Frameworks for Test Automation

Mike Kelly

First Published on SQAtester.com

Basing an automated testing effort on using only a capture tool has its drawbacks. Running complex and powerful tests is time consuming and expensive when using only a capture tool. Because these tests are created ad hoc, their functionality can be difficult to track and reproduce, and they can be costly to maintain.

A better choice for an automated testing team that's just getting started might be to use a test automation framework. In this article I'll attempt to shed a little light on a handful of the test automation frameworks I'm familiar with — specifically, test script modularity, test library architecture, keyword-driven/table-driven testing, data-driven testing, and hybrid test automation. No framework is better or worse, they each are powerful in their own respective application domains.

The 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.

This framework should be the simplest to grasp and master, as 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.

Properly implemented this framework yields a high degree of modularization and adds to the overall maintainability of the test suite.

The Test Library Architecture Framework

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 that represent modules, sections, and functions of the application-under-test. These library files are then called directly from the test case script.

This framework also yields a high degree of modularization and adds to the overall maintainability of the test suite.

The Keyword-Driven or Table-Driven Testing 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.

As an example if we were to map out the actions we would perform while testing the Open function in Word, we could create the following table. The Window column refers to which application window we are performing the mouse action on. The Control column refers to the type of control that the mouse will be clicking. The Action column is the action taken with the mouse (or by you the tester). And the Arguments column is there reference a specific control.

Window	Control	Action	Arguments
Word	Menu		File, Open
Word	Pushbutton	Click	Folder Name
Word	Pushbutton	Click	Folder Name
Word	Pushbutton	Click	Folder Name
Word	Pushbutton	Click	Open
Word		Verify Result	
Word	Menu		File, Close

These tables can be made as needed in order to represent a series of tests. Each table should represent one complete test. Once you've created your data tables, you simply write a program or a set of scripts that reads in each step executes the step based on the keyword contained the Action field, performs error checking, and logs any relevant information. This program or set of scripts would look similar to the pseudocode below:

```
Main Script / Program
    Connect to data tables.
    Read in row and parse out values.
    Pass values to appropriate functions.
    Close connection to data tables.
Menu Module
    Set focus to window.
    Select the menu pad option.
    Return.
Pushbutton Module
    Set focus to window.
    Push the button based on argument.
    Return.
Etc…
```

From this example you can see that this framework requires very little code to generate many test cases. The data tables are used to generate the individual test cases while the same code is reused.

The Data-Driven Testing Framework

Data-driven testing is a framework where test input and output values are read from data files (ODBC sources, cvs files, Excel files, DAO objects, ADO objects, and such) and are loaded into variables in captured or manually coded scripts. In this framework, variables are used for both input values and output verification values. 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.

This framework tends to reduce the overall number of scripts you need in order to implement all of your test cases, and it offers the greatest flexibility when it comes to developing workarounds for bugs and performing maintenance. Much like table-driven testing, data-driven testing requires very little code to generate many test cases. This framework is very easy to implement using most any toolset, and there's a lot of detailed documentation available with how-tos and examples online.

The Hybrid Test Automation Framework

The most commonly implemented framework is a combination of all of the above techniques, pulling from their strengths and trying to mitigate their weaknesses. This hybrid test automation framework is what most frameworks evolve into over time and multiple projects.

You can implement modularity by nesting test scripts and using the library files/object to implement functions, procedures, or methods. You can implement whichever data-driven technique you choose and to the extent you find it useful. The trick is to use the best framework(s) for the job and the only way to figure that out is to jump in and start using them.