An Exploration of Model Based Testing

Vasudha Singh , Subburaj Ramasamy

Abstract— Testing consumes about 50% of the total software development costs. The purpose of testing is to check the correctness of any software created as to whether it is working according to what was expected and what it was supposed to do. Software testing is a process of executing a program or application with the purpose of finding defects. The aim of software testing is not only to find the defects but also to find out the situations that could cause negative impact on the customer. There are many techniques and ways to test the application and software but in this paper we bring out one of the efficient and effective methodologies namely; model based testing (MBT). MBT facilitates generation of effective test cases from the developed model of the software. A model describes the functionality and behavior of the system under test (SUT). This paper introduces model-based testing and gives case study of the same.

Index Terms— Model based testing, Software Development Life Cycle(SDLC), System under test(sut), Test case generation .

.

1 INTRODUCTION

Billions of dollars are lost because of programming errors. They are consequences of poor programming quality. Program testing has one of the essential and significant roles in Software Development Life Cycle [1]. According to an analyst's assessment, on the average about 50-60% of development time is committed for Software testing in Industries.

Software testing and fault detection activities are time consuming and require systematic efforts especially in complex systems. But they are essential for ensuring quality of the project and product. Through manual testing we can find out defects in a software application but it is a time consuming task. Automating the testing tasks may decrease the time and efforts. Once the tests are automated, they can be run quickly and results are generated.

There exist two main methodologies for test case generation namely; Black-Box [2] and White-Box [3] techniques. Black box testing refers to the testing of the system in terms of its behavior. Black box testing involves traversing of various possible inputs and the corresponding possible outputs from them. It does not consider the internals of the system or the process adapted for the functionality. In the case of White-Box testing on the other hand, it is essential to understand internal functionality of the system, internal data structures and logic used in the coding etc.

As the needs of customers are growing rapidly, and so is functionality, one needs to develop complex software system. More complex the system is, more efforts are required to test it. A small change in the system may expand the time and efforts of testing the system. Similarly with the increasing demands for software products, user expects to acquire a software which is good in performance, more dependable, productive and more reliable. The competition demands the developer to deliver the software meeting all the user's requirements within scheduled time and within budget. It is a tight rope walking for the software development organization. There comes the need of test automation that may diminish the efforts and time as well as the cost.

The traditional testing of the system is carried out by checking the usefulness and working of the system manually. But in automation testing, the testing process involves usage of tools and scripts to generate test cases and perform testing automatically. One evolving automation technique is Model Based Testing[4]. This process involves creation of a model that describes the test cases, test data and the system under test execution environment. Test cases are generated from the model. The main advantage of using MBT is that the time required for modeling the behavior of the system is less than manual test case writing and execution. Also it generates wide range of test cases which more often may not be possible to be derived manually.

MBT [4] is the generation of software test procedures, using models of system requirements and behavior. In MBT we create model of system under test (SUT) and generate test cases from the model. It is different from traditional testing because in traditional testing we have to go through

- *Vasudha singh is currently pursuing masters degree program in Information technology in SRM University, Chennai, India, PH-08939236239. E-mail: vasudha.s10@gmail.com*
- *Dr. R. subburaj, professor and consultant in SRM university, dept. of IT, kattankulathur-603203, Chennai area , India. E-mail: subburaj.r@ktr.srmuniv.ac.in*

the code and specifications and then inspect the functionality of a system. It takes much effort and time to understand the code and functionality of the system. In MBT we create model which specify the functionality and behavior of the SUT and from that model we create the test cases. Later on, if any changes are made in the functionality of the SUT then accordingly the model is modified.

In this paper in section 2, we discuss the MBT. In section 3 we discuss different types of models and approaches to testing. In section 4 a case study is given which uses MBT as a testing process. Conclusion is given in section 5. Finally ended with the table having list of some MBT tools.

2 BASIC MBT APPROACH

Models are created from SUT requirements and behavior. Model-based testing (MBT)[4] is the automatic generation of test procedures, using models. MBT requires efforts in building the model. From the model, we can derive test cases for functional testing of the SUT. In order to model an SUT, the internals of SUT need not be visible.

MBT is defined as "software testing in which test cases are derived in whole or in part from a model that describes some (usually functional) aspects of the system under test (SUT). A definition or a general description of a software model [5] is the following:

"Simply put, a model of software is a depiction of its behavior. Behavior can be described in terms of the input sequences accepted by the system, the actions, conditions, and output logic, or the flow of data through the application's modules and routines. In order for a model to be useful for group of testers and for multiple testing tasks, it needs to be taken out of the minds of those who understand what the software is supposed to accomplish and written down in an easily understandable form.. With these properties, the model becomes shareable, reusable, precise description of the system under test."

Model can help to know the behavior of the system and they are simpler than actual software descriptions. The main benefit of MBT is to create wide range of test cases in a short span of time. It is easy to detect defects in modeling at earlier stages based on the test cases that are derived. MBT also facilitates traceability by considering the matching of test cases with the requirements.

MBT helps to establish a systematic way of testing. The main goal of testing is to find faults in SUT. But the number of faults which we find depends upon the way in which the system is modeled. A wrong understanding of requirements may lead to incorrect modeling of the system, as a result it leads to fault.

A model is a halfway presentation of the SUT. A model can also be created from the specifications and requirements. Test sequences are derived from the model. These test sequences are known as a unique test suite. The role of test sequences is to control the system under test, driving it into the different conditions under which it can be tested for conformance with the model. The test oracle observes the progress of the implementation and issues a pass or fail verdict. In some model-based testing situations, models contain enough data to create executable test suites. A general representation of MBT process is given in Fig. 1.

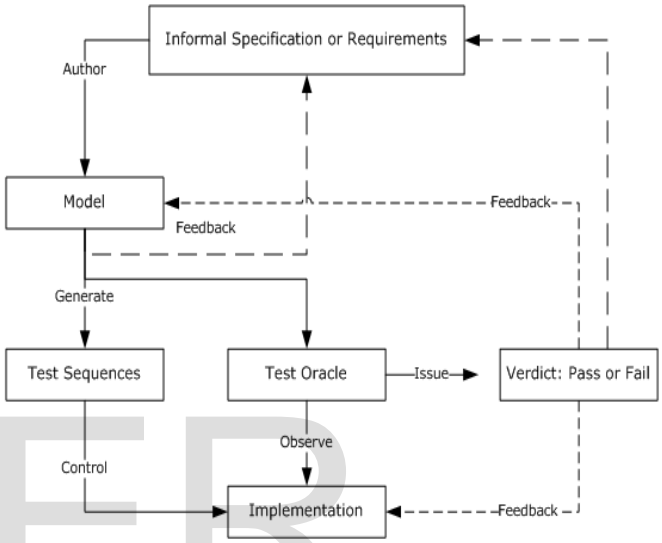

Fig 1. Genral MBT process.

.

To understand the MBT, we take a simple case study of " ROBOT" as an application for testing. For this case study we create FSM model by taking valid combination of inputs as the initial state and consider output as the final state. The ROBOT application has following functionality: It can be turned on and off, can walk, run, raise and lower its arms, turn its head, and talk.

A listing of all the possible events and states associated with the ROBOT application is given in Table 1.

Event	State
turnOn	Activated
turnOff	Deactivated (Idle)
stop	Stopped
walk	Walking
run	Running
raiseLeftArm	LeftArmRaised
lowerLeftArm	LeftArmLowered
lowerLeftArm	LeftArmLowered
raiseRightArm	RightArmRaised
lowerRightArm	RightArmLowered
turnHead	HeadTurned(direction)
speak	Talking(text)

Table 1: list of all events and their states in the ROBOT application

From the requirements captured in Table 1, we can create a model for ROBOT system. The model is shown in Fig. 2. Now we can create the test cases from this model. In test cases, we start from initial state which is shown as filled circle in Fig. 2. The first test case is "turn on" of the ROBOT which will result in ROBOT getting TurnedOn. From there we can generate 5 test cases; talk, run etc.

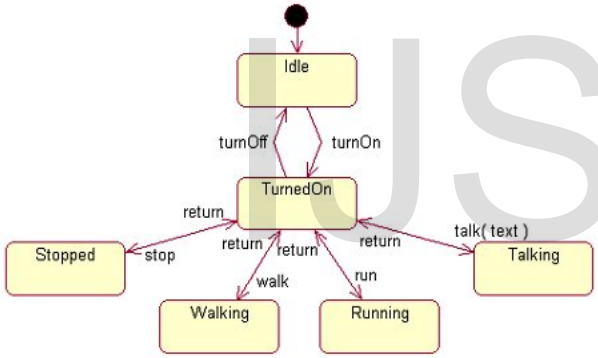

Fig 2 FSM model for the case study.

3 TYPES OF MODELS IN MBT APPROACH

Model of software is the description of functionality and behavior and it can be described in terms of input sequence, action, condition and output logic. Many models are used for software testing, some of which are described below:

3.1 Finite state machine

Finite state machines have been around even before the beginning software engineering. Finite state machines are created for an application, which has a limited number of particular states. Consider a typical testing situation: a tester applies inputs and evaluates the result. The tester then chooses an alternate input depending upon the previous result, and again apply the set of conceivable inputs. At any given time, a tester has a particular set of inputs to apply to the SUT. Such a model is the finite state machine [6].

3.2 Statecharts

State charts[7] are an augmentation of finite state machines that particularly address real time and complex systems. They give a structure to defining state machines in a pecking order, where a solitary state can be "extended" into an alternate "lower-level" state machine. They additionally accommodate simultaneous state machines. Furthermore, the structure of State charts includes outside conditions that influence whether a move happens from a specific state, which as a rule can decrease the size of the model being made. State charts are similar to the most influential type of automata: the Turing machine. Then again, State charts are more down to business while keeping up the same expressive abilities. State charts are easy to understand as compared to finite state machines.

3.3 UML (unified modeling language)

Modeling is the designing of software applications before coding. Modeling is an essential part of large software projects, and helpful to medium and even small projects . UML[8] helps one to specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements. UML model can be either platform-independent or platform-specific.

UML is to models what C or Pascal are to projects – a method for depicting very complex behavior. UML can likewise incorporate different sorts of models inside it, so finite state machines and state charts can get to be segments of the bigger UML

3.4 Markov chain

Markov chains[9] are stochastic models [kemeny & Snell 1976]. A particular class of Markov chains, the discrete-parameter, limited state, time-homogenous, irreducible Markov chain, has been utilized to model the utilization of programming. They are structurally like limited state machines and can be considered probabilistic automata. Their essential worth has been in creating tests. Testing specific frameworks can be found in(Avritzer & Larson 1993)[12] and (agrawal & Whittaker 1993)[13].

3.5 Grammar

A formal grammar[10] is a set of rules for rewriting strings, along with a "start symbol" from which rewriting starts. Therefore, a grammar is usually thought of as a language generator. Grammar has mostly been used to describe the syntax of programming and other input languages. Functionally speaking, different classes of grammars are equivalent to different forms of state machines. Sometimes,

they are much easier and more compact representation for modeling certain systems such as parsers. Although they require some training, they are, thereafter, generally easy to write, review, and maintain. However, they may present some concerns when it comes to generating tests and defining coverage criteria.

In order to demonstrate MBT, we selected a website named "www. Orbitz.com "

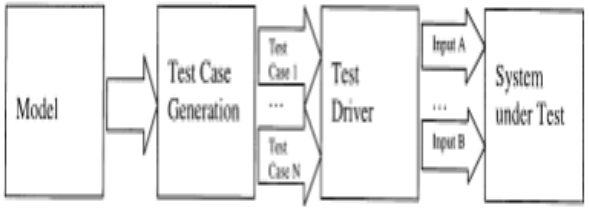

Fig. 3.Process of creating test cases from the model.

As shown in Fig. 3 we create our model for the website. In our case study we have basically six pages like hotels, flights , activities, packages, cars, cruises and for each page there is individual form. we can go from one page to another page by giving valid input. This whole scenario is described in Fig 4 in the form of model . and then test cases are derived in Table 3 and Table 4.

We will use FSM to model and test the web site. The basic FSM terminology is given below:

Formally a finite state machine representing a software system is defined as a tupels (I, S, T, F, L), where

- I is the set of inputs of the system (as opposed to input sequences).
- S is the set of all states of the system.
- T is a function that determines whether a transition occurs when an input is applied to the system in a particular state.
- F is the set of final states the system can end up in when it terminates.
- L is the state into which the software is launched that means initial state

Finite state machine models can be represented as graphs, also called state transition diagrams, with nodes representing states, arcs representing transitions, and arc-labels representing inputs causing the transitions. Usually, the starting and final states are specially marked. Automata can also be represented as matrices, called state transition

Some tools available for MBT are given in Appendix 1.

4. CASE STUDY

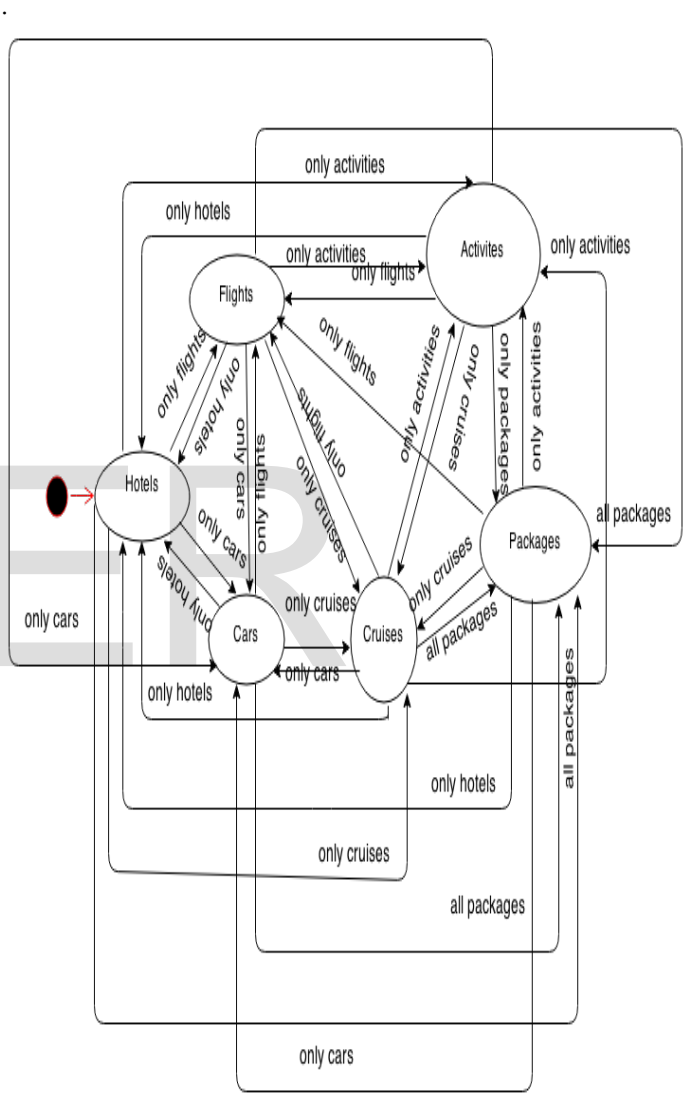

Fig. 4. FSM model of the SUT.

matrices. In FSM methodology models are created by person knowledgeable in the application heuristically(as explained in fig 4) .After creating the model test suits are derived for each page of the application. In a linear organization, start with the
home page as well. We may then test successive Web pages in a column- first or row- first order. In a network organization, begin with the home.

Then test all the web pages that are reachable (via links) from the home page. The test suit for testing the " flight only page " are given below. Similarly create test suites for each pages in our website to check that all the links, buttons forms are working accordingly to the specifications or not. It has been ensure that there is no broken and dead links and for this we create our test cases as given in Table 3.

	A	B	C	D	E	F
1	**Module**	**orbitiz_flight**				
2	**Project**	**www.orbitz.com**				
3	**TestCase**	**TC_1.01**				
4	**Requireme**	**user must be able to receive the flight details**				
5	**Requireme nt ID**	**RD01**				
6	**Condition**	**application must be hosted online**				
7	**Test Data**	**https:www.orbitz.com**				
8	**Brief Descriptio n**	**user must be able to receive the flight details and able to search the lowest fare of the flights of the round-trip option**				
9	**SL No**	**DESCRPTION**	**INPUT**	**EXPECTED**	**ACTUAL**	**STATU**
10		go to the "flight only" page	tick the radio button having option flight only	form coressponding to flight only's form should be op	As expected	pass
11	1	click on the radio button having option round trip	tick the radio button having option round trip	form coressponding to round trip's form should be op	form is opened	pass
12	2	enter the text for "source" city	enter city "lsa"	from city code. Automatically city should come in text	As expected	pass
13	3	tick the option "airport within 80miles"	tick the checkbox	show the airport within 80 miles	As expected	pass
14	4	enter the text for "destination" city	enter destination city "la"	from city code. Automatically city should come in text	As expected	pass
15	5	tick the option "airport within 80miles"	tick the checkbox	show the airport within 80 miles	As expected	pass
16	6	enter the leaving date	enter the date in MM/DD/YYYY format	from date picker date autmatically select mention dat	As expected	pass
17	7	enter the return date	enter the date in MM/DD/YYYY format	from date picker date autmatically select mention dat	As expected	pass
18	8	check the time box	tick the timebox for anytime search of flights	all the flights details of any time should come on selec	As expected	pass
19	9	select persons individually	select person - 1 adult	show the price for 1 person	As expected	pass
20	10	click on search button	click on the search button	user can see the all flights details for selected data	As expected	pass
21	**Module**	**orbitiz_flight**				
22	**Project**	**www.orbitz.com**				
23	**TestCase**	**TC_1.02**				
24	**Requireme**	**user must be able to receive the flight details**				
25	**Requireme nt ID**	**RD01**				
26	**Pre**	**application must be hosted online**				
27	**Test Data**	**https:www.orbitz.com**				
28	**Brief Descriptio n**	**user must be able to receive the flight details and able to search the lowest fare of the flights**				
29	**SL No**	**DESCRPTION**	**INPUT**	**EXPECTED**	**ACTUAL**	**STATU**
30	1	go to the "flight only" page	tick the radio button having option flight only	form coressponding to flight only's form should be op	as expected	pass
31	2	click on the radio button having option one way trip	tick the radio button having option one way	form coressponding to one way trip's form should be	form is opened	pass
32	3	enter the text for "source" city	enter city "lsa"	from city code. Automatically city should come in text	As expected	pass
33	4	enter the text for "destination" city	enter destination city "la"	from city code. Automatically city should come in text	As expected	pass
34	5	enter the leaving date	enter the date in MM/DD/YYYY format	from date picker date autmatically select mention dat	As expected	pass
35	6	check the time box	tick the timebox for anytime search of flights	all the flights details of any time should come on selec	As expected	pass
36	7	click on search button	click on the search button	user can see the all flights details for selected data	As expected	pass

Table 3: Test suits for the flight page of the web application

Knowledge on software modeling, test criteria, test metrics, or languages to generate test scripts is tester skill requirements. Different approaches require different skills and these are needed ahead of usage. The main issue is how to minimize the human skill required and simultaneously maximize the automation level. To make ideal test cases, consider negative test suits which should not be accepted by the application. Test suits with negative value are also given below in Table 4.

The test case in red color show that we are trying to give the invalid values to the fields which should not accepted by the form. After creating all the test suits we feed them to the tool for "automation testing" which gives us the result, whether our test cases have passed or failed. When we use MBT for generating the test cases it is unlikely that any significant feature will remain uncovered during testing. The automation helps in selection of test cases for regression testing.

Module	**orbitiz_flight**				
Project	**www.orbitz.com**				
TestCase	**TC_1.04**				
Requireme	**user must be able to receive the flight details**				
Requireme nt ID	**RD01**				
Pre	**application must be hosted online**				
Test Data	**https:www.orbitz.com**				
Brief	**user should not receive the flight details**				
Descriptio	**if we are entering the wrong city in the**				
SL No	**DESCRPTION**	**INPUT**	**EXPECTED**	**ACTUAL**	**STATU**
1	click on the radio button having option round trip	tick the radio button having option round trip	form coressponding to round trip's form should be op	As expected	pass
2	enter the text for "source" city	enter city "xyz" that does not exist	city name should not be accepted	As expected	pass
3	enter the text for "destination" city	enter destination city "la"	from city code. Automatically city should come in text	As expected	pass
4	enter the leaving date	enter the date in MM/DD/YYYY format(02/11/2	from date picker date autmatically select mention date	As expected	pass
5	enter the return date	enter the date in MM/DD/YYYY format(02/28/	from date picker date autmatically select mention date	As expected	pass
6	select persons individually	select person - 1 adult	show the price for 1 person	As expected	pass
7	click on search button	click on the search button	user cannot see the all flights details it shows " please select valid city name"	As expected	pass
Module	**orbitiz_flight**				
Project	**www.orbitz.com**				
TestCase	**TC_1.05**				
Requireme	**user must be able to receive the flight details**				
nt ID	**RD01**				
Pre	**application must be hosted online**				
Test Data	**https:www.orbitz.com**				
Brief	**user should not receive the flight details**				
Descriptio n	**if we are entering the wrong date format in the**				
SL No	**DESCRPTION**	**INPUT**	**EXPECTED**	**ACTUAL**	**STATU**
1	click on the radio button having option round trip	tick the radio button having option round trip	form coressponding to round trip's form should be op	As expected	pass
2	enter the text for "source" city	enter leave city "syd"	from city code. Automatically city should come in text	As expected	pass
3	enter the text for "destination" city	enter destination city "la"	from city code. Automatically city should come in text	As expected	pass
4	enter the leaving date	enter the date in wrong format DD/MM/YYYY format(21/02/2015)	form should not accept the date	As expected	pass
5	enter the return date	enter the date in MM/DD/YYYY format(02/28/	from date picker date autmatically select mention date	As expected	pass
6	select persons individually	select person - 1 adult	show the price for 1 person	As expected	pass
7	click on search button	click on the search button	user cannot see the all flights details it shows message	As expected	pass

. Table 4: Test suits for the flight page of the web application

5. CONCLUSION

In this paper a study of model based testing is carried out. MBT provides the roadmap for automatic testing of software. It provides effective test strategies without missing any important test cases. Model-based testing is an efficient and adaptable method of testing software by creating a model describing the behavior of the system under test. The availability of tools makes the job easier. Two case studies are given in this paper which will help to understand the basic concept of MBT and also describe the way to generate test suits from the model. Large number of test cases can be generated from this model. The method that we have proposed in this paper is primarily for verifying the functionality of the SUT.

6 .REFERENCES

[1] Jinalben Patel, Roger Lee, Haeng-Kon Kim: Architectural View in Software Development Life-Cycle Practices in 6th

IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007) pp- 194-199.

.

[2] T. Y. Chen, and Pak- Lok Poon,: Experience With Teaching Black-Box Testing in a Computer Science in Software Engineering Curriculum Education, IEEE Transactions on (Volume:47 , Issue: 1) pp: 42-50, 2004

[3] Rajiv Chopra, Sushila Madan : Reusing Black Box Test Paths For White Box Testing of Websites in Advance Computing Conference (IACC), 2013 IEEE 3rd International pp: 1345-1350 .

[4]. Neto, a.d. ; subramanyan, r. ; vieira, m. ; travassos, g.h. ;shull, f.:improving evidence about software technologies: a look at model-based testing in software, ieee (volume:25 , issue: 3). Pp:10-13, 2008.

[5] El-Far, I. K. & Whittaker, J. A. Model-based Software Testing . In: Marciniak, J. (ed.), Encyclopedia on Software Engineering, Volume 1. New York, USA: John Wiley & Sons Inc, 2001. pp. 825-837. ISBN 0-471-21008

[6] David Lee, Mihalis Yannakakis: Principles and method of testing Finite State Machine in Proceedings of the IEEE (Volume:84 , Issue: 8) pp :1090-1123, 1996

[7] Hassan Reza, Kirk Ogaard, Amarnath Malge: model based testing technique to test web applications using statecharts in Information Technology: New Generations, 2008. Fifth International Conference. pp: 183-188, 2008

[8] Emanuela G. Cartaxo, Francisco G. O. Neto and Patricia D. L. Machado: Test Case Generation by means of UML Sequence Diagrams and Labeled Transition Systems in. pp: 1292-1297

[9] Philippe, B. *Introduction* to the Numerical Solution of *Markov Chains in Computational Science & Engineering, IEEE* (Volume: 3 , *Issue:* 2) *(1996)*.

[10] Chris Jones : An Example-Based Introduction to Graph Grammars for Modeling. System Sciences in Proceedings of the Twenty-Third Annual Hawaii International Conference on (Volume:iii). pp: 433-442, 1990

[11] Http://mit.bme.hu/~micskeiz/pages/modelbased_testing.html#tools

[12] A Avritzer, B Larson - ACM SIGSOFT Software Engineering Notes, 1993 - dl.acm.org

[13] IK El-Far, JA **Whittaker** - Encyclopedia of Software Engineering, 2001 - Wiley Online Library

APPENDIX 1

LIST OF OPEN SOURCE TOOLS FOR MBT[11]

NAME	INPUT FORMAT	DESCRIPTION
FMBT	Custom (AAL)	fMBT (free Model-Based Testing) generates test cases from models written in the AAL/Python pre/postcondition language using different heuristics (random, weighted random, lookahead)
GRAPH WALKER	FSM	Test generation from Finite State Machines. Search algoritms: A* or random, with a limit for various coverage criteria (state, edge, requirement). Formerly called as mbt.
JTorX	LTS	JTorX is a reimplementation of TorX in Java with additional features. The LTS specification can be given in multiple format, and it can interact on-the-fly with the implementation under test.
MODELJUNIT	EFSM	ModelJUnit allows you to write simple finite state machine (FSM) models or extended finite state machine (EFSM) models as Java classes, then generate tests from those models and measure various model coverage metrics.
OSMO	JAVA	OSMO uses model programs written and annotated in Java, and creates tests by exploring these models using different strategies (random, using constraints to guide
PyMODEL	Python source	yModel supports offline and on-the-fly testing. It uses composition for scenario control. Coverage can be guided by

		a programmable strategy.