A NOVEL APPROACH FOR GUI TEST AUTOMATION USING CONSCIOUS DEVELOPMENT CONSTRUCTS

[1]Silvi Goel, [2]Dr. Rinkle Aggarwal
Thapar University, Patiala
Email:[1]goelsilvi90@gmail.com, [2]raggarwal@thapar.edu

Abstract- With rapid advancement in technology, the expectations from software industry have seen a drastic change. Providing interactive and maintainable solutions is an essential need of hour. A graphical interface to the system is an appropriate solution to such needs. GUIs are playing an important role in making systems more interactive, configurable and maintainable. They are not only benefiting the end-users, but can also contribute to the project lifetime, offering help at the developer end. Thus, reliability and accuracy of GUI must be thoroughly ensured. With lesser complexity, such assurance was easy with manual efforts, but present scenario of complex systems, makes it very questionable. Nowadays, testing must be approved with a full proof plan and its completion is subject to bias if done manually. In this paper, accuracy, fault-tolerance, control coverage, event coverage and functional coverage are primary objectives while solving the overhead of script based automated testing, so that human intervening can be minimized and testing process can be scalable and sufficiently complete to support multiple releases.

Index Terms- GUI, Accuracy, Fault-tolerance, Functional coverage, Control coverage.

I. INTRODUCTION

Increasing complexity of software systems pose a big challenge to the companies on various fronts such as ease-of-use, maintainability of system, proceeding with legacy codes. Graphical User Interface (GUI) establish a friendly way of communication between end-users and ease the understanding about existing system if a new candidate is employed for system up-gradation. Thus a well-built GUI can directly impact the client and developer significantly. Moreover efficiency and quality of software highly depend on the communication interface, quality of development and testing techniques adopted by developer. Minimization of risk factor is another criteria for evaluating the applicability of the techniques adopted. In this paper, testing the GUI components is being put to focus. But making an appropriate choice is not as easy since GUI testing bring along a number of challenges. Certain factors that play determining role while opting for a testing strategy are the technology used for GUI design, deployment platform, structural profile of GUI [1] , complex event interactions and functional design of GUI. Increasingly complex GUI systems are now beyond reach of manual testing. Software quality parameters are now more quantitative and a number of trade-offs are being made as per project requirements. Thus an automated tested system can evaluate the System Under Observation with lesser human efforts and errors that may crawl-in due to manual bias.

II. TESTING MODES

Testing the GUI can be done in manual or automated mode. Pros and cons are being discussed in the section and a justified switch to automated test suite can be thus ruled out. Manual Testing is a tedious task executing a number of test cases manually designed by engineer and requires the tester to possess ample patience, good observing power, creative, open-minded, innovative, conjectural and skillful at job [2]. Despite being time consuming and that is takes a lot of focus and effort, reliability of manual testing is questionable, as far as the complexity of ongoing projects in the domain, is observed. Thus, it makes almost no alternative to approve manual testing, when it cost too much without being reliable and may get error-prone and highly error-prone. While manual testing have serious implications on system quality, automated testing offer a number of advantages. Automated Testing is reliable and faster as compared to manual option. It can facilitate better regression testing of system, with considerable effort on one-time design of test suite automation. Automating the GUI testing, if implemented properly, can save a lot of time, cost and effort.

Significant benefits of automated test suite are[3]:

A. Improved quality of system.
B. Reduced testing effort.
C. Support for testing over a number of platforms.
D. Repeatedly generation of certain states of system for exploration becomes significantly easy.
E. Impact of upgraded test cases can be easily studied.

III. CHALLENGES IN TEST AUTOMATION

A number of challenges are inherent from the complexities in GUI design and test automation. Completeness, accuracy and reliability become the most crucial parameters for software quality in such a case. Few constraints are such that they totally impact the choice.

Challenges associated with GUI TEST AUTOMATION

A. Programming Language

The chosen programming language for implementation play a very important due to its dependency on compiler and third-party support. A variety of programming paradigms can make an approach far easy in one language and extremely difficult in another. For instance, a good alternative for Java based GUI may not prove as good for VB based GUI. A change in language may simply need the developer to modify whole recording structure of test suite.

B. Platform

The operating system platform would also come into play if GUI relates to some kind of system softwares or if it is .closely coupled with the underlying operating system.

C. Structural Profile

Structural profile of GUI [1] constitutes the components planted on the interface and it becomes a concern when that is dynamically changing throughout the user interaction. In such case monitoring of changes and the complex event interactions may need to be published into reports for proper tracking.

Although a number of tools for automated testing are available, there is not much significance of them for many companies.It is so because many of them are based on test scripts which are either developed or they may have been created using recorded and replayed approach, which fails to counter the impact of changing layouts [3].

IV. LITERATURE REVIEW

A number of approaches for automated GUI testing have been worked upon in recent times. L. White and H. Almezen [4] has described the user interactions be framed as Complete Interaction Sequences [4], while Z. F. Yang, Z. X. Yu, B. B. Yin, C. G. Bai [1] gives a Bayesian model for covering each-state, of the system. While CIS may be time consuming in systems where complex user-interactions are involved, Bayesian model also require the system source code to be frozen during the test execution.

V. METHODOLOGY

The methodology in this paper revolves around the effort to develop a product keeping in mind the faults that may later arise. The effort has been made to allow the system to take major share of responsibility once it is out for testing phase. Thus a development strategy is being modelled that can facilitate easier testing automation. Basically, the paper views GUI test in three perspectives as depicted in Fig. 1.

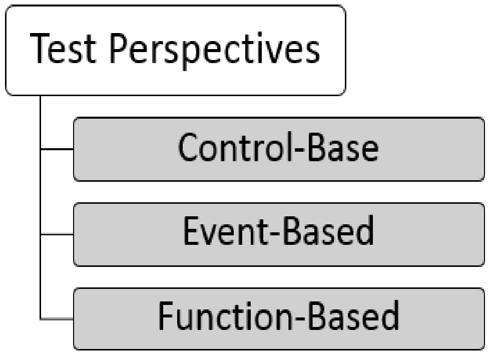


Fig.1 Three perspectives for GUI Testing

A. *CONTROL AND EVENT BASED COVERAGE*

The term "control" refers to the layout components that may dynamically change with unpredictable user interactions and event fires. As soon as a control is initialized in memory, it get registered with the "Binding Table" with certain set of attributes, that further help not only to execute automated set of tests, but also facilitate easier troubleshooting while failures. To ensure that no in-memory control is missed in the Binding Table, a provision to cross-check from the stack is employed.

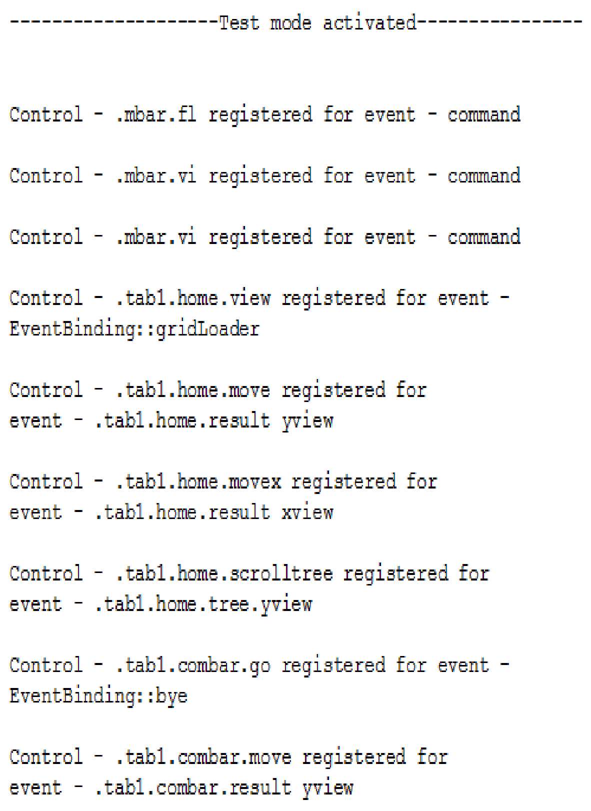
The "Binding Table" concept ensure that each dynamically added control, during interaction of user with GUI, is being in the list of testing program. Whenever an event is invoked on a control, certain function is performed. Is_update variable will act as indicator to the completion of event testing for a particular control. The only effort it adds is a single line call code to initializer function, which allocates memory and registers the controls.

Pseudo code depicting Binding Table Structure is depicted in the Fig.2.

```
class BindingTableStructure
{
public
        Table_Size;
        structure BindingTableEntry
        {
                control_name;
                registered_event_name;
                control_type;
                BindingTableEntry child_controls;
                is_update;
        };
        function registerEvents(std::string
control_name, std::string label);
        structure BindingTable
        {
                BindingTableEntry entry[1000];
        };
        BindingTable getBindingTable();
Private:
        BindingTable table;
}
```

Fig. 2. Binding Table Structure using pseudo-code

Fig. 3 depicts brief log of Binding Table contents enabled during test mode operation.



```
--------------------Test mode activated----------------


Control - .mbar.fl registered for event - command

Control - .mbar.vi registered for event - command

Control - .mbar.vi registered for event - command

Control - .tab1.home.view registered for event -
EventBinding::gridLoader

Control - .tab1.home.move registered for
event - .tab1.home.result yview

Control - .tab1.home.movex registered for
event - .tab1.home.result xview

Control - .tab1.home.scrolltree registered for
event - .tab1.home.tree.yview

Control - .tab1.combar.go registered for event -
EventBinding::bye

Control - .tab1.combar.move registered for
event - .tab1.combar.result yview
```

Fig. 3. Sample log of Binding Table contents

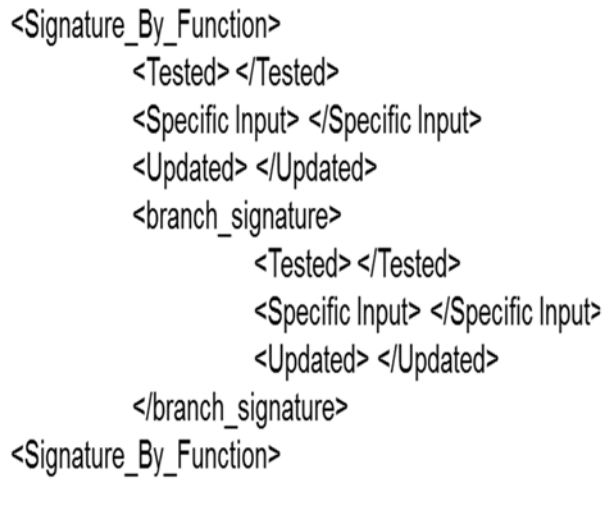
B. FUNCTIONAL COVERAGE

Functional coverage of system is ensured using a script of milestones for each function. Say 'n' functions are the features provided in the system, it will take approximately O(n) flags in the script to be coded individually. This do not need any extra effort during testing, instead it can be accommodated in the design phase as a tracker for various functions.

It is always better for a developer to keep a check on tracking requirements while preparing a design so that, a reliable structure for test plan is prepared by the time design document is finalized. It can also contribute to automated execution of test-case generated.

Fig. 4 show the xml syntax in which trackers for every function and every branch of it can be placed into a milestone-holder file. This file will later act as input to the function coverage test module. This can also be maintained as an in-memory xml to save the complexity of file operations.

Basically **dump_milestone*()*** would be responsible to generate this file.



```
<Signature_By_Function>
          <Tested> </Tested>
          <Specific Input> </Specific Input>
          <Updated> </Updated>
          <branch_signature>
                    <Tested> </Tested>
                    <Specific Input> </Specific Input>
                    <Updated> </Updated>
          </branch_signature>
<Signature_By_Function>
```

Fig. 4. XML format for milestone representation used to dump branch information.

Brief description of the fields in the syntax are as follows:

1. Signature_By_Function

It can take the value as any alias representing a milestone to a particular feature of the system. It must be ensured that each function should be tagged with a unique milestone.

2. Branch Signature

It will place the milestone at each branch of a function. Branch includes all the choice-based constructs of the function.

3. Tested

It can take the value as 0 or 1, indicating if the testing has been done after latest update or not. It will be set to 0 as soon as updated flag corresponding to it is set to 0.

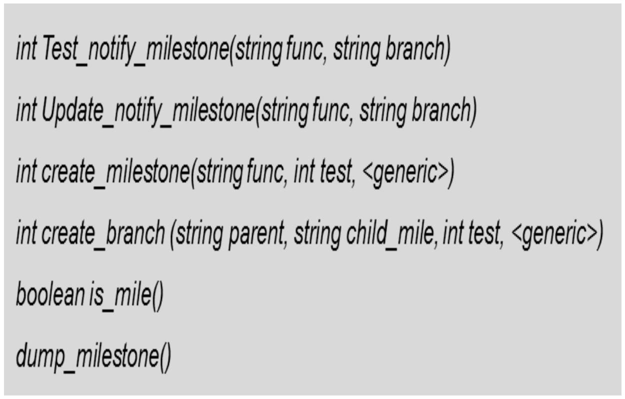
4. Specific Input

Any special inputs to the branch if required for testing can be dumped into this field. It will ease the test case generation of specific scenarios.

5. Updated

It can take the value as 0 or 1, indicating if a repeated testing of a function is required after change in system state. It automatically set to 1 if the tested option is updated.

Associated API Model for the System:



```
int Test_notify_milestone(string func, string branch)
int Update_notify_milestone(string func, string branch)
int create_milestone(string func, int test, <generic>)
int create_branch (string parent, string child_mile, int test, <generic>)
boolean is_mile()
dump_milestone()
```

Fig. 5. API model associated with Functional Coverage Mechanism

If not dumped into the file, the in-memory mile_stone_xml can also be directed into the test case generator. This mile_stone based mechanism can be controlled to be executed only in test mode using TEST_ENABLE flags.

C. TEST CASE GENERATION

Test case generator is primarily placed to automatically generate inputs for the functions, and test all the branches of it without human intervention. It is particularly meant to ensure that no control fails the system under abnormal set of inputs and no specific case of inputs is missed in the system. It thoroughly checks each function with positive and negative inputs and any specific cases for the function under test are taken from the milestone data-structure. Positive inputs can be supplied using boundary values and a random value generator functions for each data type. Negative inputs can be provided with out-of-bound values or garbage value generator to the function.

VI. CONCLUSION AND FUTURE SCOPE

A lot of efforts are being put towards providing user-friendly interfaces to complex system designs. The testing methodology and its automation is a center of focus because many of the systems that are proprietary systems of companies, are still in dire need of appropriate methods for testing. In such a scenario, we present a plan to incorporate helping structures during development phase, which can later make the system being more responsive in the testing phase, and would contribute to a scalable code for testing scripts.

REFERENCES

[1] Z. F. Yang, Z. X. Yu, B.B. Yin, and C.G. Bai, GUI Reliability Assessment based on Bayesian Network and Structural Profile, International Journal of Signal Processing and Pattern Recognition, Vol. 8, No. 1, pp. 225-240, 2015.

[2] R. M. Sharma, Quantitative Analysis of Automation and Manual Testing, International Journal of Engineering and Innovative Technology, Vol.4, Issue 1, ISSN: 2277-3754, 2014.

[3] G. M. D. Gandhi, A. S. Pillai, Challenges in GUI Test Automation, International Journal of Computer Theory and Engineering, Vol.6, No.2, April 2014.

[4] H. Almezen, L. White, “Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences”, 11th International Symposium on Software Reliability Engineering.