International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 4 Issue 06, June-2015

# Developing a Robust Framework for Test Automation

Vaishnavi S Kulkarni
Post graduate Student
The National Institute of Engineering,
Mysore India

Asha N
Associate Professor
The National Institute of Engineering,
Mysore India

***Abstract*- Manual software testing is used for verification of most software projects. However manual testing suffers from many setbacks. To improve software testing, it is important to automate the testing process. Only record and playback does not provide robustness, portability and script stability. We developed a new framework with an emphasis on modularity for test automation. This would help speed up creation of automation scripts and enable easy maintenance of scripts.**

***Keywords: Testing, Automation, Framework***

## I.INTRODUCTION

The software development lifecycle consists of many phases like Requirement gathering, Design, Implementation or coding, Testing, Deployment and Maintenance.

Ideally the allocation of time and efforts has to be equivalent or almost equivalent in all these phases. But most companies tend to ignore the testing phase leading to poor quality of software which leads to customer dissatisfaction and even at times recalling the software.

Verification and Validation are very important and a dedicated team needs to be deployed for the same. As the oft repeated definition goes "Verification tells are we building the product right whereas Validation tells are we building the right product. In this regard keeping in mind the needs of the stakeholders testing assumes an important position in reliable software development.

The increased complexity of systems as well as short product release schedules makes the task of testing difficult. One of the prime problems is that testing typically comes late in the project release cycle, and normal testing is performed manually. When bugs are detected, the cost of rework and additional regression testing is costly and further impacts the product release. Nowadays the increased complexity of software-intensive systems means that there are a potentially indefinite number of combinations of inputs and events that result in distinct system outputs, and many of these combinations are often not covered by manual testing. Thus, Automation Testing is gaining importance in the present times.

## II.LITERATURE SURVEY

It must also be noted that manual testing suffers from various disadvantages like manual testing is slow and costly. It takes a long time to complete tests.

Also, Manual tests don't scale well. As the complexity of the software increases, the complexity of the testing problem grows exponentially. Manual testing is not consistent. Variations in how the tests are performed are inevitable, for various reasons. There is also lack of training. The staff should be well-trained in the different phases of software testing like Test design, Test execution, and Test result evaluation Testing is difficult to manage. [1]

Test automation is more difficult to execute than plan if a high percentage of tests are to be automated it may be required to invest a lot of effort and costs, and it might take a long time to get there. This gets even worse when there are changes to the system under test force the team to revisit and revise part or all of automation. The testing team can end up spending more time on automation than on testing, and as a result may produce fewer test cases, thus negating a prime potential benefit of automation.[2]

Systematically and yet efficiently testing healthcare software systems is very difficult due to the following reasons: 1) Healthcare software systems are integrated, as they are aimed at supporting the integration of a wide variety of healthcare workflows Healthcare software systems often embed a large volume of special domain knowledge which is related to both healthcare financial and clinical workflows. The healthcare software industry, particularly for healthcare information management, is not highly profitable, as healthcare IT expenditures are a very small percentage (2%) of overall healthcare expenditures. [3]

## III.EXISTING SYSTEM

However it must be clarified here that only record and playback cannot be considered as reliable automation testing. ; Automated testing automates not only test case execution, but also test case generation and result verification. A fully automated testing system is able to test software without any user intervention. This cannot be achieved without building a robust test automation framework.[4]

The four main capabilities of reliable automation frameworks are:

•Testers should be able to visualize each step of the workflow and edit test cases intuitively.

• Frameworks must have capability to integrate with spreadsheets and provide powerful calculation features.

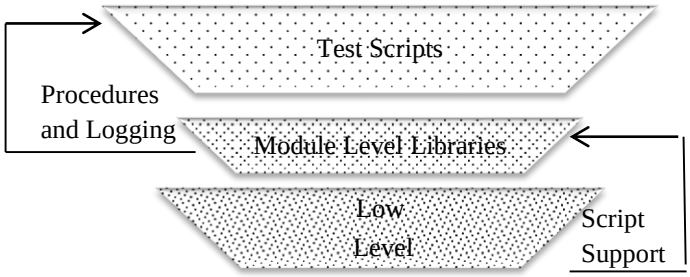
• The framework must automatically generate reports of the test run and show the results in an easy-to-read format. The reports should provide specifics about where application failures occurred and what test data was used.

• A key factor to consider while designing a Test Automation Framework is to identify if it is to be tightly

integrated or loosely coupled with the system-under-test (SUT). [5]
With this in mind we have designed a robust framework for test automation.

## III.PROPOSED SYSTEM

System Design consists of identifying the responsibilities of various participating entities and the algorithms used for generating the key identifiers.




Development of a RobustFramework for Automation

Here the development of a robust framework for development of completely automated test cases, with reporting module having strong underlying script library is presented.

*Defining a robust folder structure*

It is important to recognize the SUT requirement and define a non-changing, access friendly folder structure which efficiently stores framework components of all levels encompassing

- Components
- Modules
- Basic Script scripts
- Logging
- Modal dialog recognition and handling
- Test script maintenance

Hence we decided on a folder structure that would comprise of a parent folder consisting of various subfolders including but not limited to:

- Low Level Library consisting of pure scriptsupport and few custom actions
- Modules Tree consisting of a large number of subfolders divided on the basis of theirappearance in UI. Each of these subfolders contains the associated module level file and the component file and Unit test files (discussed shortly).
- Module level logging logs the message of each action at the module into a separate file for manual verification of test results. This makes it easier to drive forward keyword based test automation in the near future.
- Generic dialog and module dialog handling scripts

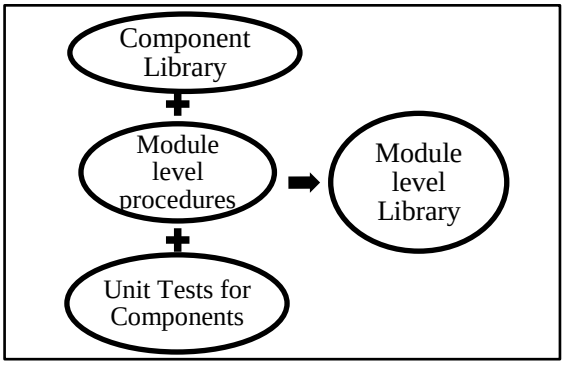
Development of the Low level library

The low level library consists of various procedures to handle every kind of event involved in the SUT. For example robust scripts are written in Script for handling events like mouse clicks, table selection, combo box selection, tab selection, image comparison, check box selection and deselection etc. based on the passing of few basic parameters like component ID and value to be selected if any.

It must be noted that each of these actions consists of logging and is parameterized for generality across clients and applications.

*Development of Module level library*

A module consists of all actions required for a particular subset of related UI and hence functionality of the given subset in relation to the SUT. The parts of the module level library are shown in the figure below:




Each subfolder consists of the following:

*Component Library*

The recorded components are transformed into Window and Component nodes which form a hierarchy that represents the actual structure of the GUI. Unfortunately, Every time a sequence is recorded, nodes are generated for components that are not yet represented. When the sequence is discarded later on, the Components remain; hence Component nodes have a tendency to proliferate in all major tools.

For Component nodes the tool based id has an important function. It is the unique identifier for the Component node by which events, checks and other nodes that have a target component refer to it. Such nodes have tool component ID attribute which is set to the Component's ID. This level of indirection is important.

If the GUI of the SUT changes in a way that the tool cannot adapt to automatically, only the Component nodes for the unrecognized components need to be updated to reflect the change and the test will run again. When creating a Component node, the tool has to assign an ID automatically. It does its best to create an expressive value from the information available.

But the same can get repetitive across various tabs, confusing and even inconsistent as the size of the component file increases. Hence it is required to have clear, concise and consistent component IDs across the GUI.

*Module level library*

A module for each functionality consists of the SUT consists of the procedures to simulate related actions.

The module thus calls from the low level library required scriptencompassing parameterized function and is called by test cases whenever required.

*Unit Test for Components*

Unit testing is the basic level of testing. Unit testing focuses separately on the smaller building blocks of a program or system. It is the process of executing each module to confirm that each performs its assigned function. The advantage of unit testing is that it permits the testing and debugging of small units, thereby providing a better

way to manage the integration of the units into larger units. In addition, testing a smaller unit of code makes it mathematically possible to fully test the code's logic with fewer tests. Unit testing also facilitates automated testing because the behavior of smaller units can be captured and played back with maximized reusability.

*Handling Logging*

It is very important to have logs of:

• Each script based action at low level
• Each procedure at module level
• Each action at test case level
• Each step at test case level

For the purpose of understanding whether the SUT is behaving as expected. The log results must be available outside the tool and also in the runlog. This is achieved by automating the creation of a file which has all details of the actions taken by the script and also the UI behavior for the same. It also details the version information and the step as well as whether the overall test case has passed or failed.

Handling Modal Dialogs

Various user guidance, error messages, warnings appear in the normal course of operation to prevent undue actions by users or to seek user input at various points or to prevent inappropriate usage of the system.

Test cases are designed to check the contents of these modal dialogs also. But sometimes they result in exceptions to the test case in unexpected manners. To prevent or contain the effect of the modal dialogs that may pop up in the case of normal or abnormal operation, these must be appropriately handled as below:

• The contents of the warning/error message/guidance must be checked for appropriateness
• The actions expected from the user should be evaluated
• The actions must be handled

These are handled by taking the parent component of the modal dialog and checking and writing Script scripts to evaluate the same.

1) Get Component ID of parent as well as child nodes
2) Compare if parent or child node belongs to class "Widget Button"
3) If it belongs to the class "Widget Button" extract text from item and store as variable
4) Click on the obtained text

## IV.CONCLUSION

It can be concluded that efficiently running tests that support the purpose of regression and confidence testing are very important as they allow the manual Q&A to concentrate on new features while itself taking care of the regular workflow, sanity and load testing.

Towards this end a modular framework has been developed and proved beneficial.

We have successfully been able to test various components in the applications by using the various methodologies a simple yet high level framework that serves the purpose in scripting test cases in reliable, fast and efficient manner.

The Script support has allowed testing of various complicated components. The availability of Java APIs has provided greater flexibility in testing the various modules of the software.

## V. FUTURE ENHANCEMENTS

Keyword-Driven Testing and Table-Driven Testing refer to an application-independent automation framework. This framework requires the development of data tables and keywords, independent of the test automation tool used to execute them and the automation script that "drives" the System-under--test (SUT) and the data.

The captured logs are planned to be used for checking the presence and expected actions using an upcoming framework that recognizes these components and drives the required (possibly rewritten to suit the needs of automation) test case forward using suitable engine to recognize the components and actions captured.

## VI. BIBILIOGRAPHY

[1] "Empirical Observations on Software Testing Automation", by Karhu K., Lappeenranta University of Technol., Lappeenranta, Repo,Software Testing Verification and Validation, 2009. ICST 2009
[2] "Reconciling Manual and Automated Testing" by Andreas Leitner, IlincaCiupa, Bertrand Meyer, Chair of Software Engineering, Department of Computer Science, ETH Zurich
[3] "Applying Model-Based Testing to Healthcare Products" by Marlon Vieira, Xiping Song et alpublished in 30th International Conference on Software Engineering, 2008. ICSE '08. ACM/IEEE
[4] "Global Software Test Automation" a book by Hung Q. Nguyen, Michael Hackett and Brent K. Whitlock
[5] "Guidelines to create a Robust Test Automation Framework" Alliance Global Services 2009
[6] "Why Model-Based Test Automation is Different and What You Should Know to Get Started" by Mark Blackburn, Robert Busser, Aaron Nauman
[7] "A Survey of Unit Testing Practices" by Per Runeson, of the Lund University published in IEEE Software
[8] "Software test and Quality assurance 5%" by Hans Buwalda published in The STP magazine
[9] "Building Your Dream Team" by Hans Buwalda tells published in The STP magazine
[10] "Automated GUI Interface Testing" by LR Kepple, DC Laroche, MH Parker published as US5781720 grant