IJITKM Volume 7 • Number 2 Jan– June 2014 pp. 214-219 (ISSN 0973-4414)

Test Automation Framework

Rajesh Popli
Manager (Quality), Nagarro Software Pvt. Ltd., Gurgaon, INDIA
rajesh.popli@nagarro.com

ABSTRACT
A framework is a hierarchical directory that encapsulates shared resources, such as a dynamic shared library, image files, localized strings, header files, and reference documentation in a single package. Framework is a wrapper around complex internal architecture which makes end user to interact with the system easily [3]. Test Automation Framework is an application-independent framework that deals with all possible actions and verifications that can be performed on an object. Therefore, the code for the same object can be used across different applications.

1. FRAMEWORK FEATURES

A short list of high level requirements or framework features which are applicable for all large scale test automation framework are described below[1]:-

- **Automatic Test Execution**: - Fully automatic test execution is of course the number one requirement for test automation frameworks. Just executing tests is not enough, however, and the framework must also be capable to for example analyze test outcome, handler errors and report results.
- **Ease of Use**: - Framework must be easy to use by test engineers or it is very likely to be abandoned. Framework users must be able to design and edit tests, run them and monitor their status easily without any programming skills.
- **Maintainability**: - It must be easy and fast to maintain both test data and framework code when the tested system changes or updates are needed otherwise. It should also be easy to add new features to the framework. The following items detail out the technical requirements that will help achieve maintainability.
- **Executing Tests Unattended**: - Framework must be able to start executing tests with a push of a button and run tests on its own. This means that framework must be able to set up the test environment and preferably also check that all preconditions are met.
- **Starting and Stopping Test Execution**: - It must be possible to start test execution manually. It is also convenient if tests can be started automatically at a specified time or after a certain event (e.g. new version of the SUT available). Easiest way to start execution at a certain time is making it possible to start test execution manually from command line and using operating system's features for scheduling (at in Windows and cron in Unixes). Starting after predefined events can be implemented similarly using external tools.
- **Handling Errors**: - Part of running tests unattended is recovering from errors caused by the tested system or the test environment not working as expected. Test framework ought to notice error situations and continue testing without manual intervention. Handling all possible but rare errors is seldom worth the effort, though, and over-engineering should be avoided.
- **Verifying Test Results**: - An integral part of test execution is verifying test results. Fewster and Graham (1999) define verification as one or more comparisons between actual outcome of a test and predefined expected outcome. They also explain different comparison techniques which are out of the scope of this thesis.
- **Assigning Test Status**: - After a test is executed and its results verified it ought to be given a status. If the test was executed without any problems and all comparisons between actual and expected outcomes match the test gets a pass status. In every other case the status is fail. Besides the status every test case should also get a short but descriptive status message. For passed tests this message is normally not that important but with failed tests it can give details about the cause of the problem.
- **Handling Expected Failures**: - The most frustrating part of failure analysis is going through test cases which are known to fail and checking whether they have failed similarly as before or have new defects appeared. That is wasteful and clearly something that should be automated as Fewster and Graham (1999) recommends. To be able to differentiate expected failures from new ones the framework must know what is the expected outcome when test fails. This means that the framework must store both the expected outcome

and the expected failed outcome of the test. When a test fails, and is expected to do so, the expected failed outcome can be compared against the actual outcome.

- **Detailed Logging: -** Test automation framework ought to give enough information to test engineers and developers so that they can investigate failed test cases. Giving only test statuses and short status messages is not enough. Instead more detailed test logs about the test execution process and how the tested system behaved are needed. On top of that the framework must log what it is doing internally to make it easier to debug problems in the framework itself. The main dilemma with any kind of logging, as stated by Pettichord (2002), is how much to log. Logging too much makes finding relevant information hard and huge log files cause a storage problem. Then again logging too little can make logs totally useless. There is no perfect solution for this problem but for example using different logging levels make the problem smaller. Multiple logging levels provide a mechanism for controlling how much detail is captured.
- **Automatic Reporting: -** Test logs have all the information from test execution but, especially if they are long, they are not good for seeing test statuses at a glance. This kind of view is provided by concise test reports. Test reports provide statistical information about the quality of the tested system and they are not only important for test engineers but also for developers, managers and basically everyone involved with the project. (Fewster and Graham, 1999) Reports should not be too long or detailed. Having a short summary and list of executed test cases with their statuses is probably enough. List of items every test report should have is presented below—adapted from Buwalda et al. (2002)—and Figure 2.3 shows a simple example.
  - Name and version of the tested system.
  - Identification of the executed test suite.
  - The total number of passed tests and executed tests.
  - List of errors found.
  - Optionally a list of all executed tests along with their statuses.
- **Use of Variables**:-Variables can be defined and used across the generated test script. This can be used to capture runtime values, which can be reused as input elsewhere during test execution.
- **Conditional Checking**: - Conditional constructs such as 'if' can be implemented using keywords to handle different flows based on various conditions.
- **Data-driven Testing**: - This framework supports data-driven testing by importing data from an external data sheet.
- **Customized Reports**: - Customized reporter messages can also be used to perform effective analysis on execution reports. These reports can be customized to display the pass or fail condition of any functionality, even during the verification of any checkpoints.
-

2. FRAMEWORK BENEFITS

Duplication of work is minimized at every level. For instance, a user might have to perform a certain action on an object of a similar class (e.g., clicking a button) repeatedly. This can be in the same test case or in a different application altogether. In both cases, the same code can be reused. Various other advantages of Test automation framework are as below:-

- **Reusability**: Being an application-independent, a Test Automation Framework gives facility to not write code whenever that framework is going to be used for a new application. The framework code (all possible actions and verification checkpoints) need to be reused and only scripting according to new application need to be done. In the way it avoids the duplicity of framework code.
- **Optimum utilization of the Too**l: The framework has the advantage of using keywords as the input for triggering an action. This well built framework uses the features of the tool effectively. For instance, QTP uses a shared object repository where all the objects required can be added and reused across the scripts for an application under test.
- **Less Effort**: While using best framework either keyword or hybrid with keyword driven feature, the effort involved in coding and reviewing is minimal when compared to other frameworks since a good percentage of coding is done within the framework. The tester simply has to enter the keywords, reducing the time required for coding. Recording is also not required as the global repository is used. The amount of rework required for migrating from one application to the other on the same platform is reduced since the code remains the same.

- **Increased Quality**: The scripts will be of uniform quality since they make use of the same code.
- **Greater Productivity**: If Open Source Test Automation Framework is used then it provides both qualitative and quantitative benefits for automation and is highly productive compared to any other framework. This framework also addresses the ongoing maintenance of the test scripts in a cost-effective manner.
- **Maintenance:** Simple modifications to the application can be easily handled in the code. The changes will be done only in the external file containing the code and the scripts need not be changed. Hence it is easy-to-maintain the scripts and provide cost-effective solution for the test automation. Framework provides a structured for test library having systematic maintenance of automation scripts.
- **No scripting skills required by the End User**: No coding skills required to automate and to review the scripts. The scripts are user friendly with good readability. Scripts can be interpreted easily by a person who does not have complete knowledge of the tool. It protects non-technical testers from the code.
- **Return on Investment is high**: Although the initial effort for building framework is high, in the long run, the return on investment will be high because of the reusability and optimum utilization of the tool.
- **Reuse of Tests**: When a certain set of test scripts require frequent execution it makes more sense to automate them and reap the benefits of unattended automated execution of test scripts. Another example will be in cases where a single script needs to be executed with multiple data sets. In such cases the effort to automate a single script and running it with multiple data sets is far less than the manual execution effort for all those data sets. The same scripts can also be used if it needs to be executed in multiple environments.
- **Time Save:** Running unattended automated test scripts saves human time as well as machine time than executing scripts manually.
- **Better use of people**: While automated scripts are running unattended on machines, testers can do more useful tasks
- **Cost Saving**: On test engagements requiring a lot of regression testing, usage of automated testing reduces the people count and time requirement to complete the engagement and helps reduce the costs.

3. TYPES OF FRAMEWORK

There are various frameworks available for automation, such as:

A. Linear Framework (Record and Playback):

It is the simplest of all Frameworks and also known as "Record & Playback". In this Framework, a Tester manually records each step (Navigation and User Inputs), Inserts Checkpoints (Validation Steps) in the first round. He then plays back the recorded script in the subsequent rounds. This method is v.easy and fast.

Advantages

- Fastest way to generate script
- Automation expertise not required
- Easiest way to learn the features of the Testing Tool

Disadvantages

- Little reuse of scripts
- Test data is hard coded into the script
- Maintenance nightmare

B. Modular Framework (Test Script Modularity Framework):

The test script modularity framework requires the creation of small, independent scripts that represent modules, sections, and functions of the application-under-test. These small scripts are then used in a hierarchical fashion to construct larger tests, realizing a particular test case. Of all the frameworks I'll review, this one should be the simplest to grasp and master. It's a well-known programming strategy to build an abstraction layer in front of a component to hide the component from the rest of the application. This insulates the application from modifications in the component and provides modularity in the application design. The test script modularity framework applies this principle of abstraction or encapsulation in order to improve the maintainability and scalability of automated test suites. This is also a general frame work that can be used by some people [2].

Advantages

- Fastest way to generate script
- Automation expertise not required
- Easiest way to learn the features of the Testing Tool

Disadvantages

- Little reuse of scripts
- Test data is hard coded into the script
- Maintenance Nightmare

C. Test Library Architecture Framework:

It is also known as "Structured Scripting" or "Functional Decomposition".The test library architecture framework is very similar to the test script modularity framework and offers the same advantages, but it divides the application-under-test into procedures and functions instead of scripts. This framework requires the creation of library files (SQA Basic libraries, APIs, DLLs, and such) that represent modules, sections, and functions of the application-under-test. These library files are then called directly from the test case script.

Advantages

- Higher level of code reuse is achieved in Structured Scripting as compared to “Record & Playback”
- The automation scripts are less costly to develop due to higher code re-use
- Easier Script Maintenance

Disadvantages

- Technical expertise is necessary to write Scripts using Test Library Framework.
- More time is needed to plan and prepare test scripts.
- Test Data is hard coded within the scripts

D. Data-driven Automation Framework

In this Framework, while Test case logic resides in Test Scripts, the Test Data is separated and kept outside the Test Scripts. Test Data is read from the external files (Excel Files, Text Files, CSV Files, ODBC Sources, DAO Objects, ADO Objects) and are loaded into the variables inside the Test Script. Variables are used both for Input values and for output (verification) values. Test Scripts themselves are prepared either using Linear Scripting or Test Library Framework. Navigation through the program, reading of the data files, and logging of test status and information are all coded in the test script. This is similar to table-driven testing in that the test case is contained in the data file and not in the script; the script is just a "driver," or delivery mechanism, for the data. Unlike in table-driven testing, though, the navigation data isn't contained in the table structure. In data-driven testing, only test data is contained in the data files.

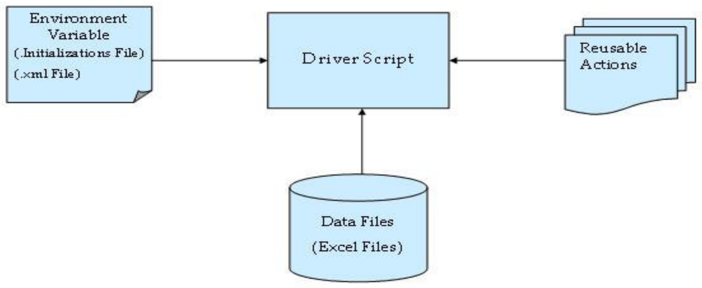

Figure 1: Data Driven Framework

Advantages

- Changes to the Test Scripts do not affect the Test Data
- Test Cases can be executed with multiple Sets of Data
- A Variety of Test Scenarios can be executed by just varying the Test Data in the External Data File

Disadvantages

- More time is needed to plan and prepare both Test Scripts and Test Data

E. Keyword-driven Automation Framework

Keyword-driven testing and table-driven testing are interchangeable terms that 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 test script code that "drives" the application-under-test and the data. Keyword-driven tests look very similar to manual test cases. In a keyword-driven test, the functionality of the application-under-test is documented in a table as well as in step-by-step instructions for each test. There are 3 basis components of a Keyword Driven Framework viz. Keyword, Application Map, Component Function. Keyword is an Action that can be performed on a GUI Component. Ex. For GUI Component Textbox some Keywords (Action) would be Input Text, Verify Value, and any other action/function which can be taken/perform and Verify Property and so on.
An Application Map Provides Named References for GUI Components. Application. Maps are nothing but "Object Repository' which maps each object to its property.
Component Functions are those functions that actively manipulate or interrogate GUIcomponent. An example of a function would be click on web button with all error handling, enter data in a Web Edit with all error handling. Component functions could be application dependent or independent.

Advantages

- Keywords are re-usable
- Provides highest code re-usability
- Test Tool Independent
- Independent of Application Under Test
- Maintenance is low in the long run
- Test cases are concise, readable for the stake holders and easy to modify
- Keyword re-use across multiple test cases
- Not dependent on a specific tool or programming language
- Test case construction needs stronger domain expertise - lesser tool / programming skills
- Keyword implementation requires stronger tool/programming skill - with relatively lower domain skill
- Abstraction of Layers

Disadvantages

- Initial investment being pretty high, the benefits of this can only be realized if the application is considerably big and the test scripts are to be maintained for quite a few years.
- High Automation expertise is required to create the Keyword Driven Framework.
- Longer Time to Market (as compared to manual testing or record and replay technique).
- Moderately high learning curve initially as compared to manual testing.

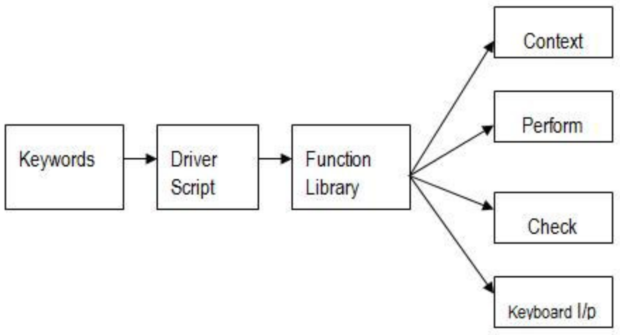

Figure 2: Keyword Driven Framework

F. Hybrid Automation Framework

As the name suggests this framework is the combination of one or more frameworks discussed above pulling from their strengths and trying to mitigate their weaknesses. This hybrid test automation framework is what most frameworks evolve into overtime and multiple projects. Maximum industry uses Keyword Framework in combination of Function decomposition method.

4. CONCLUSION AND FUTURE WORK

For a functional automation, a framework helps a tester a lot as studied in the second and third section of this report. So when a tester thinks of automating its manual scenarios they must go for a framework. But before selecting a framework they should know the pros and cons of every framework. Otherwise it could be more costly than manual testing. As different languages are evolving in time, in future more complex framework can be expected which can help tester in automating every kind of script and with these features as basic framework functionalities.

References

[1] Pekka Laukkanen "Data-Driven and Keyword-Driven Test Automation Frameworks", Espoo, February 24, 2006.

[2] http://www.onestopqa.com

[3]http://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WhatAreFrameworks.html