**● How do you deal with flaky tests?**

**1. Start small**

Stick with the basics of writing automated tests. Each test should be small and have a single purpose to test one particular functionality. Try not to write tests that are dependent on one another. This also helps you know exactly why a test failed without having to look into the application code.

The goal is to have the ability to pick any group of tests and run them in any order. If this is not possible, then consider splitting the tests.

**2. Run tests regularly**

Once we write tests, run them regularly — preferably daily or at least weekly. The tests should get triggered automatically when new code is checked into the branch.

After the same test runs multiple times, we will have a lot of information about it: the average time it takes for the test to run from start to finish, the number of times the test passed, the actual functionality it is testing, how and when the test gets triggered, and other useful information.

**3. Identify unstable tests**

Based on the number of times you run a test, you will start figuring out whether a particular test is flaky or not. The test can fail for multiple reasons, such as page load times, assertions, bad data, problems in the environment, synchronization problems, and much more.

Analyse why the same test passes and fails intermittently. What is happening underneath the hood that makes the test act in a certain way? What types of failures are repeatedly happening?

we can also check the error logs to understand the failure.

**4. Separate out flaky tests**

Once the failed test analysis is complete, separate the flaky tests from the stable test suite. Just because a test fails intermittently, it does not mean that all other stable tests shouldn’t run, or that you will get a failed test report every time you run a set of tests.

It is important to keep the tests “green” as much as possible so that people take the results seriously and trust that automation is adding value. So, separate the unstable tests from the rest as soon as we start noticing the intermittent failures.

**5. Fix flaky tests one at a time**

One common mistake testers make is to try to be efficient by taking out many flaky tests and trying to fix and run them all at once. This will actually consume more time, because it will be hard to get to the root cause of the failures and create a permanent fix.

Work on flaky tests one at a time. While doing so, check whether a test has any dependencies on other tests. Debug the root cause of the problem by commenting on some code, adding print and wait statements as needed, adding breakpoints, and constantly monitoring the logs.

**6. Add tests back to the stable test suite**

Once we fix a flaky test, run it multiple times to ensure it is passing. After consistent successful runs, add the fixed test back to the stable test suite. Rerun the stable test suite multiple times to ensure there are not any unexpected outcomes.

● **Let's suppose there is a test pipeline taking about 1 hour to finish, what would you do to decrease the time of it.?**

Having too many Selenium commands in your test script flies in the face of atomic testing and is one of the most common underlying causes of long and unstable tests. Every single command takes time to execute and represents a new opportunity for something to go wrong. Minimize the number of commands required to execute a test case and run time will shrink accordingly.

Another effective way to reduce test run time is to use explicit waits rather than implicit waits. Implicit waits set a default wait time between each step or command across a test script, such that the subsequent step only executes after the pre-defined amount of time has elapsed. Explicit waits, on the other hand, enable the next step in a script to execute as soon as the preceding step is complete. Though more complicated to implement, using only explicit waits can have a significant positive impact on test run time.

**● Imagine you have the possibility to ask software engineers to develop tools for you that will increase your productivity as full-stack QA, please describe to them your requirements.**

Software Requirements Specification Document (SRS) gives business owners a clear vision of how much does it cost to develop their custom software product (web, mobile or desktop one).

1. It provides a realistic basis for estimating product costs, risks and schedules.
2. It provides an informed basis for deploying a product to new users or new operational environments.
3. It provides a basis for product enhancement.
4. It forces a rigorous assessment of requirements before design can begin and minimizes later redesign.
5. It establishes the basis for agreement between the acquirers or suppliers on what the product is to do (in market driven projects, the user input may be provided by marketing).
6. Organizations can use the specifications to develop validation and verification plans.