**How to avoid false positives and false negatives in test automation**

While building any test automation framework and designing scripts at times we might end up having flakiness in scripts if the proper standards and the test data generation is bad. Given the fact that in test automation framework results are vital as they state whether the test passed or failed and ultimately that concludes the stability of the application under test. Holistically result reports are the ones which help the business stakeholders to understand where we are with respect to the software product launch.

In this blog we are going to talk about the importance of the results and how should we ensure we don’t fake them. False results are capable of creating a big mess in the system and they are quite costly to fix and learn.

Having said about the fake results, let’s understand what we are meaning to say by the word fake

It can be of something that having a bug in the software system but the test shows its passed. By the term of medical science this can be addressed as false positive, because we have seen a pass result that is not actually true

On the other hand there can be a scenario where the functionality in the system could have been in working state, however the test shows it failed. This behavior being in line with medical science term called as false negative.

**Relation between automation testing and false positive/negative**

Glad that we understood a couple of new terms, but it is still a question as what’s the relationship between them and the automation testing? why only applicable to automation testing?

If we just recall the purpose of automation, it is essentially to script the test case and execute it in future so that the manual effort spent in reduced. So when we are scripting it for future use high degree of care should be taken and get the things right. The reason for insisting the promptness and care is we are not going to be visiting the script thoroughly every now and then. If at all due to oversight something wrongly implemented it obviously serves the wrong results and ultimately leads to flakiness it could be either false positive or false negative

And to answer the question why it doesn’t applicable to functional manual testing, there user applies his sense than some automated process, any wrong input gets corrected during the execution itself hence there is no room for this concept in manual testing.

**Importance to look after false positive/negative**

As briefed above in a very high level that these false positive and false negative results can cause a great loss in the business as we are purely mistaking the results. It could impact in the following ways

A false positive can lead to bug leak in the market as it was pursued to be working good at our sight, this can greatly impact the reputation and business loss for the company

A false negative can lead to spending extraneous effort by various people because of the reported problem when it’s not true. The bug has to be initially understood then learn to repeat then spend efforts to triage the same to conclude it. These efforts ultimately affect the deadline of a release.

**Reasons for flakiness in scripts and possible solutions**

Let’s understand the prime reasons for occurring false positive/ negative

**Incorrect test data**

Test data is mandatory input for any test case to execute. We should always generate the test data as per the guidelines set for the fields in the application. Violation of the rules will lead to abrupt script failure this causes the test case to be failing.

The above failure of the false under the category of false negative, reason being we were not supposed to see a failure had there been in data issue

**Duplicate test data**

In the recent era applications have gained a lot of maturity in order to serve the user better. Certain important fields will not accept the duplicate data, so if we fail to generate the unique data we again hit a false failure this again comes under the category of false negative.

In order to overcome that we should always generate unique data. One possible solution would be to generate random values and append to the actual test data that is being read from the external excel excel file.

**Improper coding standards**

While designing the scripts proper coding standards must be followed failing which will led to the failure of the scripts in the form of false positive or false negative. Let’s discuss a few key points in coding standards as what we really mean by and what its contribution in leading to a failure is

**Wait handling events**

When we are especially automating the GUI we expect some delay for a page to load and the elements to generate and attach to DOM. We also have a parameter of network, considering all these things we should lay out a plan to handle the wait properly. We should also keep in mind that due to the inclusion of waits the order of n of a script should not go too high. We should be very careful enough to select the appropriate wait, to see below there are 3 waits that we can use as per need

Implicit wait

Explicit wait

Fluent wait

Java wait- thread.sleep(1000)

**Initiating and closing browser connections**

As we know that the test runs in browser hence invoking and quitting the browser are most common actions that we interact with. The opening and closing actions should be managed correctly by the testNG annotation so that we have new instance created after each test of after each execution as desired. Recommendation would be to launch a new instance after every test so that the impact of one test will not be there on the subsequent test.

Also we should ensure that if at all any browser is running in port even after completion of test we must close those connections before we re-launch the new browser instance for subsequent test. It is recommended to use driver.quit() to help with closing all connections as driver.close() only closes the current browser. This way we can avoid any flakiness that browser can contribute.

**Not handling the exceptions**

Exception handling brings lot of sense to the code, it helps application from not to terminate abruptly on failure cases. This is being achieved by surrounding the code snippet with try and catch blocks with necessary exceptions hierarchy.

What basically happens is that, if at all we don’t have a mechanism of catching the anticipated exceptions in the corresponding code block, when we face that exception unfortunately test case interrupts abruptly and no report would capture that failure as we don’t have that execution caught also no finally block to execute certain mandatory information. In this case we will get report with the actual result status of the previous test steps (be it pass/ fail) and report generation stops and it flushes. Ultimately the report doesn’t have the proper results printed in it, its creating false positive or negative in this case.

The prominent solution would be to use the try catch blocks properly with necessary test step information printed, so that everyone can understand that the test actually failed.

**Not printing the test step description and status in report**

Report is one of the best evidences to check whether a particular test failed or passed. As it’s a bible it’s essential to capture the right information. We should be very diligent while writing the report and capturing the test step information. A small wording mistake can change the meaning of the result and it creates ambiguity.

**Conclusion**

Since automated tests are executed by a system and given the fact that it has no intelligence thus far rather than performing the assigned task, it’s our responsibility to get the things right on very first time, so that the verification and analysis would not take much longer to understand while publishing the results. Small mistake can even be costlier based on the priority so nothing can be misjudged or misunderstood

Hope the information gave some insights on the topic, thanks for reading through.

Happy learning!!! ☺