BEGINNER'S GUIDE TO TEST AUTOMATION FRAMEWORKS

Testing frameworks are vital to any testing process that's automated – and that you want to succeed. Before any application or software can be deployed, proper testing must occur to make sure it works the way it's meant to.

Testing used to be an entirely manual process, but as automation continues to gain traction, there are many ways to test an application in a more automated way: you may need to set things up ahead of time, but once it's up and running, the testing can become more efficient, taking up a lot less time before release.

Automated testing can be overwhelming on its own. Luckily, several frameworks exist to help your QA team adjust to and incorporate test automation. The benefits of using a test automation framework, or more than one as appropriate, is to help reduce maintenance costs by reducing the resources spent on testing. Plus, successfully employing frameworks can provide a higher ROI for both development and QA teams that are seeking to optimize and harness agile process.

This article aims to provide some insight into the most common types of test automation frameworks.

Defining a test framework

Any testing framework should provide guidelines or rules that help you create and design test cases. Frameworks should be systematic and allow the user to achieve the desired result – which, in

this scenario, is automation in testing. Ideally, the practices and tools that accompany such a framework should improve the lives of QA professionals by making their testing processes more efficient.

Here are some examples of what may be included in such guidelines:

- Coding standards
- Object repositories
- Handling methods for test data
- Processes for saving and storing rest results
- Resources for external assistance

Testers can certainly script and record tests without using such a framework, but these rules often help organize the testing processes and provide additional benefits.

Benefits of employing a framework for test automation

The goals of using a test automation framework is to increase the QA team's speed and efficiency, improve the accuracy of tests, and reduce risk, which altogether help reduce the cost of test maintenance costs.

A more efficient testing process also provides the following benefits:

- Minimizes manual intervention,
- Maximizing test coverage
- Creating code that is reusable

Popular automated testing frameworks

There are a range of frameworks for automated testing. The following sections outline some of the most popular and common types.

Linear automation framework

A linear automation framework is a basic, introductory level testing framework. It's sometimes known as a linear scripting framework or a record and play back framework.

It works like this: testers create and write test scripts sequentially, performing it individually for each individual test case, hence the record and play back. It is best suited for small-sized functions or applications. It can also be used to test scripts on the fly, with little planning or time.

The main benefit of this framework is that you can test a case quickly, as long as it's not too complicated. Its workflow is quick and easy. There's no need for testers to write custom code, so they don't need to be experts in automation to use this framework. Most testing tools include a record and play back feature, so you don't need to plan ahead much.

Still, there are drawbacks: the script can't be reused, because the hard-coded data is built into the test script, so it cannot be run again with multiple data sets. Because it is best suited for small-scale testing, this version isn't great with scalability or longer-term maintenance.

Modular-based testing framework

Also known as a modular testing framework or a modularity framework, this option takes a test case and breaks it down into smaller modules, so that each test is small and independent. Therefore, testers are creating individual test scripts for each module.

Once the modules are broken down, testers can also combine the individual test scripts into a single, master test script for particular test scenarios. These master sets can then represent and be reused for a range of test cases.

A major benefit of this framework is that scripts can be saved and reused – once a tester writes a function library, scripts can be stored in it.

It is easy to make changes to the application within this framework because you only need to make changes in the individual test script without adjusting the entire application. Because various modules can be reused, creating test cases often takes less effort and time.

Data remains hard-coded into the test script, due to the isolated execution of the tests, which means tests still cannot use multiple data sets within a modular framework. This framework also requires testers have programming knowledge in order to build the framework.

Library architecture testing framework

Derived from a modular framework, a library architecture framework offers a few more benefits. Like the modular testing framework, this one employs an architecture that values modularization, so you'll benefit from easier test maintenance and scalability, which will reduce the time and money spent on testing. But the modularization works differently: instead of breaking the application into various scripts, the framework identifies similar tasks within the scripts and groups them by function. So, the modular parts aren't about functions directly, but about common objectives.

Then, a library stores the functions, sorted by objectives, and the test scripts can call onto the library for certain functionality as needed. This library enables a lot of reuse and ease of testing across multiple test scripts.

Still, there are drawbacks. Data is still hard-coded into the script with this framework, so any changes to data mean you must change the scripts. The writing and analysis of common functions within scripts still require technical expertise, so certain team members are better suited than others. The hard-coding and development of scripts means this framework isn't for entry-level testers, and it will take more time to develop test scripts in general.

Data-driven framework

A data-driven framework actually separates the test scripts logic away from the test data. This allows testers to store data externally, such as in spreadsheets, databases, XML files, etc., but the real goal is to create test scripts for reuse that can test different sets of data. Because of this goal, test data cannot be hard-coded into the script itself, as in frameworks described above.

Setting up this type of framework takes time: the tester must store the data, then pass the parameters from the test scripts to the data source. The tester can then connect the data source directly to the script, which can read and populate necessary data. Each function of the data is laid out in a table, with a series of sequential instructions for how the test must be run.

The pros of this framework are important: tests can be reused across many data sets. This allows for fewer scripts, as you can change the scenarios being tested simply by varying the data. (The more data sets, the better!) Because the data isn't hard-coded into the script, you can simply make changes to the script as needed instead of also addresses the data set. All this combined means you're testing faster, so you can execute more quickly and efficiently.

The drawbacks of this version are clear: this is a more complicated framework, so the testers should be experienced and fluent in many languages. Setting up the initial framework takes a lot of time – that may be recouped down the road.

Keyword-driven framework

This framework is also known as table-driven testing or action-word based testing. Like a data-driven framework, a keyword-driven framework separates the test data and script logic but takes this even further. In addition to externally stored data, keywords are stored in a separate location. These keywords become part of the script, associated with actions to test the GUI – as simple or as complex as need be. The keyword table, like the data table, organizes each step sequentially, associating with a specific object or the part of the UI where the action is performed.

An object repository must be created in order to map objects with the correct actions. Once the table is established, testers write code to prompt the necessary keyword-based actions. During the testing, the test data points to the correct keyword, which triggers the appropriate script.

Less scripting knowledge is needed in this framework than in the data-driven framework. This framework also values re-usability, because one keyword can be used across many test scripts and test scripts can be independent of the current application.

Still, like data-driven frameworks, the initial set up time is significant and complex, as testers must employ object repositories, data libraries, and keyword tables. As the need for scaling increases, you must continue growing the keyword tables and repositories.

Hybrid test automation framework

It makes sense that as more systems and applications become integrated and interconnected, their testing processes would as well. A hybrid test automation framework, then, combines parts of any testing framework in order to harness advantages while mitigating the inherent weaknesses.

As teams continue the shift towards agile models, a hybrid framework may provide the necessary and helpful flexibility – the more you can adapt it for your environment, the better results you may achieve, much more efficiently.

Choosing a testing framework

The many options for automated testing tools means it may feel overwhelming to choose just one. There are various open-source tools on the market, and some specialize in certain types of frameworks whereas others work within a given programming language.

If you're new to them, experts recommend starting with a tool that offers a lot of out-of-the-box solutions, so that you begin improving your testing processes quickly. Seek one that supports many types of applications and languages – even if you typically only rely on one or two languages now,

you may soon be relying on others.

After you've chosen a framework to get started with, here's one suggestion: don't reinvent the wheel. If your framework requires some customized code, stop and check for a library or framework that may already exist – these can work right away or may only require some smaller tweaks to get the scripts you need.