Assignment 5

Test generation and execution

Course ID: DAT260 Model based testing

Group 2:
Vu Ngoc Quang
Choo Yi Ming
Khaled Al Sabbagh

Software under test

Our software under test (SUT) is Fossil. Fossil is not a small service and has many functions, our group chose to reduce the scope of testing so that our project will be more manageable and the model will not be very complex. Specifically, we chose to test 3 sub systems of Fossil: tickets, reports and wiki. Although each of them is only a part of the entire Fossil system, each still has a large set of functions to be tested.

Difficulties and remarks about the implementation of our model and adapter

Ticket system

The actions that are included in the models are transition from Home page to Ticket Page, new ticket, submit ticket, edit ticket and submit edited ticket. The result of any transition is verified by checking the result Url. To do this, part of the current Url is checked against known keywords by assertEquals statements. For example, after clicking “New Ticket” link, the browser should be directed to a new Url that contains the keywords “tktnew”.

Additionally, after submitting a new ticket or an edited ticket, the browser is supposed to be in the View Ticket page. To verify if a ticket is created or edited successfully, our code will verify the “Title” and “Description” field of the ticket against what are expected.

Report system

One problem when implementing our model is that it is difficult to write the reset function. Our SUT does not have any function to remove all existing report efficiently. Instead, our code has to go through each report like how a human user would do and manually delete one by one until the number of reports is reduced to the initial number.

One remark about the implementation of the model is that the @Action viewReport( ) will randomly choose a report from list to display. This simulates the fact that user can choose any report that he is interested in.

To verify the correct state that the SUT is in, after every action, the current URL is obtained through getCurrentUrl() of the WebDriver. This will be compared with the expected Url that the SUT is supposed to be in. To compare, assertEquals statements are used to check for keywords in the Url.

Wiki system

As listed earlier in assignment 4, the EFSM for the Wiki service tackled nine different actions. The generated outcome was tested using four different testing techniques (Random, Lookahead, Greedy, and Allround).

To validate the implementation of the test, we used the method printGraphDot to reverse engineer the code back to the original model. The resulting text description, found in the WikiM.dot file, was then executed in Webgraphvis.com, which in turn has visualized the model in a graph (Appendix).

The design of our model was missing a few connections between the states. As a result, the code was throwing exceptions at some transition actions. We had to rethink and improve our EFSM design to mitigate the problems. Moreover, resetting the fossil repository was a cumbersome task which required an ample time of research. The problem with the reset function is that Fossil does not allow the deletion of existing artifacts. We created a batch file named “Wiki”. The file creates a new repository with a unique name every time the user runs executes it. In this manner, the test will operate on an empty repository every time a reset is required. Note that the batch file needs to be placed on the Desktop directory to execute. Also note that the batch file was created on a MAC-OS, i.e there might be some problems if it was ran on any other platform.

Reset function implementation

Our reset function in MyAdapter.java will quit the current Firefox driver. After that, it will open a new Firefox process and go to the local home page of Fossil. Specifically, the function will do:

```
driver.quit();
driver = new FirefoxDriver();
driver.get("http://127.0.0.1:8080/");
```

Furthermore, the reset function will delete all the reports created during testing by calling a deleteAllMethod() in the adapter. This function will go to the Ticket Main Menu page and delete all the present reports one by one. For each deleting, the code will simulate user action by going to ViewReport page, clicking on Edit link and finally clicking on Delete button. Wiki status is also reset to the initial state by setting wStatus = WikiStatus.Initial.

Lastly, the reset calls the function clickHome() in the adapter to direct the browser to the Homepage, which is the starting state of the test.

Test generation strategy

There are 4 test generation strategies available in ModelJUnit:

GreedyTester

In this method of testing, the test will make a walk through the EFSM model of the SUT. If a transition out of a state is never taken before, the test prefer that transition to other transitions. Only when all transitions are already taken then the test will choose randomly.

AllRoundTester

This method is similar to GreedyTester but it will stop the test when the loopTolerance variable is exceeded. In other words, the test will halt when a state is entered more often than what is set by the loopTolerance variable.

LookaheadTester

This tester will choose the action that is least traveled before at N-level ahead in the graph. Any action has a value attached to it that is determined by the value of the state it leads to minus the number of time that action has been taken. Therefore, this value will decrease over the course of the test the more time it is taken and thus the tester will be less likely to take this action again.

RandomTester

In this method, the test will choose randomly a transition during a state in the EFSM model of the SUT.

Coverage metric

Four coverage metrics are available in ModelJUnit:

ActionCoverage

This metric measures the ratio of the number of action methods taken by the SUT during testing over the total number of actions available.

StateCoverage

This is a metric to measure the ratio of the number of times the SUT reaches each state over the total number of available states of the SUT.

TransitionCoverage

This metric is used to show the ratio of the number of transitions that the SUT has taken over the total number of transitions available to the SUT.

TransitionPairCoverage

This metric measures the ratio of the number of adjacent transition pair (one transition leading to a state and another transition leading out of that state) traversed by the SUT during test over the total number of transition pair available to the SUT. In other words, it counts the number of unique actions performed by the test.

Coverage table

Tested using generate(200)

	ActionCoverage	StateCoverage	Transition Coverage	TransitionPair Coverage
AllRoundTester	18/20	4/16	27/30	25/69
GreedyTester	18/20	15/16	27/30	49/69
LookaheadTester	20/20	16/16	30/30	53/69
RandomTester	15/20	12/16	22/30	44/69

Discussion

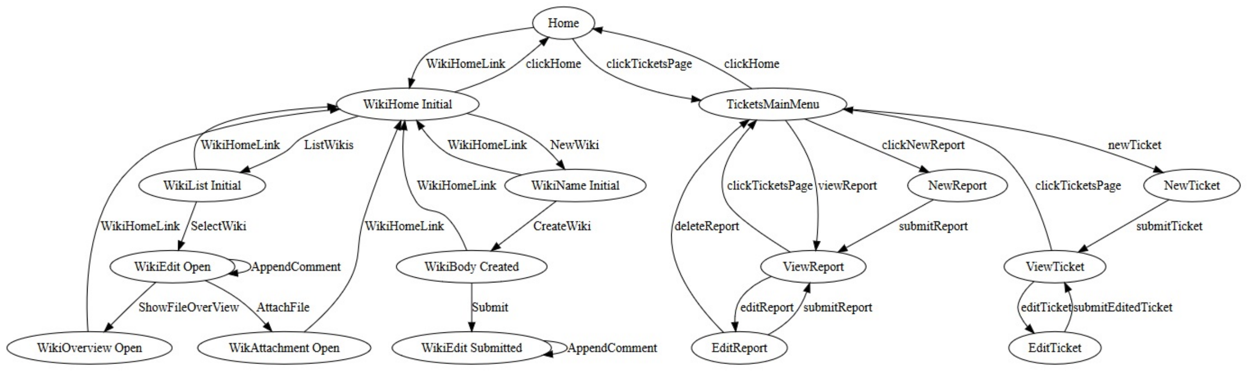
The advantage of using model-based testing (MBT) on Fossil is that the technique helps to generate long test sequences which simulate continuous usage of Fossil by users. MBT improves test coverage which would be difficult to achieve in other testing techniques by adding 3 coverage metrics (transition, state and action metrics). By using RandomTester provided by Model JUnit, random actions and transitions can be called during testing. This simulates how a user interacts with Fossil by going from page to page, creating and editing tickets, reports and wiki. Other testers like LookaheadTester, AllroundTester and GreedyTester ensure more coverages as well based on how each of those testers works.

One drawback of MBT on Fossil is that it requires a very detailed design of the model. The model must capture all essential pages of the system. Even when we decide to reduce our scope of testing on Fossil, the ticket-report-wiki features still required us to dedicate a large amount of time on designing its model and adapter. Additionally, our model has a lot of transitions between states. This leads to a very long testing time since the testers have many choices to choose from the range of transitions available at any state. In order to achieve high coverage, we need to run many tests, which also increases test runtime.

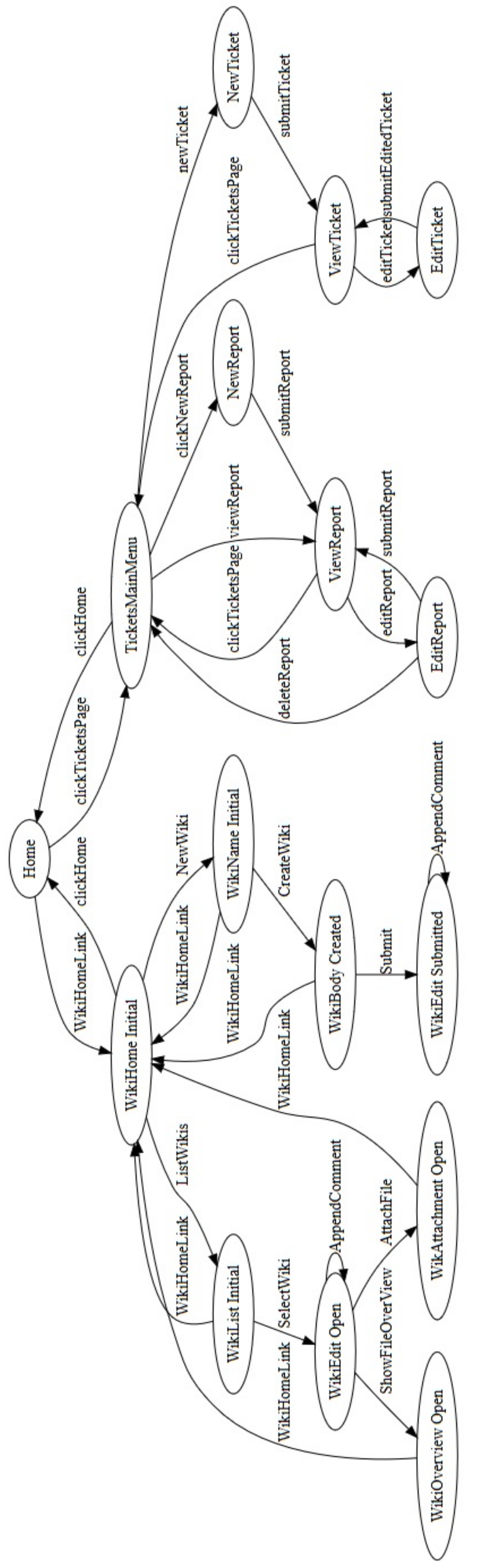
Despite the drawback, it is worth the time and effort to use MBT on Fossil since the advantage outweighs the drawback given the efficiency of the technique. More tests can be run by using MBT than other testing techniques. Furthermore, with MBT, test sequence is automatically generated by the tester which greatly helps avoid the pesticide paradox. Last but not least, since tests are automatically generated, even when we modify our model, the new test cases will be taken care of by the testers.

<u>Appendix</u>

Webgraphvis generated model



(Rotated image on next page)


Home
TicketsMainMenu
NewTicket
ViewTicket
EditTicket
NewReport
ViewReport
EditReport
WikiHome Initial
WikiName Initial
WikiBody Created
WikiEdit Submitted
WikiList Initial
WikiEdit Open
WikAttachment Open
WikiOverview Open
clickHome
clickTicketsPage
newTicket
submitTicket
clickTicketsPage
editTicket
submitEditedTicket
clickNewReport
submitReport
viewReport
clickTicketsPage
editReport
submitReport
deleteReport
clickHome
WikiHomeLink
NewWiki
WikiHomeLink
CreateWiki
WikiHomeLink
Submit
AppendComment
WikiHomeLink
ListWikis
WikiHomeLink
SelectWiki
AppendComment
AttachFile
WikiHomeLink
ShowFileOverView
WikiHomeLink