themselves a stochastic model and are used to create a second Markov chain to encapsulate the history of the test, including any observed failure information.

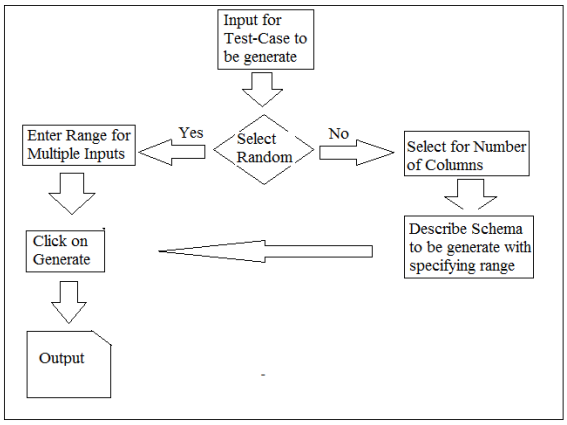
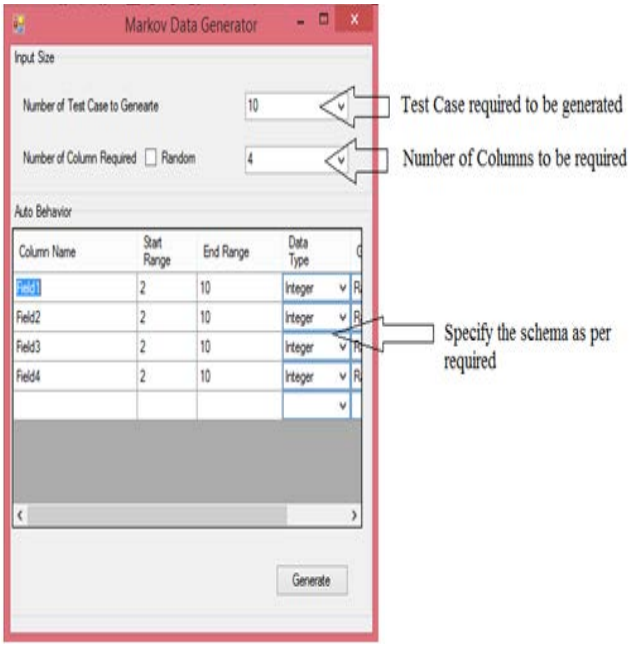


Figure 2: The above figure depicts the flow to operate the MarkovDG Tool.

A sequence of test inputs are to be given to generate different test cases.A number of test cases facilitates us to distinguish the appropriate selection of test data and the result thus obtained is as per the requirement of the user.

The Figure 2 shows how to generate test data based on the random selection of inputs and decision taken based on the selection of a range of input sequences.On the basis of given inputs the decision based on the random selection of data is to be taken into consideration to generate the different test case.

It is the choice of the user for selection of a number of inputs on the random basis and to generate the number of test cases based on multiple inputs and comparing the results thus obtained.The decision is upto the user whether the inputs suit for the appropriate result on the basis of her choice of selection of inputs to generate the test data.



Snapshot 1: The above snapshot describes the operational requirement to be feeded into the MarkovDG tool.

The schema can be easily defined with the MarkovDG Data generator for Software Testing. Tester should specify the details according to the software to be tested.

4 EMPIRICAL STUDY

An Empirical study performed on Elevator or lift Software. It is one of the well known transportation of goods and people between floors or level of a multi floors building. To operate and manage the lift appropriately need for the software is must. Researcher chooses the elevator software to be tested using the concept of 'memoryless' and 'nonmemoryless' means MarkovDG and manually respectively.

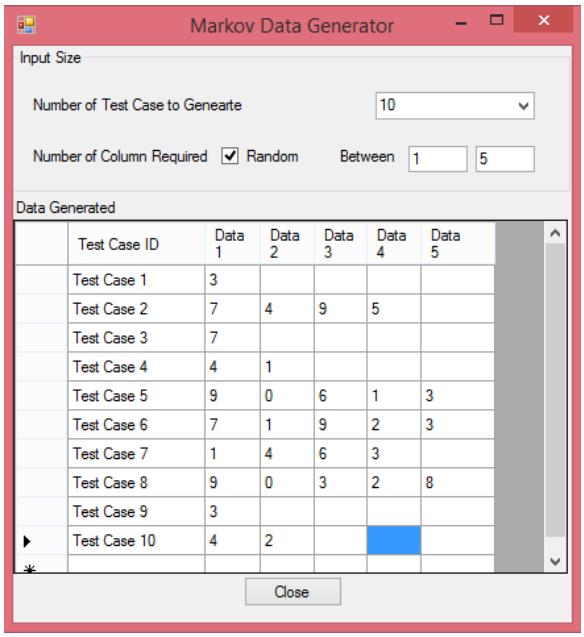
Following is the Test Cases generated manually:

Test Case#	Test Steps	Pre Condition	Expected Result	Status
1	1. Click on the exe file.	User should have application(exe)	Application window should be opened	Pass
2	1. Launch the application. 2. Press 'S'	Application window is opened.	It should start from '0' floor	Pass
3	1. Press '4'. 2. Press 'S'	Application window is opened and lift is at floor '0'.	It should go at floor '4' and show message 'Door opening. Door Closed'.	Pass
4	1. Press '0'. 2. Press 'S'	Application window is opened and lift is at floor '4'.	It should go at floor '0' and show message 'Door opening. Door Closed'.	Pass
5	1. Press '1', '6', '8', '9' 2. Press 'S'	Application window is opened and lift is at floor '0'.	1. It should go up at floor '1' and show message 'Door opening' 2. It should go up at floor '6' and show message 'Door opening' 3. It should go up at floor '8' and show message 'Door opening. Door Closed' 4. It should go up at floor '9' and show message 'Door opening. Door Closed'	Pass
6	1. Press '2', '3', '7', '9' 2. Press 'S'	Application window is opened and lift is at floor '5'.	1. It should go down at floor '3' and show message 'Door opening' 2. It should go down at floor '2' and show message 'Door opening' 3. It should move up now 4. It should go up at floor '7' and show message 'Door opening. Door Closed' 5. It should go up at floor '9' and show message 'Door opening. Door Closed'	Pass

7	1. Press '3', '4', '6', '7' 2. Press 'S'	Application window is opened and lift is at floor '9'.	1. It should go down at floor '7' and show message 'Door opening' 2. It should go down at floor '6' and show message 'Door opening' 3. It should go down at floor '4' and show message 'Door opening. Door Closed' 4. It should go down at floor '3' and show message 'Door opening. Door Closed'	Pass
8	1. Press '2', '3', '6', '7' 2. Press 'S'	Application window is opened and lift is at floor '5'.	1. It should go up at floor '6' and show message 'Door opening' 2. It should go up at floor '7' and show message 'Door opening' 3. It should move down now 4. It should go down at floor '3' and show message 'Door opening. Door Closed' 5. It should go down at floor '2' and show message 'Door opening. Door Closed'	Pass
9	1. Press '2',''6' and '8'. 2. Press 'S' 3. Elevator is at floor '8' and press '5' 4. Press 'S'	Application window is opened and lift is at floor '9'.	1. It should go at floor '8' and show message 'Door opening' 2. It should go at floor '6' and show message 'Door opening' 3. It should go at floor '5' and show message 'Door opening' 4. It should go at floor '2' and show message 'Door opening'	Fail
10	1. Press 'X'	Application window is opened.	Application window should be closed.	Pass

Table 1: Shows The Test Cases and Results received

Now move to the MarkovDG, The Snapshot given below depicts the test cases generated via MarkovDG Tool.



Test Case #	Test Steps	Pre Condition	Expected Result	Status
1	1. Press '3' 2. Press 'S'	Application window is opened and lift is at floor '0'.	1. It should go up at floor '3' and show message 'Door opening'	Pass
2	1. Press '7', '4', '9', '5' 2. Press 'S'	Application window is opened and lift is at floor '3'.	1. It should go up at floor '4' and show message 'Door opening' 2. It should go up at floor '5' and show message 'Door opening' 3. It should go up at floor '7' and show message 'Door opening. Door Closed' 4. It should go up at floor '9' and show message 'Door opening. Door Closed'	Pass

3	1. Press ‘7’ 2. Press 'S'	Application window is opened and lift is at floor '9'.	1. It should go down at floor '7' and show message 'Door opening. Door Closed'	Pass
4	1. Press ‘4’, '1' 2. Press 'S'	Application window is opened and lift is at floor '7'.	1. It should go down at floor '4' and show message 'Door opening. Door Closed' 2. It should go down at floor '1' and show message 'Door opening. Door Closed'	Pass
5	1. Press ‘9’, '0','6','1','3' 2. Press 'S'	Application window is opened and lift is at floor '1'.	1. It should go down at floor '0' and show message 'Door opening' 2. It should move up now 3. It should go up at floor '3' and show message 'Door opening. Door Closed' 4. It should go up at floor '6' and show message 'Door opening. Door Closed' 5. It should go up at floor '9' and show message 'Door opening. Door Closed'	Fails
6	1. Press ‘7’, '1','9','2','3' 2. Press 'S'	Application window is opened and lift is at floor '9'.	1. It should go down at floor '7' and show message 'Door opening. Door Closed' 2. It should go down at floor '3' and show message 'Door opening. Door Closed' 3. It should go down at floor '2' and show message 'Door opening. Door Closed' 2. It should go down at floor '1' and show message 'Door opening. Door Closed'	Fails
7	1. Press ‘1’, '4','6','3' 2. Press 'S'	Application window is opened and lift is at floor '5'.	1. It should go up at floor '6' and show message 'Door opening' 3. It moves down now. 2. It should go down at floor '4' and show message 'Door opening' 3. It should go down at floor '3' and show message 'Door opening. Door Closed' 4. It should go down at floor '1' and show message 'Door opening. Door Closed'	Pass
8	1. Press ‘9’, '0','3','2','8' 2. Press 'S'	Application window is opened and lift is at floor '1'.	1. It should go down at floor '0' and show message 'Door opening' 2. It should move up now 3. It should go up at floor '2' and show message 'Door opening' 4. It should go up at floor '3' and show message 'Door opening. Door Closed' 5. It should go up at floor '8' and show message 'Door opening. Door Closed' 6. It should go up at floor '9' and show message 'Door opening. Door Closed'	Pass
9	1. Press ‘3’ 2. Press 'S'	Application window is opened and lift is at floor '9'.	1. It should go down at floor '3' and show message 'Door opening'	Pass
10	1. Press ‘4’, '2' 2. Press 'S'	Application window is opened and lift is at floor '3'.	1. It should go up at floor '4' and show message 'Door opening' 2. It should go down at floor '2' and show message 'Door opening'	Fails

Table 2: depicts the behavior of Test Cases generated via MarkovDG Tool

5 FINAL RESULTS

The final results of the research are so promising. The comparison analysis between both the test cases generated manually and automatically is different. In manual test case execution, only one failure occurs and in the auto generated with MarkovDG there occurs three failures. So here the researcher's study reveals that the MarkovDG generated test cases cover many aspects and able to find out more and more bugs.

6 FUTURE SCOPE

Finding issues in the product is not alone the work of a tester in that company, He/she should act smart and automate the regular stuffs that are to be done by the testers. This is about solving the problems along with the developers and business people.

The researchers conclude that we should eliminate manual testing all together because if we have to run the tests more than about 3 times, it's just cheaper to start out automating it, so we can run the tests repeatedly with less (hopefully close to 0) effort. There is a bit of manual testing buried in creating only automated tests, because sometimes we just have to "try stuff" to see how it actually works. But the focus should be (in my opinion) automate all the testing.

In this research work the tool is developed to generate test cases to automate the testing with respect to find out the bugs. Some of the considerations are to identify the case study for biggest software failures and how the testing can automated and executed to find out more and more errors/ bugs/ failures. Our study reveals that the test case generated via MarkovDG Tool is best and helps to find out many more bugs. The research can be enhanced to generate test cases for the testing using fuzzy logic, ontology etc. These areas may produce more viable solutions to software testing.

REFERNCES

1. A. S. Andreou, K. A. Economides and A. A. Sofokleous, "An automatic software test-data generation scheme based on data flow criteria and genetic algorithms", 7th IEEE International Conference on Computer and Information Technology, pp. 867-872, 2007.
2. Balakrishnan, N., Mohanty S. G., Aki S. Start-up demonstration tests under Markov dependence model with corrective actions, Annals of the Institute of Statistical Mathematics, 49:155–169,1997.
3. J. A. Whittaker, "What is Software Testing? And Why Is It So Hard?" IEEE Software, January 2000, pp. 70-79.
4. J. A. Whittaker and J. H. Poore, "Markov Analysis of Software Specifications". ACM Transactions on Software Engineering Methodology. V. 2, pp. 93-106, January 2002.
5. J. Whittaker and M. Thomason, "A Markov Chain Model for Statistical Software Testing," IEEE Trans. Software. Eng., vol. 20, no. 10, pp. 812–824, Oct. 1994.
6. P. McMinn, "Search-based software test data generation: a survey", Software Testing, Verification & Reliability, Vol. 14, No. 2, pp. 105–156, 2004.
7. S. Desikan and G. Ramesh, "Software testing principles & practices", Pearson Education, 2007.
8. Yashodhan Kanoria, Subhasish Mitra and Andrea Montanari "Statistical Static Timing Analysis uses Markov Chain Monte Carlo" in 2010 EDAA.
9. Yi Wan, Chengwen WU "Software reliability model based on stochastic theory" in 2009 IEEE.
10. Z. W. Liang, W. H. Sen, Z. Jun and X. D. Jian, "Novel particle swarm based on stochastic theory" in 2009 IEEE.