### **Test Automation Framework**

A “Test Automation Framework” is scaffolding that is laid to provide an execution environment for the automation test scripts.

Framework is a constructive blend of various guidelines, coding standards, concepts, processes, practices, project hierarchies, modularity, reporting mechanism, test data injections etc. to pillar automation testing. Thus, the user can follow these guidelines while automating application to take advantages of various productive results

**The framework should be scalable and maintainable.**

**Advantage of Test Automation framework**

* Reusability of code
* Maximum coverage
* Recovery scenario
* Low-cost maintenance
* Minimal manual intervention
* Easy Reporting

**5 Generic Test Automation Best Practices for UI, API or Any Other Kind of Test Automation:**

**1. Test Class and Method Name and Variable (Camel Notation)**

All Test classes should have the same name of the class that will be checked followed by the “test” word.

Such as this:**ClassNameToBeCheckedTest; 0** (first letter should be capital)

Method should be like: **methodName()** (first letter should be small)

Variable should be like**: int countName=0** (first letter should be small)

**2. Test Folder and Source Folder**

All Test classes should be saved inside a folder called “**test**” and it MUST be a kind of mirror of the source folder, meaning it will follow the same structure of the main project folder, but it will only contain tests; For example:

Main source folder **= Project A > src > main > java**

Main test folder **= Project A > src > test > java**

**3. Setup Information**

Any kind of setup should be done by the @Before method, or some similar annotations. So the framework being used will know these actions need to be executed before performing the automated tests.

Information examples: type of browser, which URL should be opened, which timeout should be respected, should the browser be maximized or not and so on.

**4. Test Data Builder Implementation**

With Test Data Builder, all the data you will need to create and manage during your tests, can be created in a separated class to be reused in the future.

For example: product’s creation with size, price, market etc.

**5. Which Tests to Automate?**

When following test automation best practices, one should understand fully what to automate. Although it depends on the kind of test automation project you’re working on, these are general guidelines to follow:

**Visual Regression Testing:**all possible visual scenarios related to CSS or not, such as layout structure (size and position), colours, font size, etc.

**Static Content:** checking title, policy terms, logo and other types which rarely change.

**Links**: the most important links to check are broken and redirection links.

**Form Fields**: test input and returned data, fields rules (number of characters or right type of characters) submit/cancel operations, text fields, drop down, check boxes (\*Note: pay attention, because usually it involves multiple pages).

**Multiples Pages**: using form fields, such as, submit/cancel buttons, or next/back steps, involves multiple pages/tabs and the data consistence in between them should be checked.

**Data Integrity:**the data added is stored correctly in the data base, the data returned is the right one. For example: fill up the zip code and the address will be filled automatically, or if you have a product list in a dropdown element and two more fields related to this dropdown, such as size and color. As soon as you select the product in the first dropdown, the rest should be reloaded only with the related size and colour.

**Session and Basic Security**: the user session is expiring respecting the correct amount of time, the browser back button can’t access the previous restricted page, or the password, login, credit card fields are under https and restricted fields? My intention is not penetration testing or any other fancy security testing here. But, only checking if the expiration time session defined in the cookie is being respected, or if the browser back button being blocked in pages that it should not be?

**Dynamic** **Elements and Ajax:** pages that are generated with different tag IDs, names or values given by the server. Probably are page results based on user action and the ajax inside the pages. Meaning, these small parts are updated without reloading the page.

**9 Test Automation Best Practices, Specifically for UI Automation:**

**1. Assert vs. Verify (Using Junit or TestNG)**

**Asserts:**will break the test and give an immediate response, as soon as the test fails and will not perform any other action.

**Verify:**will continuous your tests, executing the other commands, even with a fail result.

Deciding which one to use depends on the case.

**2.**[**Web Elements Location Strategy**](http://blog.testproject.io/2017/02/09/inspect-web-elements-chrome-devtools/)**(e.g. Using Selenium)**

**Id / Name:** these parameters are easy, efficient, increase performance and readability.

**XPath:** although slow in some browsers, sometimes it’s the only way to get an object. It is also a good option if you need to ensure that some object must appear after another, like div//a;

**Link Text or <a>:** efficient, good performance, but take care if the text changes too often.

**Dynamic Elements / AJAX:** these elements are generated by the server and, normally, the id/name changes each time the page is opened, so the best way is use the XPATH to map them.

**P.S.: Mapping Elements Using Custom Attributes Created by the QA Team**

Using IDs or NAME attributes as much as possible should be the reliable and fast way, but it always depends on the developers to insert these attributes in the code and it is known that they forget to use them many, many times. So, why not create a custom attribute and leave it empty to be filled by the QA engineers?

The only thing the development guys should do is refactoring the entire code adding the empty attribute and the test engineer should be able to fill it in his own environment and perform the test. Imagine the custom attribute called “id-for-qa-test”; then the page code should be something as the following:

<html>

<div id-for-qa-test="start-date">

<br>Start Date</br>

<span>Month</span>

<select>

<option id-for-qa-test="jan">Jan</option>

<option id-for-qa-test="feb">Feb</option>

<option id-for-qa-test="mar">Mar</option>

</select>

But the idea here is not to have a value defined by the development team. The development team will just add the parameter with empty values (id-for-qa-test=””) and the QA team before running the tests will fill up the parameters with their respective desired values.

The test automation code will use the **findElement(By.xpath([contains(@id-for-qa-test, ‘VALUE’)]))**.

**3. Wait**

**DON’T** use Thread.sleep, because it will add many problems to your test, starting from increasing the run time.

**Wait or FluentWait:** You will get more robust, deterministic and in case of element not found, the exception will be clearer.

**You can use something such as:**

until(ExpectedConditions.elementToBeClickable OR

 wait.until(ExpectedConditions.urlMatches…  OR

wait.until(ExpectedConditions.titleContains…  OR

wait.until(ExpectedConditions.urlContains…  OR

 wait.until(ExpectedConditions.titleContains

**4. Don’t Use a Specific Driver**

**Using** Parameter notes you can easily handle different browser types and prepare your test for [cross-browser and parallel execution](http://blog.testproject.io/2017/02/09/cross-browser-testing-selenium-webdriver/).

**Using JUnit** you have @RunWith(Parameterized.class) with @Parameters (browser).

**Using**[TestNG](http://blog.testproject.io/2017/04/03/testng-tutorial/) you have @Parameters(“browser”) with XML configuration.

**5. Taking Screen Shots and Logs from HTML Sources**

It will easily help the investigation in case something goes wrong. Basically, you have to create a kind of Watcher/ Listener for your tests.

In **JUnit** you can use @Rule.

In TestNG you can use extends TestListenerAdapter and use a XML configuration <listener> and org.testng.Reporter.log

**6. Making Your Own Set of Methods (Wrapping Selenium Calls)**

This is often used to speed up the code and easily reuse repeatable code pieces, such as: click on this and wait for that. So, just create a method that will click on something and wait for another thing and call this method every time you need this.

The example above is frequently called “click and wait actions” or the “**clickAndWait** method”.

You can improve your Wrapping Selenium calls including many other commands. For example, adding a verification to check if the element is still available before clicking on it and then, wait the page. But be careful, to not include too many Selenium calls at the same methods.

**7. UI Mapping**

To map the user interface could be a very efficient way, not only to create and perform automated tests but also to maintain them. You have all the information stored and organized in one place and if anything changes in the front end or HTML or CSS etc., you just need to update one single file. So, you can easily manage your tests.

 You can perform UI mapping in a variety of ways. For instance, creating a class or structure to store the element’s name with its locator / id OR a property file with key/value pairs OR a JASON, YAML and so on.

**8.**[**Test Design Pattern or Page Object Designer Pattern**](http://blog.testproject.io/2017/02/09/page-object-pattern-advantages-implementation/)

It is an object-oriented class that serves as an interface for a page of your AUT.

It enhances test maintenance and reduces code duplication.

Page Objects should never make verification or assertions, but there is a single verification which can and should be within the page object. This verification mission is to verify the page, if critical elements on the page were loaded correctly, a good way to do so is checking it in the constructor of the page object. A page object does not necessarily need to represent an entire page, it could be used to represent components on a page.

**9. Data Driven Testing**

To use the same test with different inputs due to test a considerable variety of scenarios without code changing.

There you have it! By implementing as many of these test automation best practices as possible, you will immediately notice the positive impact on your automation testing workflow!

**Ref site:**<https://blog.testproject.io/2017/04/16/test-automation-best-practices/>

<https://www.blazemeter.com/blog/top-15-ui-test-automation-best-practices-you-should-follow/>