Model based Testing for Software Systems: An Application of Markov Modulated Markov Process

Abhinav Kashyap, Thomas Holzer, Shahryar Sarkani, Tim Eveleigh
Department of Engineering Management and Systems Engineering
The George Washington University
1776 G Street, NW, Washington DC, USA-20052

ABSTRACT

Software systems require the validation of design features through regression testing. Two primary challenges in system validation are ensuring that test suites reflect actual system usage, and managingthe test suite size to keep testing costs low while keeping testing results meaningful.To create a test environment that is close to actual system usage, we propose using Markov chains to create system behavioral models from available system usage data.Knowing that certain factors are not captured in system usage, we will use the Markov Modulated Markov Process to model hidden processes. The models are used to create test plans that employ a unique, likelihood-based, test prioritization scheme. The proposed methodology not only provides a stochastic modeling framework for software systems, but also considerably improves the coverage factor of generated test suites. This paper also presents a real-world web application case study to demonstrate the capabilities of the proposed system validation methodology.

General Terms

Software testing, System Validation, Test Case Prioritization

Keywords

System Validation; State Transition Diagram, Model-based Testing; Markov chains

1. INTRODUCTION

Software testing continues to be an expensive yet integral part of the software engineering development life cycle. Testing enables the developer to verify the product and provides the means for the end user to validate the system according to their expectations. Traditional software engineering relies on system “shall” requirement statements for deriving test sequences. However, large, complex software systems can end up having unmanageably large numbers of test sequences, and the high cost of performing such tests can make it unreasonable to run them. An alternative approach is model-based testing, which greatly reduces the complexities associated with the system development life cycle.

The choice of a software modeling scheme depends on the type of testing to be performed on the system. Typically, testing activities are performed throughout the design and deployment phases of a software project. Although there are multipletypes of testing techniques forcapturingthe various aspects of system functionality and failure modes[1], the primary goal of all software testing is to improve the quality of the designed system by uncovering as many faults as possible within the allocated time.[2–5].

The research question that we address in this study is whether model-based engineering activities used in system validation improve overall system reliability. One problem with existing systemtest plans is that they are large, ad-hoc, and manually generated thus generating a number of usage paths and scenarios that do not accurately reflect how a system is used in real time. A gap that we address here is that up till now, no one has attempted to use 'likelihood of a test plan representing actual usage' as a criterion in test suite reduction or test plan generation techniques. We attempt to fill this gap.

Figure 1 presents the framework used for ourresearch. First, an existing system usage data set is identified and a system behavioral model is created using Markov Chains.The generated model is used to create test plans and prioritize them. This paper presents a functional testing or “black box testing” methodology that relies on modeling of available system usage data. We also present acomparative study to help explainthe effectiveness of the proposed solution.

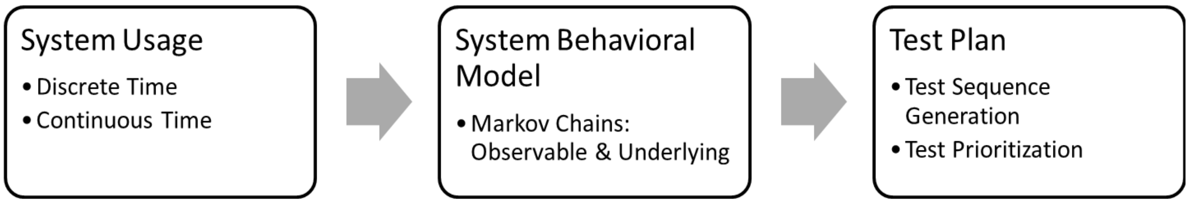

Figure 1: Research Framework

2. RELATED WORK

The use of models specifically for system testing has made huge advances in the last few decades. As described by Dalal et al.[5], model-based testing depends upon three key technologies:

- The notation for modeling,
- The test generation algorithm and
- The tools for supporting infrastructure.

There are a number of standard notations for describing a system model. These notations are categorized by [6] and [7] into state-based (Post/Pre), data-flow based, function-based, operational, transition-based, and stochastic notations. The approach we propose is influenced by the transition-based and stochastic approaches described in the literature.

2.1 State and Flow-based model notations

The state-based notation model describes the internal system state as a collection of variables using a post and pre condition definition. The use of model-based specification languages, such as Z and VDM, to derive formal specifications forsystems and for test-case generation has been extensively researched [8], [9]. T-Vec is another language used for state-based functional specifications of the system [10]. The paper by Zweben et al. [11] was one of the earliest to propose the use of flow graphs as a testing technique. The technique presented in [11] used data-flow and control-flow to represent the system functions as nodes (states) of the graphs and applied them to abstract data types. A similar approach was proposed by Offutt [12] using specification graphs. Specification graphs differ from flow graphs in thatthe graphsrepresent the behavioral states in the system instead of functions.

2.2 Transition-based model notations

Transition-based models use node and arc-like graphical representations to describe the transition between various system states. Finite State Machines (FSM) are widely used for modeling the system state transitions. In [13], the authors used FSM to model the system behavior and a model-based test generator to derive test suites. State-charts are another transition-based approach for modeling a system. State-charts are considered an extension of FSM with several augmented concepts for complex systems specification modeling. In [14], the authors present a method for the selection of test sequences from state-charts and for enabling requirement verification. Pretschneret. al.[15] alsopresented a test-generation tool using state transition diagrams to capture system behavior. Activity diagrams have also been used for automated test generation. In [16], the authors create UML activity diagrams for a JAVA program and then generate test cases using an adequacy criterion.

2.3 Stochastic model notations

Stochastic models define the system as a probabilistic model of events. Markov chains are an example of a stochastic approach. They are used to model the expected usage of a system. Markov chains are mathematical representation of transition between finite states of a system. The transitions are characterized by a random process which has a property of memorylessness i.e. the next state only depends upon next state.Whittaker and Thomason [17] describe an approach using Markov chains to model the behavioral functional diagram of a system. In [18], the authors present a similar technique to generate test cases using Markov chains from operational profile data. The authors of [19]provided a framework for usage-based statistical testing. It uses the frequency count of interactions between the system environment and the target system. In [20] the authors present the use of concept analysis to cluster the user sessions/test cases thereby reducing the test suite for web applications. The approach we present in this paper is different from [20] in that we created a model that can be reused in system regression testing or in system upgrades.

The test-generation algorithm is another key element in model-based testing as described by Dalal et al. [5]. In recent years, the focus has shifted from manual methods to automatic test-case generation. The goal of automation is to generalize the techniques and develop automated tools for supporting the testing infrastructure, including the expected outcomes. Regression testing, for example requires a test-generation approach that can regenerate the test cases in accordance with changing requirements. The ability to prioritize test cases for some goal, such as increasing the rate of fault detection is another highly desirable feature for an automated test-case generator. Numerous prioritization techniques have been described in the reference literature[21], [22]. In [21] the authors present a comparative case study using eight C Programs and various prioritization techniques, such as branch coverage and statement coverage. The techniques presented are code based, i.e., the test cases include coverage of code elements. Large systems can require thousands of lines of code to be tested and may consist of millions of interactions between the objects and different user profiles, making it difficult to apply code-based techniques. The approach suggested by [20] differs from code-based approaches in that it generates usage statistics obtained through operational profiles to generate and prioritize the test cases. Our approach relies on modeling of the available system usage data, but it differs from [20] in that we created a model that can be reused in system regression testing or in system upgrades.Wong et al.[23] describe an algorithm based on increased cost per additional coverage of the software code. In [24] the authors present a comparison of a greedy algorithm, a genetic algorithm, and a hill-climbing algorithm for test-case prioritization. These algorithms are heuristic search algorithms that find an optimal solution in a given search space.

3. MODELING A SOFTWARE SYSTEM

Modeling a software system requires mapping the functionality that the software offers to the programmed objects. Consider the example of an Automated Teller Machine (ATM) software system, as shown in Figure 2.

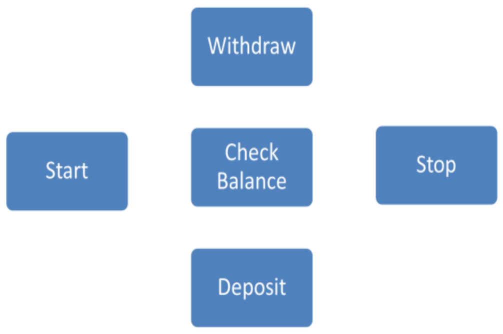

Figure 2: Functions of an ATM software system

An ATM can be used to achieve various goals – for example, to check the current balance, to withdraw money, to deposit money or to transfer money. When simulating a software system using FSM, various system functionalities can be described in the form of stimuli to the system and responses in the form of state progressions. To keep this analysis simple, we can demonstrate the system functionality of the ATM using five states: (a) Start; (b) Stop; (c) Withdraw; (d) Deposit; and (e) Check balance. It is expected that this system would always start in the Start state and end in the Stop state. There are multiplepaths that the system can follow between these terminal states. One straightforward way to describe the system functionality would be to describe each and every possible path under various scenarios, as shown in Figure 3. This methodology is comprehensive but exhaustive, and it could become quite complicated as the system functionality is developed.

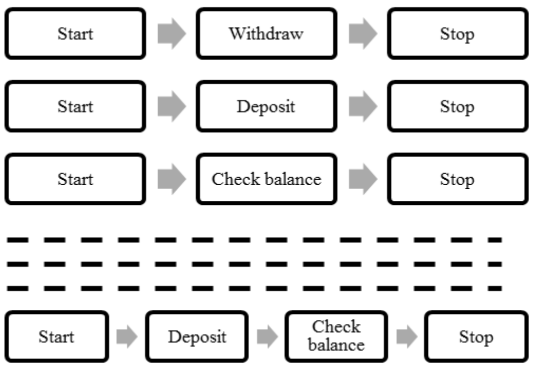

Figure 3: Scenarios for an ATM software system

Another method for capturing all the system functions is to describe the functions of the system in the form of a state transition model. Figure 4 shows a model of the system with various possible paths. By using the state transition model, we reduce the complexity of the system. The model can be used togenerate as many scenarios as required for deriving various system functionalities.However, amajor issue with model-based engineering is that the created model may be as complicated as traditional methods, because model based techniques can generate a large number of possible scenarios. Although this could help in defining the system behavior in a comprehensive manner, it might not be useful for large systems where the system offers an unmanageable number of functions/states.

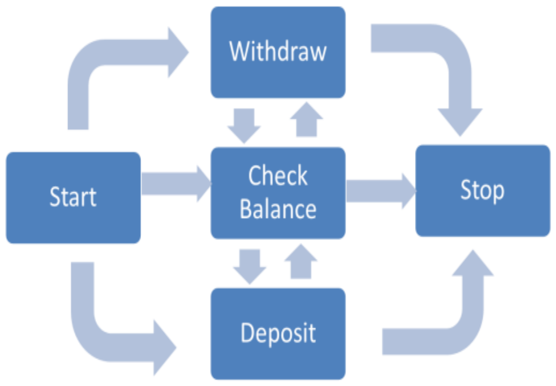

Figure 4: Flow-paths for an ATM software system

Going back to the ATM example, the system offers a total of only 3functions, yet the system can be used in 28 different ways. Some of the possible scenarios would occur only rarely in actual system usage and some would never occur.The model would need to be modified to be able to capture the frequency response of a particular functionality. As described in Section 2, Markov chains have been widely used for capturing the stochastic behavior of systems. Figure 5 shows a Markov chain-based model of the ATM software system. With probabilistic information, the system can be described in a more analytical mannerby defining state progression in terms of likelihood. For example, referencing to figure 5 a customer is more likely to check the balance after withdrawal than to proceed directly to exit the system. The model can be further improved by modeling the probabilities in continuous time. The state progression in continuous time would be modeled using continuous time Markov chains (CTMC). When modeling the system using CTMC, instead of assigning a discrete probability, we assign a transition rate. For example, the ATM software system would have a transition rate of 5 seconds when going from start to deposit and a transition rate of 9seconds going from start to stop. In CTMC, the transition rates are modeled using exponential random variables and the transitions are modeled using Markov chains.

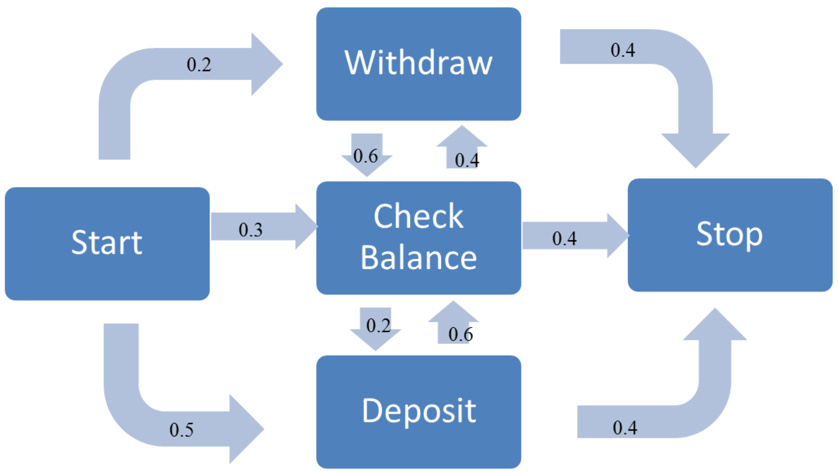

Figure 5: ATM software system state transitions modeled using discrete time Markov chains

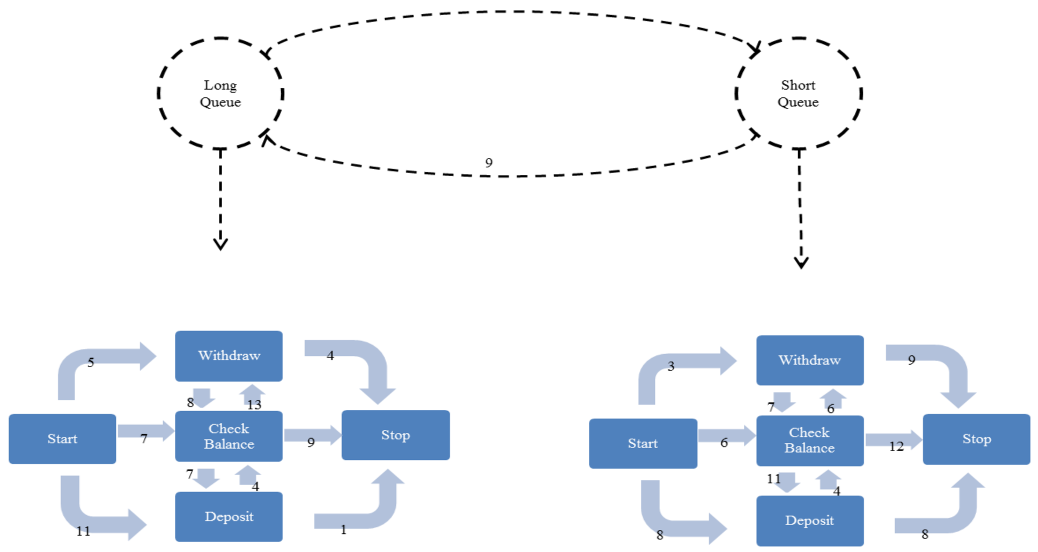

Figure 6: ATM software system state transitions and customer queue lengths modeled as MMMP

3.1 Markov Modulated Markov Process

A common problem in modeling real world processes is that the output signal from a system gets distorted due to the change in environmental conditions affecting the source. In signal processing literature, Hidden Markov Models (HMM) have been used for modeling the signals in learning about the sources where the overall system behavior is described by a doubly stochastic process [25]. In this paper, software systems which have doubly stochastic characteristics are modeled as Markov Modulated Markov Processes (MMMP). An MMMP is a process where the system has two continuous time processes: 1) the observable process, visible in the system usage data, and 2) the underlying process, not visible but directly affecting the transition rates of the observable process.

Again referring to theexample of an ATM software system, system transitions are modeled as MMMPs, as shown in Figure 6. The underlying process here is the type of queue, as it transitions from one state to another. Here the queue assumes two distinct states, a longer queue and a shorter queue. Depending upon the queue length, the user would have a different response while using the system. The underlying process itself is modeled as a CTMC. For each underlying process, the ATM system would have a set of transition probabilities. Taking the underlying process transition probabilities together with the observable process transition probabilities, we can define the complete system behavior in a stochastic manner. The generator for an MMMP is given by Equation 1:

$$\Phi = \{Q, G_1, G_2 \ldots G_r\} \qquad (1)$$

Here, Q represents the observable process transition probabilities and G_1; $G2$ ...Gr represents the transition probability matrix for the underlying process, where r is the number of underlying states.

3.2 Parameter Estimation

Creating the stochastic model of the software system using MMMP requires an estimate of the probabilities associated with both the observable process and the underlying process. One of the key features of this research is that we use the available system usage data to estimate these probabilities.Using the available system usage data reduces the gap between actual system behavior and the created system model. We created anMMMP model for the system by estimating the generator, Φ. We used the EM algorithm developed by Roberts and Ephraim [26]. In our previous work [27] we reviewedthe essential equations for using this algorithm. Readers are encouraged to review the equations and to refer to [27] for a more detailed review of the algorithm. To keep the discussion system-oriented, a pseudo code of the algorithm is provided in Figure 7.

```
1. Initialize parameters {Q, G_1, G_2...G_r} with $\Phi^0=\{Q^0, G^0_1, G^0_2 \ldots G^0_r\}$.

2. Compute complete log likelihood $L_{cl}$ of $\Phi_l$ given the observation data $\chi$.

3. E Step. Using forward–backward recursion, compute mean estimates of:
      a. Dwell times in underlying state i where $i=1,2,3\ldots r$.
      b. Number of jumps from underlying state i to state j, where $i,j=1,2\ldots r$ and $i\neq j$.
      c. Dwell times in observable states $l=1,2,3\ldots N$ given underlying state i.
      d. Number of jumps from observable state l to state n given underlying state I.

4. M Step: Maximize the estimated probabilities and compute new estimate of $\Phi_{l+1}$
      a. Compute new estimates of Q by dividing 3b with 3a.
      b. Compute new estimates of $G_1, G_2 \ldots Gr$ by dividing 3d with 3c
      .
5. Repeat 2 to 4 until convergence, which could be $L_{cl+}L_{CL+1}/L_{cl}$<desired limit.
```

Figure 7: Pseudo code for EM algorithm for MMMP [27]

The first step is to initialize the parameters that need to be estimated. Next, the log likelihood is estimated using all the available observable data. The third step is the expectation step (E-Step), in which the forward-backward recursion procedure computes the dwell times and the number of jumps associated with the observable and underlying processes. In the maximization step (M-Step), new estimates for the transition probabilities are generated. The convergence criterion used for the EM algorithm is the improvement in the likelihood function between the previous and the new estimates.

3.3 Likelihood-based Prioritization

The test sequence prioritization problem as defined in[21] is as follows:

Given: T, a test suite; PT, the set of permutations of T; f, a function from PT to the real numbers.

Problem: Find T'∈PT such that,

$(\forall T')(T'' \in PT)(T'' \neq T')\ [f(T') \geq f(T'')]$.

Here, *PT* represents the set of all possible prioritizations(Orderings) of *T*, and *f* is a function that, applied to any such ordering, yields an award value for that ordering. In the proposed methodology, we used a likelihood-based approach for prioritizing the test sequences. The likelihood-based prioritization scheme works on the principle of choosing a subset of test sequences from the set of test sequences.Likelihood is used asan objective function to solve the prioritization problem. Figure 8 summarizes the prioritization scheme used in this study.

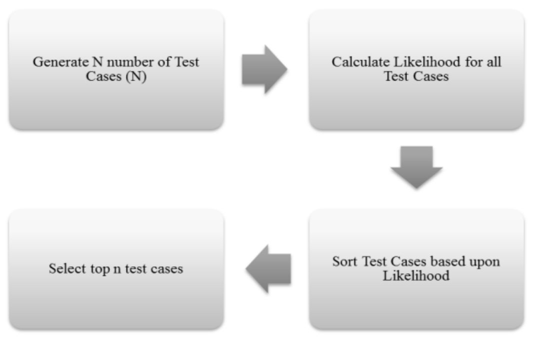
Generate N number of Test Cases (N)
Calculate Likelihood for all Test Cases
Sort Test Cases based upon Likelihood
Select top n test cases

Figure 8: Proposed test sequence prioritization scheme

A large number of test sequences are generated from the models, which are in the form of CTMCs. The likelihood function is calculatedfor each transition sequence. The likelihood function for MMMP is given in the equation below.

$$P(\chi,S;\Phi)=\pi_{s_1}\left\{\prod_{k=1}^{M} q_{s_k} e^{-q_{s_k}\Delta\tau_k}\frac{q_{s_k s_{k+1}}}{q_{s_k}}\right\} e^{-q_{s_{M+1}}\Delta\tau_{M+1}}$$

$$\times \nu_{x_1}^{s_1}\prod_{k=1}^{M+1}\left\{\prod_{n=1}^{N} g_{x_n}^{S_k} e^{-g_{x_n}^{S_k}\Delta T_n}\frac{g_{x_n x_{n+1}}^{s_k}}{g_{x_n}^{s_k}}\right\} e^{-g_{x_{N+1}}^{s_k}\Delta T_{N+1}} \qquad (2)$$

The derivation and explanation of the complete likelihood equation can be found in our previous work [27]. In the next step, the test sequences are prioritized by sorting the test sequences, based upon likelihood. From the prioritized test sequences, the top *'n'* test sequences are chosen based on testing requirements such as the estimated cost or the time allotted for the testing activities.

4. ILLUSTRATIVE CASE STUDY

The system behavior model neededto be created from the available system usage data. Since the system behavior is modeled using the time spent in a particular state, we reviewed data sets that included time aspects of system usage. Most software systems change states in the time domain, and such data can easily be recorded. Modeling of a software system has many possible uses – for instance, the model can be used in design improvement activities, in testing of current features, in extracting user demographic information, in testing browser compatibility, or in improving the user experience. Like software systems, web-based applications also collect time-based system data, which is reflected in the click stream data. To demonstrate our proposed methodology for a system behavior model and test-sequence generation, we chose WebKDD cup 2000 website click stream data[28]. WebKDD cup click stream data consist of the user sessions for a commercial retail website. This final set of click stream data was taken from the web server between January 30, 2000 and March 31, 2000. The click stream data are from a retail website which closed its operation in April 2000. There were 777,480 records, each with 217 fields.

4.1 Data Description

Each record of the clickstream data has information about a page visited by a user. There could be multiple page visits from a user in the usage data. The length of a user session is derived from the number of pages visited and the time spent on each page. There is lot more information available from each record – for example, session ID, cookie ID, and first page visited. For our case study, the fields required from the data set weresystem states, i.e., the web pages visited by the user and the dwell time in that particular state. We reorganized the click stream data by calculating dwell times for each state, which we did by subtracting the time between clicks for each successive web page visited. We also separated one session from another by using session ID as the criterion for identifying a unique session.To perform the modeling at an abstract level, all the web pages were categorized into one of the four categories. Table 1 shows the categorization of web page types.

Table 1: Categorization for web application case study

Category	Type
1	Account Activity, Billing
2	Company Information, News
3	Departments, Browse
4	Home Page, Start, Returns, Replenish

After the web pages were categorized, all user sessions were transformed into state sequences, which consist of system states and dwell times.

4.2 User Behavior and System Interaction

To describe the underlying user behavior, we used user states as defined in [29]. The hidden states are purchase and non-purchase states. There are other possible ways of categorizing user behaviors. For example, we might have used purchase, non-purchase and casual browsing. The number of categorization levels used really depended upon the level of analysis required.

4.3 Web Application MMMP Model

Before modeling the web application usage data using CTMCs, we validated the dwell times for each state to haveexponential distribution. Once the data had been formatted – i.e., the web page data had been transformed to states and validated for appropriate random variables – we created two estimated probability distribution matrices, one for underlying processes and one for observable processes. The first probability matrix consisted of the estimated transition rates for the underlying process. For this case study, we used an equal probability distribution as the initial probability distribution for a user going from the purchase state to the non-purchase state. The second probability matrix was the probability distribution for the observed process. These probabilities together formed the generator *Φ*, as described in Equation 1. Next, we estimated new parameters using the EM algorithm, as described in Section 3.2. Table 2 and Table 3 show the initial and the final estimatesof the transition rates for the underlying and observable states.

Table 2: Initial and final estimates of transition rates for the Underlying Continuous Time Markov Process

	Initial Estimate		**Final Estimate**	
	Purchase	**Non-Purchase**	**Purchase**	**Non-Purchase**
Purchase	-5	5	-9	9
Non-Purchase	1	-1	5	-5

Table 3: Initial and final estimates of transition rates for the Observable Continuous Time Markov Process

	Initial Estimate				**Final Estimate**			
G1	-233	82	69	82	-169.12	4.37	52.61	112.13
	96	-199	13	88	32.96	-114.39	13.22	68.19
	19	11	-105	72	26.83	3.36	-74.87	44.67
	13	78	68	-163	41.66	9.24	17.15	-68.06
	Initial Estimate				**Final Estimate**			
G2	-164	90	40	33	-34.08	3.25	21.55	9.28
	95	-178	22	59	9.59	-35.48	12.30	13.59
	20	91	-135	21	8.52	1.46	-13.07	3.07
	36	45	19	-104	10.23	5.53	18.30	-34.08

4.4 Test Sequence Prioritization and Stopping Criterion

Using the likelihood-based methodology as described in Section 3.3, we created and then prioritized one thousand test sequencesfrom the estimated probabilities.It is important to note that since we used a sorting scheme to prioritize test sequences, the computational complexity of the algorithm reaches *O (n log n)*.We could improve the results by using an advanced sorting algorithm such as smooth sort, strand sort or cycle sort. This area of our research was also a good candidate for implementing heuristic search algorithms to identify test sequences based on an objective function, for which we could use likelihood, as well. Due to the nature of sorting, it is expected that as the number of test sequences increases, the benefit of the prioritization scheme diminishes. Since running the whole test suite could be a costly endeavor, the tester could use the likelihood vs. number of test sequences graph to choose a stopping criterion, based on the level of testing required. For our analysis, the final test suite was limited to the top 100 test sequences.

4.5 Statistical Comparative Analysis

Ourprimary research question in this work was whether the proposed prioritization scheme would increase the likelihood of a test plan being a better representative of real-world usage compared to the ad-hoc random generation of a test plan. For this problem, the research hypotheses are as follows:

Null Hypothesis H_0: *Likelihood* $_{Ad\text{-}hoc}$= *Likelihood* $_{Model}$

Alternative HypothesisH_1: *Likelihood* $_{Ad\text{-}hoc}\leq$ *Likelihood* $_{Model}$

To investigate this hypothesis, we compared the likelihoods of 25 test plans for the ad-hoc test plan generation technique with the likelihoods of 25 test plans for the proposed model-based test plan generation and prioritization technique. For the ad-hoc test plan, we created test plans each consisting of 100 test sequences randomly selected from WebKDD cup web navigation data. Using the proposed methodology, we generated the other set of test plans, with each of them also consisting of 100 test sequences. In the next step, we calculated the total test-plan likelihood for both sets of test plans. Total likelihood was the product of all likelihoods of test sequences within a test plan. Given that the likelihood data has an acceptable fit with normal distribution, we used a two-tailed student-t test to test the null hypothesis at a significant level of *alpha = 0.05*. Table 4 summarizes the results of the hypothesis testing.

Table 4: Hypothesis testing for likelihood-based test sequence prioritization scheme

	Prioritized	Un-prioritized
Mean	2794.787	448.7143
Variance	18854.96	372480.7
Observations	25	25
Pearson Correlation	0.003256	
Hypothesized Mean Difference	0	
Df	24	
t Stat	18.76462	
P(T<=t) one-tail	3.79E-16	
t Critical one-tail	1.710882	
P(T<=t) two-tail	7.57E-16	
t Critical two-tail	2.063899	

5. CONCLUSION

The previous section provided an analytical comparison based on likelihood function. Since the p-value of 7.57E-16 for the two-tailed test is less than alpha of 0.05, the null hypothesis H_0 is rejected and the alternate hypothesis H_1 is accepted. Additionally it is clear from the two means of likelihood functions is that the proposed methodology provides a higher coverage of the most likely usage scenarios.

The methodology we have presented has three contributions to system analysis and testing:

a) A system modeling approach that captures the stochastic nature of system behavior. Not only do we capture the observable progression of a system, but we also capture an underlying phenomenon that would affect the system behavior.
b) A test-sequence generation scheme for the Markov Modulated Markov process model. The presented scheme could be used for automated test-case generation. There are numerous applications for automated test-case generation – for example, during system upgrades, commercial off-the-shelf analysis, system maintenance, or legacy system comparison.
c) A test-sequence prioritization scheme that uses likelihood as an objective. The presented scheme exploits the stochastic nature of the system model in improving the coverage of the test suite.

Another area of this research explored the impact of the length of the test suite on likelihood function. In Figure 9, a cumulative likelihood versus number of test sequences for the web application case study has been plotted. Such data analysiswould provide a direct feedback to the system validation team about the time and cost required to run the extent ofthe test suite.

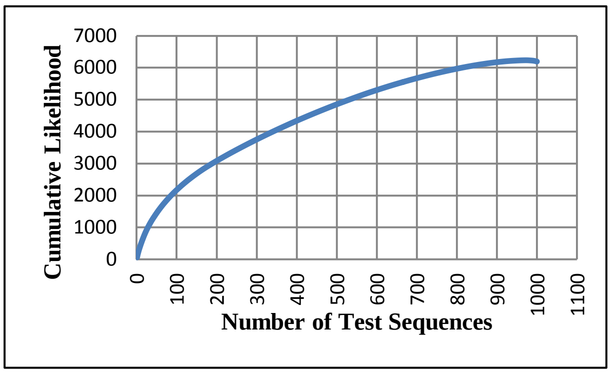

Figure 7: Cumulative Likelihood vs. Number of test sequences

Other researchers have suggestedvarious objective functions as a stopping criterion for test suite prioritization such asaverage percentage of fault detected [30].Likeaverage percentage of fault detected , a likelihood function could be used as a stopping criterion. For example, a tester could stop the proposed test suite generation at 100 test sequences for a cumulative likelihood value of 2000.

6. LIMITATIONS AND FUTURE RESEARCH

This study uses Markov chains to explore the time domain progression of a system. Not all software systems have usage data consisting of time domain data, and this imposes a limitation on the applicability of the technique we havepresented to some systems. In cases where time domain data are unavailable, the system would need to be modeled using Discrete Markov Chains and Hidden Markov Models,as required.

A second limitation of this study is that the presented case study does not validate the proposed methodology by testing the fault detectioncapabilities. To understand the fault detection capabilities of the methodology, we would need to develop asystem where a known number of faults could be seeded and tested with the generated test plans.

Thirdly,techniquewe have presented models the underlying process in categories based upon user behavior and preferences. There may be other ways of performing this categorization, for example, on the basis of environmental conditions or of the economic condition of system users. We would need amore comprehensive study to capture other underlying process models.

Lastly, the proposed model could be greatly affected by the social-cultural aspect of the system environment. Building a more comprehensive framework for the proposed methodology might be considered for future work.

7. ACKNOWLEDGMENTS

The authors would like to thank Blue Martini Software for making the WebKDD cup 2000 data available for this research. We would also like to thank Dr. William J. J. Roberts for his productive discussions and for providing valuable suggestions for improving this research.

8. REFERENCES

[1] L. Huang and B. Boehm, “How Much Software Quality Investment Is Enough: A Value-Based Approach,” *Software, IEEE*, vol. 23, no. 5, pp. 88 –95, Oct. 2006.

[2] K.-C. Chiu, J.-W. Ho, and Y.-S. Huang, “Bayesian updating of optimal release time for software systems,” *Software Quality Control*, vol. 17, no. 1, pp. 99–120, Mar. 2009.

[3] Y. F. Li, M. Xie, and T. N. Goh, “A study of project selection and feature weighting for analogy based software cost estimation,” *Journal of Systems and Software*, vol. 82, no. 2, pp. 241–252, Feb. 2009.

[4] S.M.K Quadri and Sheikh Umar Farooq,” Software Testing-Goals, Principles and Limitations,” International Journal of Computer Applications, Volume 6-No.9, September 2010.

[5] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, “Model-based testing in practice,” in *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, 1999, pp. 285 –294.

[6] M. Utting, A. Pretschner, and B. Legeard, “A Taxonomy of model-based testing,” Apr. 2006.

[7] A. van Lamsweerde, “Formal specification: a roadmap,” in *Proceedings of the Conference on The Future of Software Engineering*, New York, NY, USA, 2000, pp. 147–159.

[8] J. Dick and A. Faivre, “Automating the Generation and Sequencing of Test Cases from Model-Based Specifications,” in *Proceedings of the First International*

Symposium of Formal Methods Europe on Industrial-Strength Formal Methods, London, UK, UK, 1993, pp. 268–284.

[9] R. M. Hierons, "Testing from a Z Specification," *Software Testing, Verification and Reliability*, vol. 7, no. 1, pp. 19–33, Mar. 1997.

[10] M. R. Blackburn and R. D. Busser, "T-VEC: A tool for developing critical systems," in *In Proceedings of the 1996 Annual Conference on Computer Assurance (COMPASS 96*, 1996, pp. 237–249.

[11] S. Zweben, W. Heym, and J. Kimmich, "Systematic Testing of Data Abstractions Based on Software Specifications," *Software Testing, Verification, and Reliability*, vol. 1, no. 4, pp. 39–55, 1992.

[12] J. Offutt, S. Liu, A. Abdurazik, and P. Ammann, "Generating test data from state-based specifications," *The Journal of Software Testing, Verification and Reliability*, vol. 13, pp. 25–53, 2003.

[13] E. Farchi, A. Hartman, and S. S. Pinter, "Using a model-based test generator to test for standard conformance," *IBM Syst. J.*, vol. 41, no. 1, pp. 89–110, Jan. 2002.

[14] H. S. Hong, Y. G. Kim, S. D. Cha, D. H. Bae, and H. Ural, "A test sequence selection method for statecharts," *Software Testing, Verification and Reliability*, vol. 10, no. 4, pp. 203–227, Dec. 2000.

[15] A. Pretschner, O. Slotosch, E. Aiglstorfer, and S. Kriebel, "Model-based testing for real," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 5, pp. 140–157, Mar. 2004.

[16] M. Chen, X. Qiu, W. Xu, L. Wang, J. Zhao, and X. Li, "UML Activity Diagram-Based Automatic Test Case Generation For Java Programs," *The Computer Journal*, vol. 52, no. 5, pp. 545 –556, 2009.

[17] J. Whittaker and M. Thomason, "A Markov Chain Model for Statistical Software Testing," *IEEE Trans. Softw. Eng.*, vol. 20, no. 10, pp. 812–824, Oct. 1994.

[18] A. Avritzer and E. J. Weyuker, "The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software," *IEEE Trans. Softw. Eng.*, vol. 21, no. 9, pp. 705–716, Sep. 1995.

[19] B. Regnell, "Towards integration of use case modelling and usage-based testing," *Journal of Systems and Software*, vol. 50, pp. 117–130, Feb. 2000.

[20] S. Sampath, S. Sprenkle, E. Gibson, L. Pollock, and A. S. Greenwald, "Applying Concept Analysis to User-Session-Based Testing of Web Applications," *Software Engineering, IEEE Transactions on*, vol. 33, no. 10, pp. 643 –658, Oct. 2007.

[21] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test Case Prioritization: A Family of Empirical Studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Feb. 2002.

[22] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," in *Proceedings of the International Symposium on Software Testing and Analysis - ISSTA '00*, Portland, Oregon, United States, 2000, pp. 102–112.

[23] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal, "A study of effective regression testing in practice," in *PROCEEDINGS The Eighth International Symposium On Software Reliability Engineering*, 1997, pp. 264 – 274.

[24] Z. Li, M. Harman, and R. M. Hierons, "Search Algorithms for Regression Test Case Prioritization," *IIEEE Trans. Software Eng.*, vol. 33, no. 4, pp. 225–237, Apr. 2007.

[25] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *PROCEEDINGS OF THE IEEE*, vol. 77, p. 257–286, 1989.

[26] Y. Ephraim and W. J. J. Roberts, "An EM Algorithm for Markov Modulated Markov Processes," *IEEE Trans. Signal Process.*, vol. 57, no. 2, pp. 463–470, Feb. 2009.

[27] A. Kashyap, W. Roberts, S. Sarkani, and T. Mazzuchi, "A Model Driven approach for System Validation," in *IEEE International Systems Conference*, Vancouver, BC, Canada, 2012.

[28] R. Kohavi, C. E. Brodley, B. Frasca, L. Mason, and Z. Zheng, "KDD-Cup 2000 organizers' report: peeling the onion," *SIGKDD Explor. Newsl.*, vol. 2, no. 2, pp. 86–93, Dec. 2000.

[29] A. L. Montgomery, S. Li, K. Srinivasan, and J. C. Liechty, "Modeling Online Browsing and Path Analysis Using Clickstream Data," *Marketing Science*, vol. 23, no. 4, pp. 579–595, 2004.

[30] S. Elbaum, G. Rothermel, S. Kanduri, and A. G. Malishevsky, "Selecting a Cost-Effective Test Case Prioritization Technique," *Software Quality Control*, vol. 12, no. 3, pp. 185–210, Sep. 2004.