Journal of Computer Science 10 (2): 190-197, 2014
ISSN: 1549-3636
© 2014 Science Publications
doi:10.3844/jcssp.2014.190.197 Published Online 10 (2) 2014 (http://www.thescipub.com/jcs.toc)

A NOVEL PRIORITIZATION ALGORITHM MODEL BASED TEST-SUITE GENERATION USING REGRESSION TESTING

[1]Prabhu Jayagopal and [2]Dr. Malmurugan Nagarajan

[1]Research Scholar, Department of Computer Science and Engineering, Sathyabama University, Chennai, India
[2]Director, Sri Ranganathar Institute of Engineering and Technology, Coimbatore Tamil Nadu, India

Received 2013-09-01; Revised 2013-10-14; Accepted 2013-11-13

ABSTRACT

The fully automatic Graphical User Interface tool for any application using novel model based test suite generation techniques for a GUI. They are unable to control response time and time intervals are based on relationship between GUI events handlers and test cases with their responsibilities. We present a novel prioritization algorithm that enhances event handlers for the automated GUI tool. The proposed tool generates GUI events, it Captures and Playback event responses to automatic verification point of the results for the test cases which are written to a log file and corresponding report will be generated. This novel algorithm was able to detect new test suite and ordering of test cases to reduce a GUI fault integration defects. The number of faults detected for a single event are found after generating test cases for the application. The Average Percentage of Fault Detection (APFD) and charts has been used to show the effectiveness of proposed algorithm to find fault detection rate.

Keywords: Regression Testing, GUI Testing, Test Suite, Novel Prioritization Algorithm, Capture/Playback

1. INTRODUCTION

The Graphical User Interface application are progressively more in real-world market. GUI are now seen in mobile Phones, micro oven, cars, iPod. They are popular because of the portability, flexibility that they offer for the users. The Software systems have been built on event-driven software platforms. This enables the user either use (1) Mouse click (2) Mouse drag (3) Mouse release (4) Mouse select or short cut key to change the event state, this may include a change the software state, which may impact the execution of subsequent events. Hence, the context established by the sequence of events executes may have an impact on how it executes.GUI has been converted into a crucial component of any electronics devices with the user interact. The fundamental nature of GUI is of sensitive operation. On the other hand, as the functional complication of application increased. The repeated usage of cursor operations by the user to give suitable comments to the system. For making comparing with the GUI in order to get numerous operations such as cursor pointing, drag and dropping the menu and resizing the windows should be continual for each display object. For the past ten years, many software system had been developed on the basis of event driven software platforms. GUI has been developed from the event-driven software, which will used to start-up the user to either mouse release, mouse drag, mouse click and also key in data as input to change the event state. The generic prioritization criteria that are applicable to both GUI and Web application. It is to evolve the model and use it to develop unified theory for all Event Driven Software should be detected (Bryce *et al*., 2011). At present circumstance criteria, the GUI software application in our daily life routine. So these GUI are now available in mobile phones, micro ovens, music system, iPod so they permit a programmer to develop the GUI by coding the software event handlers.

The fully automatic model based GUI testing resulted, aggravated by work on prioritization algorithm for test data generation, The Test Case Prioritization is proposed in recent years, it can improve the fault

Corresponding Author: Prabhu Jayagopal, Department of Computer Science and Engineering, Sathyabama University, Chennai, India detection during the testing phase. The weighted and non-weighted GUI test cases based on weight scores. The weighted scores can be ranked in ascending or descending order. The result shows that dynamic adjusted-weight method can obtain a better fault-detection rate. The efficiency of detected faults is not always the same (Huang *et al*., 2010). The tester must specify the test data coverage criterion to be used, either branch coverage or mutation analysis. It is integrated into javascript compiler and test generation by a command line option (Alshraideh, 2008). The notion of utilizing a fault-based approach to test case prioritization is novel and n concrete terms how the approach may apply to test suites generated to detect faults related to logical expressions in specifications (Yu and Lau, 2011). The search effort is then distributed amongst the paths, with several 'species' working in parallel, each dedicated to finding test data for an individual path (McMinn *et al*., 2006). The interaction with it primarily using a mouse, launches programs by clicking on icons and manipulates various windows on the screen using graphical controls (Reimer, 2005). The code modifications made to create a new version may alter test execution patterns; an issue impacting the efficiency of test case prioritization techniques is whether these alterations will significantly impact the predictive value of past execution data (Rothermel *et al*., 2001).

In this study, we propose:

- GUI testing can test any application provided the appropriate packages and interfaces are written for that language
- The state based logging type, the start and end time of each event that uniquely define a state are stored in the log file. This file type contains a set of interval records each one of them is characterized as 'begin interval', 'end interval', 'continuation interval' and 'complete interval'. Since each occurrence of event is time stamped, we can measure the responsiveness of the GUI
- We can use GUI capture and playback event at the background, unlike in the automated testing. The application has to designed what to test
- We focused on an novel prioritization algorithm to generate test suite for above the same

1.1. GUI Testing

The GUI existing testing techniques have been focus on implementing the automated GUI testing tools and adopted by practioners (Marchetto *et al*., 2008; Memon, 2008). The most popular GUI testing approach used my previous work, compared various testing tools like Junit, Abbot, Marathon, Pounder, Robot, QTP.

In the automated testing process, testers have to ensure the validation of software using testing techniques. Before capture a testing process, we must decide to criteria for expressive the capability of testing software (Jatain and Sharma, 2013).

A graphical user interface for a.net may be implemented using new components, GUI events, which must be handled by the program. Thus, GUI events are an important class of inputs to.net Codes, which capture and replay correctly and efficiently, should be done in the interactive applications. Capture of GUI events is significantly different from the capture of other kinds of inputs, playing back of events in the application and the corresponding test case will be generated.It is Based on data captured and the data which is stored in the database, A report showing the type of event, unique id for the event, the time of the event and the screenshot of the application when the event took place is generated. Based on the type of event, the corresponding test cases are generated.

The existing methods used for modeling and testing a GUI also affect its reliability. Consequently, the quality of the reliability assessment process and ultimately, the reliability of the GUI depend on the approaches used for modeling and testing (Belli *et al*., 2012).

The present actual data on the experiences and to discuss if advantages can be gained using model-based testing when compared with traditional graphical user interface testing. Another contribution of this paper is a description of a keyword-based test automation tool that was implemented for the Android emulator. All the models and the tools created are available as open source (Takala *et al*., 2011). The **Fig. 4** shows an important limitation is that contain state based relationships. Relationship between E1 and E5. The desirable coverage requires large number of test suites.

In earlier work, we found a feedback-based techniques to enhance a two ways of covering test cases are as follows (1) is able to significantly improve existing techniques and helps identify serious problems in the software and (2) the ESI relationships captured via GUI state yield test suites that most often detect more faults than their code, event and event-interaction-coverage equivalent counterparts (Yuan and Memon, 2010). The GUI events interact in difficult ways an GUI reply to an event wary depending on the preceding event and their running orders. The capture and replay event have been developed as a techniques for testing the verification of interactive GUI applications. Using capture the entire event occurred in the application can be recorded. The replay event is used to repeat the application process, An quality-assurance group can run an application and record the entire interactive session. The tool records all the user's events, such mouse clicks, mouse release, mouse drag and the keys press from the keyboard. All these events will be recorded to see fault detected during implementation and it is stored in log file using JASON object. This tool can then automatically replay the exact same interactive session any number of times without requiring a user. The capture and replay events are usually not used for recording entire interactive sessions. their main aim is to record complex interaction sequences, such as the user clicking on the screen like mouse click on the file and then open to verify that this click will response by the software system or not We studied whether existing GUI capture and replay tools can be used to record entire interactive sessions with complex real-world applications and whether the tools allow or preclude the accurate measurement of perceptible performance given the overhead they impose on the application. A verification point enables during capturing the GUI application, the object information stores it in a log file. This file becomes the base of the expected state of the object during subsequent builds. When you play back the GUI Interactive events it retrieves from the log file.

Our automation tools retrieve the information from the log file for each verification point and compare it to the state of the object in the new build. After playback, the results of each verification point appear in the tester Log file. If a verification point fails you can select the verification point in the log. The Reports will be generated after correcting the bugs in the application.

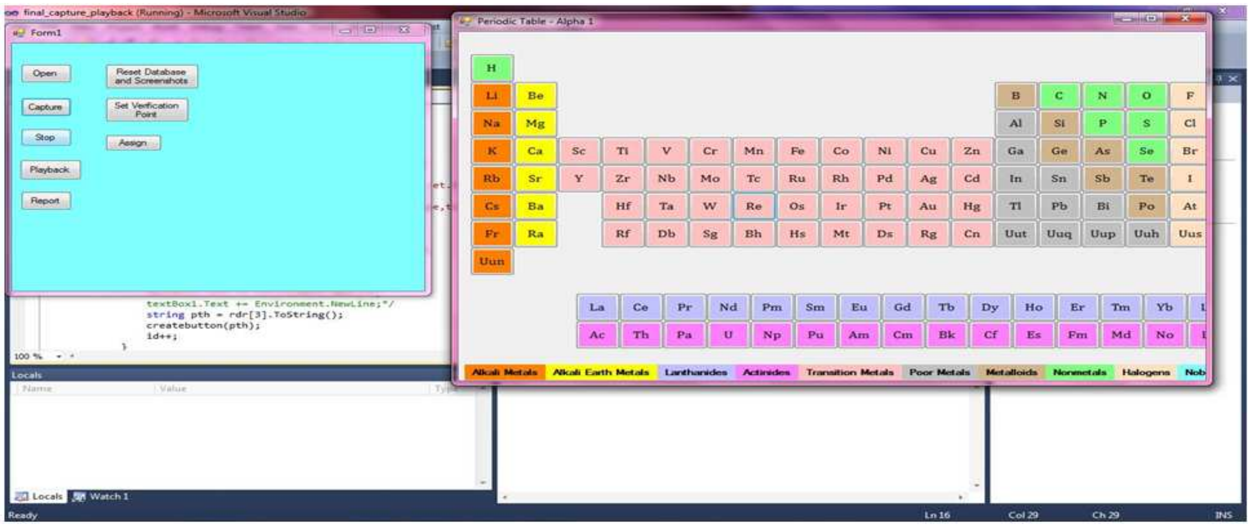
1.2. Average Percentage of Faults Detected (APFD)

To measure the target of rising a separation of the test case of fault detection. APFD founded (Ashraf *et al*., 2012).

The **Fig. 1** shows an Novel GUI tool with capure/playback, opening the application, Report generation, Reset database, set verification point and assignining the values periodic table holds the multiple colors of tables with their description.

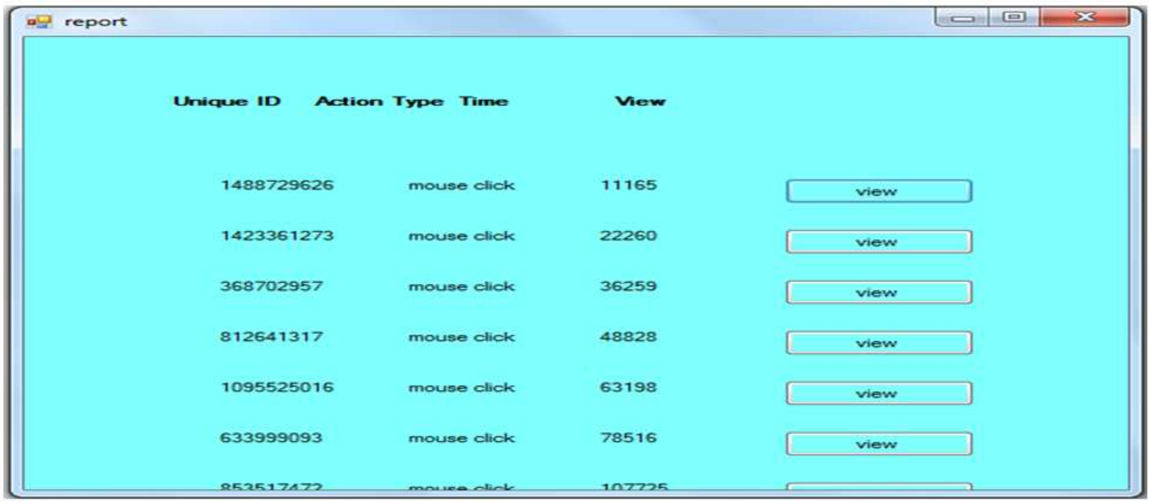
The **Fig. 2** shows an report generation of each and every event occurred in the application with their unique Id, action type and view. The **Fig. 3** shows the interface between the events occurred and their response to their other events. In earlier study of a test case Prioritization consists of input and output value and expected result before testing. Although test-case execution should be successful, if some errors occur during execution, the output value cannot be obtained or compared with the expected result (Huang *et al*., 2010).

An Event Flow Graph (EFG) consists of all events and all possible interactions. Interactions are a set of directed edges between events and events are the vertex in the graph. This graph also records which events will be invoked continuously (Huang *et al*., 2010).



```
textBox1.Text += Environment.NewLine;"/
string pth = rdr[3].ToString();
createbutton(pth);
id++;
}
```

Fig. 1. A Simple GUI tool with an application



Unique ID	Action Type	Time	View
1488729626	mouse click	11165	view
1423361273	mouse click	22260	view
368702957	mouse click	36259	view
812641317	mouse click	48828	view
1095525016	mouse click	63198	view
633999093	mouse click	78516	view
853517472	mouse click	107725	

Fig. 2. Report generation for the events

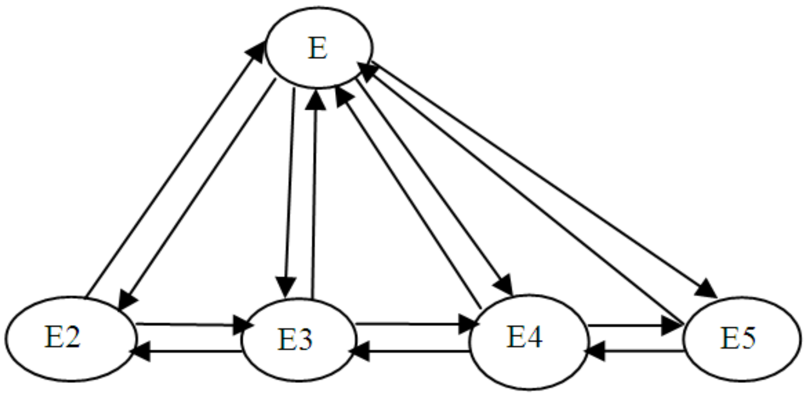


Fig. 3. Events for GUI application

Automated GUI Testing is a solution to all the issues raised with Manual GUI Testing. An Automated GUI Testing tool can playback all the recorded set of tasks, compare the results of execution with the expected behavior and report success or failure to the test engineers. Once the GUI tests are created they can easily be repeated for multiple number of times with different data sets and can be extended to cover additional features at a later time. Most of the software organizations consider GUI Testing as critical to their functional testing process and there are many things which should be considered before selecting an Automated GUI Testing tool. A company can make great strides using functional test automation. The important benefits include, higher test coverage levels, greater reliability, shorted test cycles, ability to do multi user testing at no extra cost, all resulting in increased levels of confidence in the software (Prabhu and Malmurugan, 2010).

The **Table 1** shows the events with the corresponding action occurred in the GUI application.

It measures the average rate of fault detection of test suite execution. The APFD is calculated by taking weighted average of the number of faults detected during the run of the test suites. APFD is defined as:

$$\text{APFD} = (1\text{-}TF_1 + TF_2 + ... + TF_m/nm) + (1/2n)$$

T$\rightarrow$test suite under evaluation
m$\rightarrow$number of faults
n$\rightarrow$ total no. of test cases
$TF_m\rightarrow$position of test

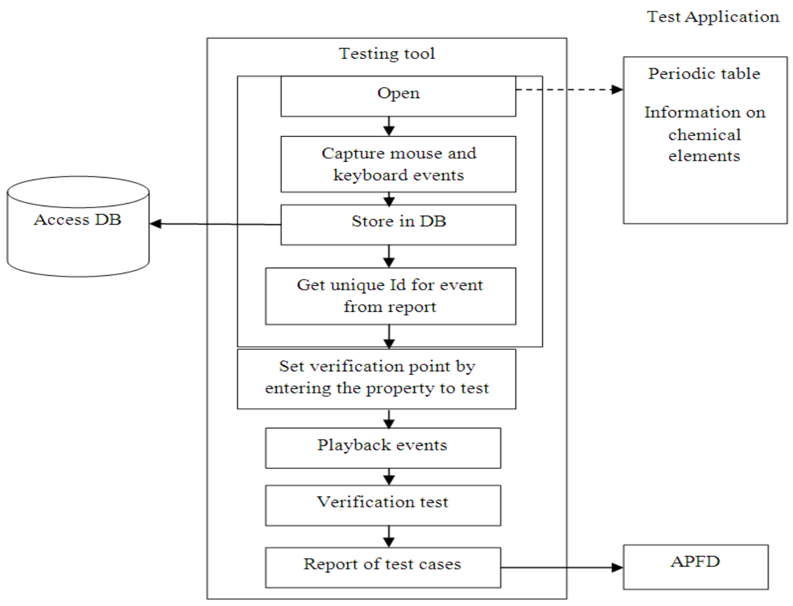

Test Application
Testing tool
Open
Capture mouse and keyboard events
Store in DB
Get unique Id for event from report
Set verification point by entering the property to test
Playback events
Verification test
Report of test cases
Access DB
Periodic table
Information on chemical elements
APFD

Fig. 4. Software architecture experimental procedure

Table 1. Events and actions in the GUI application

Events	Actions
E1	Changes in color
E2	Display the description box
E3	It glows on the button
E4	Disables the button
E5	Drag and copy the description

Table 2. The number of faults detected for an event E1 to generate test suite

	TEST SUITE								
E1	TC_1	TC_2	TC_3	TC_4	TC_5	TC_6	TC_7	TC_8	TC_9
FD_1		x		x			x		
FD_2	x		x						
FD_3					x			x	
FD_4		x				x			
FD_5			x				x		
FD_6				x					
FD_7						x		x	
FD_8	x	x		x			x		
FD_9									x
No. of Faults	2	2	2	3	1	2	3	2	2
Time	4	8	3	2	11	6	7	8	9

1.3. Novel Prioritization Technique

In earlier work, it makes take long time depending the size of test cases. How long each test case takes to run. On the other hand through the use of an effective prioritization technique. Software testers can be in random order test cases to attain an increased rate of fault detection. Novel technique presented in this study implemented a new regression test suite using prioritization algorithm that prioritized the test cased with the target of faults can be found during the execution of test suite. The below pseudo code for ordering test cases from lowest PFD value to highest PFD value. the variable means the current minimal PFD value in all test cases. Initially the value of FD will make null, Uot the test cases will be in unordered list. All test cases are sorted in order to make a effective test suite.

Algorithm

```
Input:
    Uot: Unordered test cases
    FD: Summation of fault detections
    E: Event handling
Output:
  TS: New prioritized Test Suite
      1. Begin
      2. Set TS empty
      3. Set E empty
      4. For each event E→TS do
      5. Calculate average faults found in a minute PFD =
FD×2/time
```

```
    6. End for
    7. Sort TS in ascending order based on the value of each test suite
    8. APFD value generated
    9. End
```

PFD_n = fault * 2 / time
$PFD_1 = 1$
$PFD_2 = 0.75$
$PFD_3 = 1.33$
$PFD_4 = 3$
$PFD_5 = 0.18$
$PFD_6 = 0.66$
$PFD_7 = 0.85$
$PFD_8 = 0.5$
$PFD_9 = 0.33$

Prioritization order as follows:

$PFD_5 + PFD_9 + PFD_8 + PFD_6 + PFD_2 + PFD_7 + PFD_1 + PFD_3 + PFD_4$
APFD=(1-1+0.75+1.33 +3+0.18+0.66+0.85+0.5+0.33/9*9) +(½*9)
=(1-8.6/81)+(1/2*9)
=(1-0.1061)+(1/2*9)
=(0.8939)+(1/2*9)
=0.8939+0.055
=1.4494

The Average percentage of fault detection metrics has been used to measure the efficiency of proposed and random prioritization and it shows that the proposed value based algorithm is more efficient than random prioritization to generate sequence of test cases for early rate of fault detection (Ashraf *et al*., 2012).

Definition: A test case consists of input value, output value and expected output before starting testing. The function takes as input a set of test cases to be ordered and returns a sequence that is ordered by the prioritization criterion. Because we have developed a unified model of GUI and Web applications, we need the function to be extremely general so that it may be instantiated for either application class and is able to use any of our criteria as a parameter. The function (called OrderSuite) selects a test case that covers the maximum number of criteria elements (e.g., windows and parameters) not yet covered by already-selected test cases. The function iterates until all test cases have been ordered (Sampath *et al*., 2013).

1.4. Source Code for Creating Test Casses

```
public void createtestcasebutton(string Val)
    {
        ob5[i] = new Button();
        this.ob5 [i].Text = "Test Case";
        testypos += 50;
        this.ob5 [i].Location = new
System.Drawing.Point (testxpos, testypos);
         this.ob5 [i].Size = new
System.Drawing.Size(100, 25);
         this.Controls.Add(ob5[i]);
         this.ob5[i].Click += delegate(object sender1,
EventArgs ee)
        {
           createtestcases(sender1, ee, val);
         };
         i++;
     }
     public void createtestcases(object sender,
EventArgs e, string val)
       {
          if (val == "mouse")
          {
             Mousetestcases ob = new mousetestcases();
          ob.Show();
       }
       else if (Val == "key")
        {
          Keytestcases ob1 = new keytestcases ();
          ob1.Show ();
       }
}
```

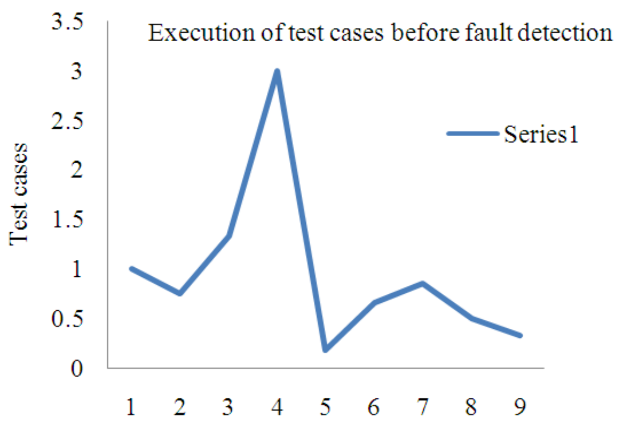


Fig. 5. The cummulative of test cases before the fault detection rate

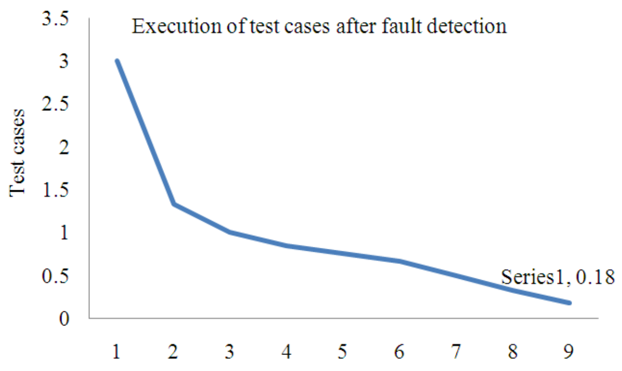


Fig. 6. The cummulative of test cases after the fault detection rate

It is used for playing back the events which were recorded during the capture phase. Based on each tick of system clock and the data stored in the structure, all mouse and keyboard events get replicated and if a verification point is set, then during playback, at the corresponding event, The data gets tested whether the test passed or failed.

2. RESULTS

From **Table 2** which is also represented in **Fig. 5 and 6**, shows the fault detection is very effective after ordering the test cases compared to unordered test cases. It is identified that the fault detection rate is sequence and computational cost and transmission cost of the proposed method are improved than the existing model.

3. DISCUSSION

The novel Prioritization algorithm for model based test suite generation presented in this study documents certain aspects of GUI testing. In this section we present an objective summary of trends in GUI testing. From the data collected, it can be seen that model-based GUI testing techniques have attracted the most attention in the Research community. However, industrial tools such as Pounder, Marathon, Jacareto, JFC Unit, QTP are model based on improving the response time, capture/Replay, ordering of test cases with prioritization with comparing the GUI testing techniques, methods and practices in the research community. There has also been a general lack of collaboration between practitioners and researchers (**Fig. 4**), although with exceptions in recent years. These techniques are typically not usable by other researchers because they are not widely applicable. It provides guidance about possible future development and research directions.

4. CONCLUSION

In this study we presented a new automated tool for any GUI applications. The proposed Prioritizatation algorithm is used to Ordered of test cases using regression testing, implemented proof-of-concept tool support for the approach and combined the implemented GUI tool with an model-based approach aims to reduce the amount of fault detection rate in the test suite generation, it is required to model based GUI applications to enable quick response time and time interval in GUI events in automated testing. In our previous work, the strengths of our approach in comparison to the automated testing tools include automatically generating human readable graphical models while requiring none or only a little manual effort. In future, we plan to improve the GUI Tool so that the generated Feedback would inform about the detected usability issues and include information about the changes that happened in the GUI after a specific interaction. The GUI Tool should indicate more clearly the states that should be manually elaborated in the model and support iterative modeling containing manual and automated phases. Also, we plan to extend the approach to be also usable on other kinds of GUI applications.

However, in this study we didn't consider that some events might give failed test cases events are unrestricted to the action take place in the application. We might need to further investigate whether the fault-detection ability of the other tool is the same as the latter. Furthermore, we still have to know how to generate report generation for other application. We plan to study and present above mentioned issues in the future.

5. REFERENCES

Alshraideh, M., 2008. A complete automation of unit testing for JavaScript programs. J. Comput. Sci., 4: 1012-1019. DOI: 10.3844/jcssp.2008.1012.1019

Ashraf, E., A. Rauf and K. Mahmood, 2012. Value based regression test case prioritization. Proceedings of the World Congress on Engineering and Computer Science, Oct. 24-26, San Francisco, USA.

Belli, F., M. Beyazit and N. Guler, 2012. Event-oriented, model-based GUI testing and reliability assessment-approach and case study. Adv. Comput., 85: 277-326.

Bryce, R.C., S. Sampath and A.M. Memon, 2011. Developing a single model and test prioritization strategies for event-driven software. IEEE Trans. Soft. Eng., 37: 48-64. DOI: 10.1109/TSE.2010.12

Huang, C.Y., J.R. Chang and Y.H. Chang, 2010. Design and analysis of GUI test-case prioritization using weight-based methods. J. Syst. Software, 83: 646-659. DOI: 10.1016/j.jss.2009.11.703

Jatain, A. and G. Sharma, 2013. A systematic review of techniques for test case prioritization. Int. J. Comput. Applic., 68: 38-42. DOI: 10.5120/11554-6833

Marchetto, A., P. Tonella and F. Ricca, 2008. State-based testing of ajax web applications. Proceedings of the 1st International Conference Software Testing, Verification and Validation, Apr. 9-11, IEEE Xplore Press, Lillehammer, pp: 121-130. DOI: 10.1109/ICST.2008.22

McMinn, P., M. Harman, D. Binkley and P. Tonella, 2006. The species per path approach to SearchBased test data generation. Proceedings of International Symposium on Software Testing and Analysis, Jul. 17-20, ACM Press, Portland, ME, USA., pp: 13-24. DOI: 10.1145/1146238.1146241

Memon, A.M., 2008. Automatically repairing event sequence-based GUI test suites for regression testing. ACM Trans. Software Eng. Methodol., 18: pp: 1-36. DOI: 10.1145/1416563.1416564

Prabhu, J. and N. Malmurugan, 2010. A survey on automated GUI testing procedures. Eur. J. Sci. Res., 64: 456-462.

Reimer, J., 2005. A History of the GUI. Arc Technical, LLC.

Rothermel, G., R. Huntch, C. Cu and M.J. Harold, 2001. Prioritizing test cases for regression testing. IEEE Trans. Software Eng., 27: 929-948. DOI: 10.1109/32.962562

Sampath, S., R. Bryce and A.M. Memon, 2013. A uniform representation of hybrid criteria for regression testing. IEEE Trans. Soft. Eng., 39: 1326-1344. DOI: 10.1109/TSE.2013.16

Takala, T., M. Katara and J. Harty, 2011. Experiences of system-level model-based GUI testing of an android application. Proceedings of the IEEE 4th International Conference on Software Testing, Verification and Validation, Mar. 21-25, IEEE Xplore Press, Berlin, pp: 377-386. DOI: 10.1109/ICST.2011.11

Yu, Y.T. and M.F. Lau, 2011. Fault-based test suite Prioritization for specification-based testing. Inform. Software, 54: 179-202. DOI: 10.1016/j.infsof.2011.09.005

Yuan, X. and A.M. Memon, 2007. Using GUI run-time state as feedback to generate test cases. Proceedings of the 29th International Conferences on Software Engineering, May 20-26, IEEE Xplore Press, Minneapolis, MN., pp: 396-405. DOI: 10.1109/ICSE.2007.94