JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

User Oriented Software Reliability by Markov Chain Model

Satish Kumar Nath
Lecturer- Computer Engineering
Board of Technical Education Rajasthan (BTER), Jodhpur

Abstract— **User oriented reliability is related to user profile and usage of the system. A system reliable for one user may not be reliable for other user. User oriented reliability can be represented as a function of the reliability of the components and frequency distribution of utilization of these components. In this work the Software Reliability of Library Management System module of SHARP software is calculated. For computation of software reliability, modular structure of chosen software was presented in form of Markov Chain Model. Reliabilities of sub modules are based on data received from end user survey of SHARP Software. Inter module transition probabilities are also based on this questionnaire. Effect of individual module over net system reliability is computed. Along with this criticality of each module is also estimated. Testing of these modules can be skipped and testing team might have more time to test more critical modules.**

Keywords - Model Based Testing, Markov Chain, Software testing with reduced test cases, Software testing basis on usage profile, Software Reliability, User oriented Software Reliability

I. INTRODUCTION

As software gets more complex, gains more features, and is more interconnected, it becomes more and more difficult to test. Software testing is difficult because of the large number of effect parameters. Today software has penetrated daily life of layman. Software developed now days potentially affects millions of people, enabling them to do their jobs effectively and efficiently. So the reliability of software has been an essential quality of it.[1]

Software testing is a process, or a series of processes, designed to make sure computer code does what it was designed to do and that it does not do anything unintended. Software should be predictable and consistent, offering no surprises to users. [2] The design of test cases for software and other engineered products is very challenging job. The selection of proper test case is very important issue for conformance testing in software engineering. Methods for the development of the test cases have received much attention in now days with conformance testing of communication protocols. Here the test cases are intended to determine whether a given protocol implementation satisfies all properties required by protocol specification. [3]

The user-oriented reliability of a program (in a certain user environment) is defined as the probability that the program will give the correct output with a typical set of input data from that user environment. Since the sequence of codes executed in a particular run is dependent on the input data, and an error in the non-executed statements or branches does not have any effect on the output of the program, the system reliability depends on the probability that a bug is activated in the run. The reliability of the system, therefore, depends on the user profile, which summarizes the dynamic characteristics of a typical execution of the program in a particular user environment.

II. *MARKOV CHAIN MODEL*

The software failure depends on its operation profile. User and its usage pattern drive the execution path among sub-systems. The usage of various sub system of a software system can be modeled using MARKOV chain method. According to usage to particular module of software system, testing priority and amount of testing time is decided. One of the important activity in testing environment is automatic test case generation - description of a test, independent of the way a given software system is designed.

Markov chain usage models are constructed to specify how a system to be tested is expected to be used once released into the field. It can be used to analyze expected use, generate tests, determine when to stop testing, and reason about the outcome of testing. Markov chain usage models are directed graphs, in which states of use are connected by arcs labeled with usage events. A usage event is an external stimulus applied to the system under test, while different states of use are used to enable proper sequencing and relative likelihood of inputs.[19]

It is very difficult to give a formal definition of the term “software reliability”. One can say that the reliability of a program is equal to one if correct, and zero if incorrect. However, many such "incorrect" programs give us the correct answer most of the time. It is better to evaluate the reliability of the program by a probabilistic measure as one minus the probability of failure. With such an understanding, and neglecting the performance requirements for the time being, the reliability of a piece of software may be evaluated from two points of view

To measure the reliability of a program, we can rate the reliability of a program by the "number" of software bugs left in the program at a particular stage. At other hand, we may also treat the reliability of a program from the viewpoint of the quality of the service it provides to a user.

III. *USER-ORIENTED RELIABILITY*

The user-oriented reliability of a program (in a certain user environment) is defined as the probability that the program will give the correct output with a typical set of input data from that user environment. Since the sequence of codes executed in a particular run is dependent on the input data, and an error in the non-executed statements or branches does not have any effect on the output of the program, the system reliability depends on the probability that a bug is activated in the run. The reliability of the system, therefore, depends on the user profile, which summarizes the dynamic characteristics of a typical execution of the program in a particular user environment.

The definition of user-oriented reliability and its relationship to the user profile is matter of discussion. A Markov reliability model is formulated under the assumptions that both module reliabilities and inter-module control transfers are independent. The potential applications of the model include reliability estimation, testing strategy, maintenance philosophy, and estimation of penalty cost. The concept of user-oriented reliability and a similar reliability model might also be applicable to hardware systems.

IV. *INTRODUCTION TO SHARP SOFTWARE AND ITS LIBRARY MANAGEMENT SYSTEM*

SHARP is an ERP for Educational Institutes. Its objective is to manage all resource of an educational institute and perform some basic operations. SHARP is developed by using VB.NET (framework 2.1) and MS SQL Server as its database. Its main modules are as shown in table 1.1:

Table 4.1: Various Sub Module of Library Mgmt System

SN	Name of Module	Name of Sub Module
1	Login Module	-
2	Show Home Screen Along with Master Menu	-
3	Book Management	3.1 New book
		3.2 Edit/Delete Book
4	Student Management	4.1 New Student
		4.2 Edit/Delete Student
5	Book Issue/Submit	-
6	Fine Calculation (Account Tally)	-
7	Various MIS Generation (Log)	-
8	Print the MIS Report / Log	-
9	Log Out	-

Library Management System is one of the important modules of SHARP. Among the various modules of SHARP, I have opted Library module for applying the User Oriented Reliability Model. Library module is responsible for computerize management of Library of an educational institute. Using this model the reliability of Library module will be predicted. The Library Module is further divided into various sub modules as shown in table 1.1.

V. *APPLYING THE RELIABILITY MODEL ON LIBRARY MANAGEMENT SYSTEM OF SHARP*

As we want to represent the structure of the "Library Management System" by a digraph where every node Mi represents a Sub-module and a directed edge (Mi, Mj) represents a possible transfer of control from module Mi to Mj. Every directed edge (Mi, Mj) is associated with a "inter module transition probability" Pij.

As result we developed a MARKOV Chain model of Library Management System. This module has a single entry and a single exit node. Every node in the graph is state of the Markov process and initial state (login sub module) is the entry node of the program digraph. Two terminal states C (correct) and F (failure) are added to the digraph. With each node Ni, an additional directed edge (Mi, F) is added with transition 1 - Ri probability. It represents the occurrence of an error in the successful execution of module Mi. Through such tool accurate reliability of each module or sub-module can be calculated. In absence of such tool in SHARP software reliabilities of each sub module must be assumed.

Table 5.1: Rel. of Various Sub Modules of Library Mgmt System

Module	Reliability Assumed	Module	Reliability Assumed
M_1	$R_1 = .980$	M_8	$R_8 = .910$
M_2	$R_2 = .990$	M_9	$R_9 = .912$
M_3	$R_3 = .950$	M_{10}	$R_{10} = .950$
M_4	$R_4 = .900$	M_{11}	$R_{11} = .975$
M_5	$R_5 = .947$	M_{12}	$R_{12} = .923$
M_6	$R_6 = .922$	M_{13}	$R_{13} = .995$
M_7	$R_7 = .953$		

Next parameter is branching Probability. To find out the probability of transition from one sub-module to another sub-module, a questionnaire is prepared and distributed to Library staff of various colleges using SHARP software's Library Module. Average branching probabilities between the sub-modules Mi and Mj are as follows:

$P_{1,2} = 1$					
$P_{2,3} = .07$	$P_{2,4} = .85$	$P_{2,5} = .03$	$P_{2,6} = .01$	$P_{2,7} = .03$	$P_{2,13} = .01$
$P_{3,2} = .04$	$P_{3,8} = .80$	$P_{3,9} = .15$	$P_{3,13} = .01$		
$P_{4,2} = .85$	$P_{4,13} = .15$				
$P_{5,2} = .04$	$P_{5,10} = .90$	$P_{5,11} = .05$	$P_{5,13} = .01$		
$P_{6,2} = .78$	$P_{6,12} = .20$	$P_{6,13} = .02$			
$P_{7,2} = .57$	$P_{7,12} = .40$	$P_{7,13} = .03$			
$P_{8,2} = .92$	$P_{8,13} = .08$				
$P_{9,2} = .88$	$P_{9,13} = .12$				
$P_{10,2} = .85$	$P_{10,13} = .15$				
$P_{11,2} = .91$	$P_{11,13} = .09$				
$P_{12,2} = .93$	$P_{12,13} = .07$				

Figure 5.1: Inter Module Transition Probabilities of Library Module

Using above mentioned module reliabilities and inter module transition probabilities, a Markov Chain control flow graph is developed as shown in figure 5.2. Matrix Q is prepared accordingly and finally reliability of the Library module is calculated as 76.40%.

VI. *RESULT ANALYSIS*

For estimating the effect of individual module on overall system reliability, an experiment has been designed where value of individual module's reliability is change from 0 (minimum) to 1 (maximum). Between these two points increment step of 0.1 is given. For each module total 11 readings had taken. (0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0). With this configuration system reliability is calculated and drawn a graph for findings.

Figure 5.2: Markov Chain Flow Graph of the Library Module of SHARP

From the figure 6.1 it is clear the Effect of Reliability of Module-1 over NET Reliability of System is linear. As the Reliability of Module-1 increases, the NET Reliability of System increases.

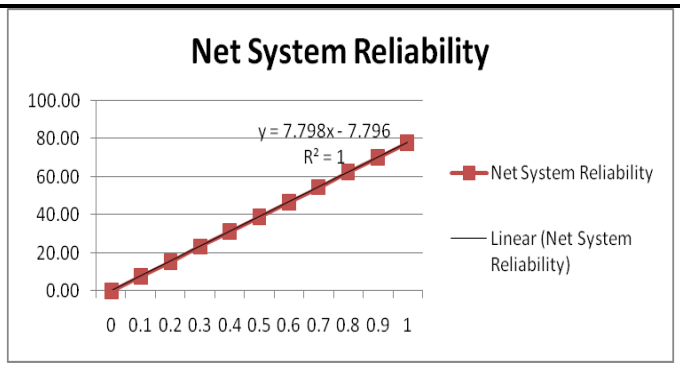
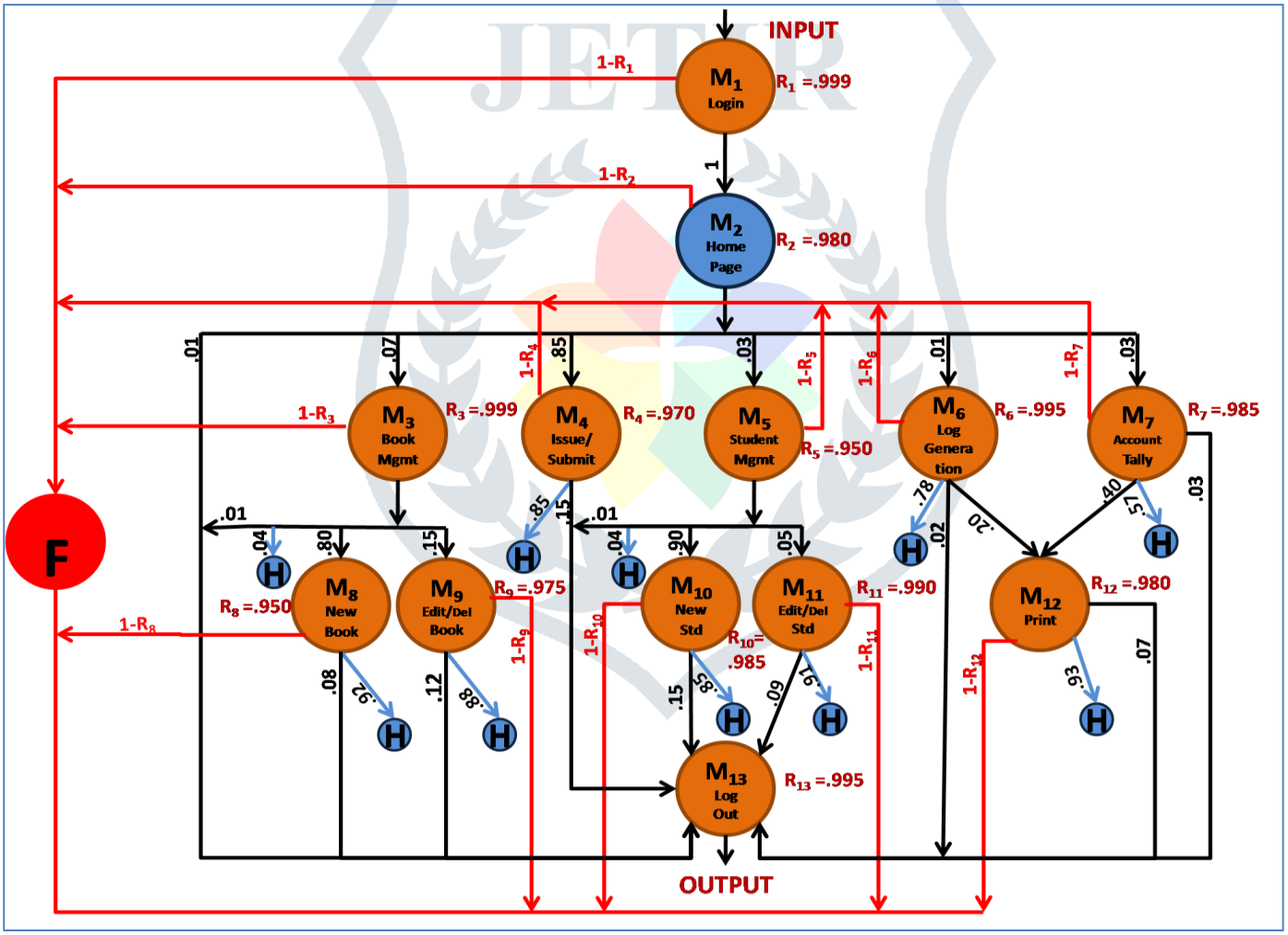


Figure 6.1: Effect of Reliability of Module-1 over NET Reliability of System

As shown in the figure 6.2 the Effect of Reliability of Module-2 over NET Reliability of System is of degree four. If Module-2 is failed to perform its task then, the NET Reliability of System will become zero. This fact made the module 2 as one of the most critical module.



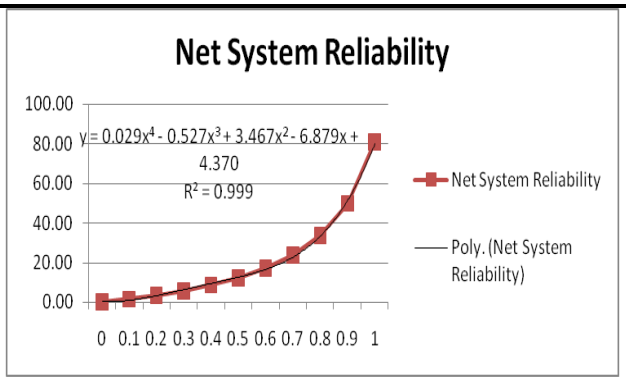


Figure 6.2: Effect of Reliability of Module-2 over NET Reliability of System

As shown in the figure 6.3 the Effect of Reliability of Module-3 over NET Reliability of System is very less compared to module 1 and module 2. If Module-3 is failed to perform its task then, the NET Reliability of System will be very less affected.

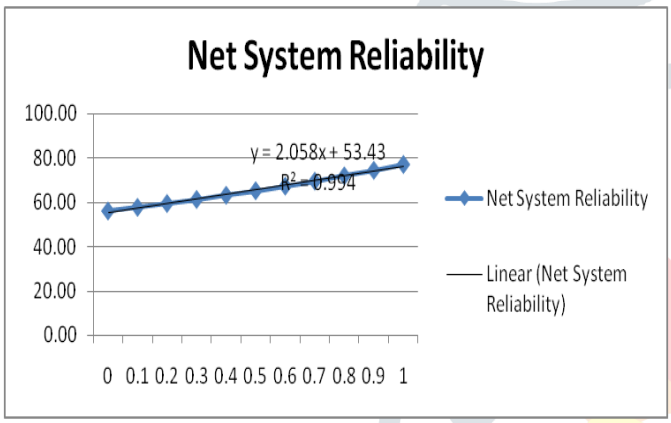


Figure 6.3: Effect of Reliability of Module-3 over NET Reliability of System

This fact made the module 3 as one of the least critical module. Therefore these types of modules can be ignored during model based testing.

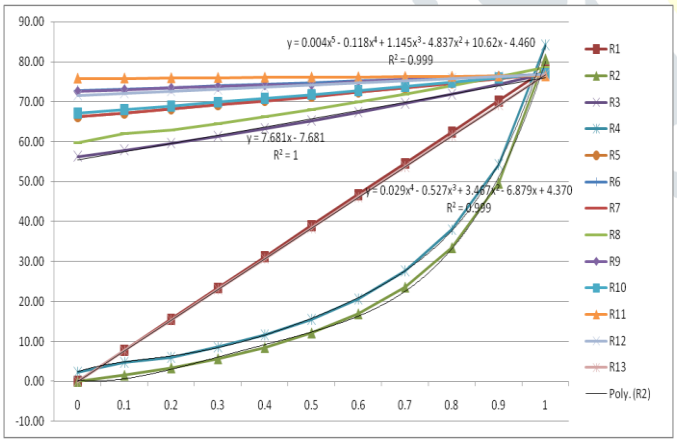


Figure 6.4: Effect of all modules on overall System Reliability

In our work module M3, M5, M6, M7, M8, M9, M10, M11 and M12 has very less effect on net system reliability. At other hand module M1, M2, M4 and M13 has significant effect on net system reliability. So testing team can emphasis more on these module and test case generation is done accordingly. If we are able to ensure maximum reliability of these modules then net system reliability will be maintained accordingly.

VII. *CALCULATION OF RELATIVE CRITICALITY OF INDIVIDUAL MODULE*

Effect of an individual module on NET system reliability is measured and represented in various line graphs shown above. These graphs and trend lines can be used to observe the impact of that particular module on net system reliability. To compare the impacts of such modules, criticality of individual module has to be calculated. For this purpose, two values for each module are considered. First value (Rx) is net system reliability when an individual module's reliability is minimum (Zero value). Second value (Ry) is net system reliability when individual module's reliability is maximum (One value).

Criticality = (Rx – Ry) and
Relative Criticality = (Ry-Rx)/ Total

For each module (from M1 to M13) two values Rx and Ry were opted. Rx denotes Net System Reliability when particular module's reliability assumed lowest as Zero. Similarly Ry denotes Net System Reliability when particular module's reliability assumed highest as one. Criticality can be computed by subtracting the Rx from Ry. In same fashion criticality for all modules (M1 to M13) were calculated. All values of Criticality are added to get a total value. This "total" value is further used to compute the relative criticality of modules. Figure 7.1 shows a PIE Chart of relative criticality of all modules. This PIE chart clearly shows the most critical modules and less critical modules. Modules that have greater impact on net system reliability are occupying more space in this PIE chart. Similarly modules that have less impact on net system reliability are occupying less space. This new finding can be used in model base testing to reduce the number of test cases and eventually reduce the testing time. Modules having less "relative criticality" values and showing with smaller area in PIE chart should given less focus during testing of such system. At the other hand modules with higher "relative criticality" values and showing with larger area in PIE chart should given more focus during testing.

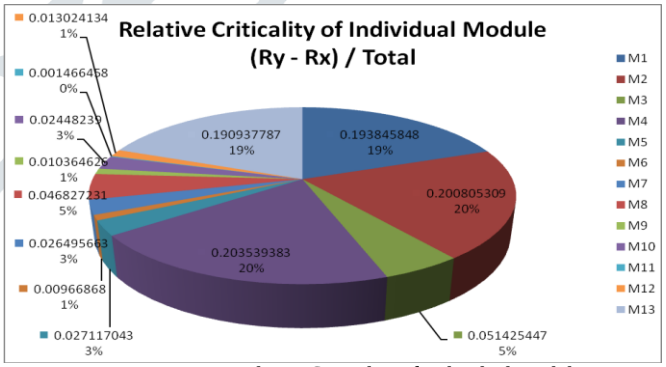


Figure 7.1: Relative Criticality of Individual Module

VIII. *RESULT AND DISCUSSION*

In this work we have calculated the Software Reliability of Library Management System module of SHARP software. For calculating the Software Reliability using User Oriented Software Reliability Model we have assumed Reliability value of each component (sub structure of the "Library Management System" is represented by a digraph. That took us to the development of MARKOV Chain model of Library Management System. module) and assigned inter module transitive probabilities based on survey

of software users. Real time values of such parameter had been acquired from end user of Library module, that is library staff of various colleges where SHARP- Library module is installed and being used.

The Experimental results show that not all the modules have similar impact on the net system reliability. There are some trends between the reliability and individual module uses. Some of the modules have high impact on the net system reliability. If testing has to be done in limited time then testing team can emphasis more on such module and test case generation can be done accordingly. If we are able to ensure maximum reliability of such critical modules then net system reliability will be high accordingly. In this way number of test can be reduced and eventually testing time will be reduced.

IX. CONCLUSIONS

In this work, user oriented reliability is computed for Library Management module of SHARP Education ERP system. For this a Markov chain model is prepared for which probability of usage was estimated through survey conducted among prominent users. This model was used for computing overall system reliability which is found to be approximate 76.40%. Further, impact of individual module on overall reliability of the system is computed. This is done by keeping all sub modules reliability constant and varying the reliability of candidate module from lowest (0) to highest (1). The result showed us that 3-4 modules are critical out of 13 total modules. Criticality of this system is approx 30% (4/13), which is align to Petro's rule. In addition to above, this is also observed that highly critical modules have impact on overall system reliability at their worst case as well as at best case. This information can be used to develop a model base testing technique where testing of few module can be skipped. Such module has very less effect on the overall system reliability. The testing of some specific module can be omitted and ensure up to a specific reliability cut off.

REFERENCES

[1] R.S. Pressman, "Software Engineering: A Practitioner's Approach", 6th Ed., TMH.

[2] Ian Somerville, "Software Engineering", 7th Ed., Pearson Education.

[3] Susumu Fujiwara, Gregor V. Bochmann, "Test Selection based on Finite State Model", IEEE Transaction on Software Engineering, Vol. 17, June 1991.

[4] Roger C. Cheung, "A User-Oriented Software Reliability Model", IEEE Transactions on Software Engineering, Vol. SE-6, No. 2, March 1980

[5] Sigrid Eldh, "Software Testing Techniques"

[6] Z. Jelinski and P. B. Moranda, "Software reliability research," Statistica Computer Performance Evaluation, Freiberger, Ed. New York: Academic, 1972, p. 465-484.

[7] James A. Whittaker, "Stochastic Software Testing", Annals of Software Engineering Vol. 4 (1997) 115-131

[8] J. D. Musa, "A theory of software reliability and its application," IEEE Trans. Software Eng., vol. SE-1, pp. 312-327, Sept. 1975.

[9] M.Prasanna S.N. Sivanandam R.Venkatesan R.Sundarrajan, "A Survey On Automatic Test Case Generation", Academic Open Internet Journal, Volume 15, 2005

[10] James M. Clarke, "Automated Test Generation from a Behavioral Model", Software Quality Week Conference, May 1998

[11] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton B. M. Horowitz, "Model-Based Testing in Practice", Bellcore

[12] C. V. Ramamoorthy, R. C. Cheung, and K. H. Kim, "Reliability and integrity of large computer programs," Comput. Syst. Rel., Infotech State of the Art Rep. 20, 1974.

[13] Ibrahim K. El-Far and James A. Whittaker, "Model-based Software Testing", Encyclopedia on Software Engineering (edited by J.J. Marciniak), Wiley, 2001

[14] Mark Utting, "Position Paper: Model-Based Testing"

[15] Larry Apfelbaum, John Doyle, "Model Based Testing", Software Quality Week Conference in May, 1997

[16] Ji Zhang, Betty H.C. Cheng, "Model-Based Development of Dynamically Adaptive Software",

[17] Manfred Broy, Bengt Jonsson, "Model-Based Testing of Reactive Systems", Springer-Verlag Berlin Heidelberg 2005

[18] B. Littlewood, "How to measure software reliability and how not to. . . ," in Proc. 3rd Int. Conf. Software Eng., May 1978, pp. 37-45.

[19] Stacy Prowell, "State of the Art of Model-Based Testing with Markov Chain Usage Models"

[20] Ydo Wexler & Dan Geiger, "Markov Chain Tutorials".

[21] Kenneth M. Hanson, "Tutorial on Markov Chain Monte Carlo", Bayesian and MaxEnt Workshop, LA-UR-05-5680

[22] D. J. Hatfield, "Experiments of page size, program access, patterns and virtual memory performance," IBM J. Res. Develop., pp. 58-66, Jan. 1972.

[23] Andrew W. Moore, "Markov Systems, Markov Decision Processes, and Dynamic Programming"

[24] C. V. Ramamoorthy, "The analytic design of a dynamic look ahead and program segmenting system for multiprogrammed computers," in Proc. Ass. Comput. Mach. Nat. Conf., 1966, pp. 229-239.

[25] Md. Shazzad Hosain, Md. Shamsul Alam, "Software Reliability Using Markov Chain Usage Model", 3rd International Conference on Electrical & Computer Engineering ICECE 2004

[26] J. G. Kemeny, J. L. Snell, G. L. Thompson, "Introduction to Finite Mathematics", 3rd ed. (Englewood Cli®s, NJ: Prentice-Hall, 1974).

[27] T. B. Pinkerton, "Program behavior and control in virtual storage computer systems," Ass. Comput. Mach., Univ. Michigan, Ann Arbor, Tech. Rep. 4, 1968.

[28] D. J. Hatfield and J. Gerald, "Program restructuring for virtual memory," IBMSyst. J., vol. 10, no. 3, pp. 168-192, 1971.