

Test Automation Frameworks – Build or Buy?

Sam Warwick, Odin Technology Ltd

Introduction

It is now widely accepted within the test automation community that the most effective way to implement automated software testing is via some kind of ‘framework’ approach. This could mean anything from developing a simple data-driven architecture to implementing one of the ‘new-breed’ commercial solutions such as Axe from Odin Technology. The purpose of this paper is to set out some of the questions to consider when deciding whether to invest in developing a proprietary in-house framework.

It should be remembered that any kind of home-grown solution is effectively going to be a software development exercise. This means the same time and resource should be given to the activities as for developing the system under test itself i.e. design, specification, code, test, implementation, documentation, support, maintenance and training. The key question here is do your testers have the time or expertise to do this? Would their time be better spent writing tests and not code? Even if the time and skills are available it is highly unlikely this approach would be any cheaper than buying an off-the-shelf solution. In most cases it is actually considerably more expensive to ‘build’ than ‘buy’.

Design Decisions

Single or multiple GUI tool

Will the framework be required to support more than one GUI testing tool? Some vendors have more than one GUI tool and typically these use different scripting languages that are not compatible. Also, many large organisations currently use GUI tools from more than one vendor. Even if you are only currently using one tool, can you be sure that you will not need to change or upgrade in the future?

Does the framework need to support non-GUI testing

Many applications need to be tested at the component level. For example APIs, databases and Web Services. Service Oriented Architectures (SOA) are becoming increasingly common.

Business process oriented – can it be used by non-technical testers

The best test resources in an organisation typically do not have the high level of technical expertise required to use automated testing tools. An approach which empowers testers of all levels may be desirable.

Should the main interface be industry standard or proprietary

One of the most important components of the automation framework is the interface. Using a standard interface like Excel will make it easier for testers to learn and use the

framework. A proprietary interface will increase the development cost of the framework and make it harder for testers to learn.

Script or model-based

Will the design be just data-driven scripts or will a testing model be used? Can tests be sub-divided into re-usable components e.g. 'Login'? Some form of functional decomposition will simplify test design and reduce maintenance costs.

How will tests created in the framework be documented

A classic problem with automated tests written in scripting or programming languages is that they are hard for non-technical people to understand. The framework should make provision for easy test documentation so that the purpose of test assets is clearly understood by all testing stakeholders.

How will the results be presented and how are they stored

The most important output from the framework is the test results. Results need to be presented in a way that is easy for testers to diagnose and for management to interpret.

Does the framework need to interface with any test management tools

Many organisations may already have a test management tool in place. Ideally results from all manual and automated tests should be collated in a central point. This may require importing results from the framework into a bespoke or third-party test management solution.

Can tests be written before the application is available for testing

One of the main limitations of GUI capture/replay tools has always been that tests cannot be developed until a working application is available. This creates a bottleneck in automated test development and makes it difficult to employ an Agile methodology. A well written framework should allow the tests to be defined as soon as the application specifications are made available.

Is the framework going to have some kind of keyword element

How many keywords will there be and what format will they take? Too many can result in too much complexity and can lead to 'scripting in tables'.

Will data-tables be pre-checked/compiled or interpreted at run-time

If there are any problems with the design and structure of the test tables then it is desirable to know this before the tests are executed. Otherwise failures and problems may not be discovered until well into the test run and valuable time will be lost.

How will the test assets be version controlled

A good framework design will lend itself to easy integration with a version control system. Both the tests and the framework itself will need to be controlled.

How easy will the test assets and system be to maintain

Ease of test maintenance is one of the most important factors in the test design. The maintenance overhead should be as low as possible, allowing automation assets to

keep up with the pace of change in the application under test. For home-grown frameworks the on-going maintenance requirement of the framework itself is a key consideration. The resource required for this should not be under-estimated.

What programming language will the system be developed in

Is the framework going to be developed in the scripting/programming language used by a particular tool, or will an industry standard, tool independent language be used? If the framework is implemented using a particular tool then this could severely restrict the growth of the framework in the future. There may also be performance considerations as many test tools use interpreted scripting languages.

Types of Framework Test Model

There are a number of common framework test models. It is a good idea to have an awareness of these when deciding on the choice of framework, regardless of whether a home-grown or commercial choice is made. Unfortunately there is currently no standardisation within the industry on these terms so often the same term may be used to refer to a very different model.

Data-Driven

Scripts are coded for each test case but data is stored in external tables. Sometimes the data may also control the flow and behaviour of the test to some degree.

Action-Based

Individual business *components* (e.g. 'Login', 'Order Product') are scripted and driven from a table or business process oriented architecture.

Keyword

The main functions of the test tool language are abstracted to more business focused terms e.g. 'InputText', 'VerifyValue'. Often there are so many keywords that the tests are really just scripts in tables. Extensive scripting code is required to implement the keywords. This is one of the most ambiguous terms used in test automation.

Scriptless

No test scripts are required for any part of the end-to-end test scenario. A small number of supporting script *functions* may be required for some complex cases. Tests may be defined in simple tables or a proprietary UI. A scriptless approach will often be combined with some form of keyword implementation.

Summary

Ideally testers should write tests not code. This means avoiding the bespoke development of home-grown frameworks and adopting a commercial solution that uses a simple and entirely scriptless approach. Axe employs a totally table-driven scriptless test model with simple keywords defined in Excel spreadsheets.

For more information on the Axe Test Automation Platform from Odin Technology please visit www.axetest.com and www.odintech.com.