**A Journey into Test Frameworks**

With the creation of the Development Product Team and quality being at the forefront of this project it was paramount to select a testing framework and infrastructure best suited to the project. The main criteria to take into consideration when selecting a testing framework are; **multiple browser** support, **mobile browser/OS** support, **continuous integration** support, **community** support, **embedded keywords** support and **readability**.

**1. Geb and Spock testing Framework**

Geb is based on the Groovy language and can be utilized with Spock, JUnit, TestNG and Cucumber. Geb used in conjunction with Spock provides the most effective combination allowing for tests to be written using the simple Gherkin Syntax allowing the test to be well structured, as can be seen in Figure 1.

Figure 1, Geb and Spock example

Another benefit of using Geb along with Spock is the page object pattern support, allowing a page to be described as an object and Geb will wait for the element specified only once the page has loaded.

Figure , page object

Figure 3, import and use of page object

After looking into Spock’s GitHub repository it was found there were numerous issues/outstanding pull requests, suggesting the support provided is not at the standard preferred. <https://github.com/spockframework/spock/issues>

**2. Gauge**

Guage is relatively new and follows the flow of defining a specification file along with a class file. The specification file displaying clearly the steps of the test being undertaken and the class file containing the exact commands required to execute each step.

Figure 4, Gauge specification file example

Gauge allows for spec files to be broken down in to concepts file (.cpt) for better understanding of a tests flow.

Figure 5, Gauge concept file example

Figure 6, Gauge class file example

A benefit of Gauge is large sets of test data can be imported from text and csv files and so minimizing the time spent during the test run of creating test data.

**3. Protractor**

Protractor testing framework is predominantly used to test AngularJS apps. One major advantage of Protractor is the automatic waiting it provides and so when writing/running a test no concern needn’t be given about waiting for the test and the webpage to sync. This is a major advantage as many testing frameworks suffer from this issue and many workarounds have to be put in place.

However if an app is not developed fully in AngularJS when using Protractor the automatic wait can be a hindrance and workarounds are required; similar to those required in other testing frameworks.

Protractor tests are written in Javascript and can follow the page object pattern.

Figure 7, protractor test following page object pattern

Figure 8, protractor test following page object pattern

Figure 9, protractor test following page object pattern

**4.Cucumber**

The Cucumber testing framework is written in a behavior-driven development style using the simple syntax language Gherkin. Cucumber is written in Ruby can be written in Ruby along with many others programming languages.

Figure 10, example Gherkin script

When used in conjunction with Capybara, a library written in Ruby, embedded keywords can be used to easily simulate how a user interacts with an application.

Figure 11, Capybara with Cucumber example

**5. Robot**

Robot framework utilizes the keyword-driven testing approach and follows acceptance test-driven development. Test libraries, written in Python or Java, can be extended with users able to extend these further or create their own.

Figure , Robot framework test steps example

Figure , Robot framework keyword example

**6. Intern**

Tests are written in JavaScript, it is best for testing JavaScript code but can test other programming languages. Intern’s execution model is well-suited to those that follow a test-last development approach and want to prevent regressions using continuous integration.

Intern always uses [Promise](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise) objects whenever an asynchronous operation needs to occur. All [suite](https://theintern.github.io/intern/#suite-object), [test](https://theintern.github.io/intern/#test-object), and [reporter](https://theintern.github.io/intern/#custom-reporters) functions can return a Promise, which will pause the test system until the Promise resolves.

Figure , Intern example script

Conclusion

Figure 15, testing framework comparison

Figure 15 displays the main criteria discussed earlier along with others which need to be taken into consideration.

For multiple browser support, mobile support via Appium (Appium drives iOS and Android apps and is an open source test automation framework for use with native, hybrid and mobile web apps) and continuous integration support all the testing frameworks investigated are able to cover these criteria.

With regards to community support, this will provide huge assistance when creating the automated tests. With little or no community support this is evidence the testing framework at some point may not updated in parallel with browsers, meaning the testing suite will become obsolete. This applies to Geb, Guage and Protractor, to an extent.

The ability to run failed tests would save much needed time at runtime of an automated suite instead of running a full suite again due to false failures. This is only available on a select few frameworks, Intern, Robot and Cucumber.

Use of a Gherkin or readable syntax would aid speed at which automated tests can be wrote, reviewed and maintained. Also depending on the stakeholders involved a Gherkin or readable syntax may have to used. Currently Geb, Cucumber and Robot offer Gherkin or readable syntax.

Test data import support depends on the project, only if large sets of test data are required to be imported is when this would be advantageous.

Asynchronous testing frameworks can be advantageous but from experience it was also found they can also be a hindrance. On most occasions the steps within a test are dependent on its previous step e.g. create user via API, login with user etc.

Other items needing to be taken into consideration, which are not easily measured, are maintenance of code and ability to calculate test coverage.

Taking all the above into account the Robot testing framework proves to be the most effective to use to fulfill an automated tests ultimate role, test a system/app and provide feedback. The main advantages being the readability of the test scripts, the ease and speed of creating a test script, the amount of embedded keywords and the fact it is not an asynchronous framework. However the ability of the framework to automatically synch the test with the webpage is a great feature Protractor offers that Robot does not there. Robot offers many embedded keywords to use instead e.g. ‘Wait Until Page Contains Element’.