**How to build an agile friendly test automation framework**

Agile is test first model, this defines how to kick start testing in parallel with the development. The main agenda of this blog is to relate the automation testing to this model. The discussion is mainly on how to develop the test pack model so that it best suites the agile velocity and with the sprint level expectations.

There should be a concrete plan while bringing automation into agile, conduct proper analysis then understands the feasible fashion of test script development.

While designing the test automation framework in an agile methodology there are certain facts to be considered then adopt the best practices so that it best matches

**Development approach**

According to me understanding the development strategy is vital. Whether the approach followed is a test driven development (TDD) or Test late approach (TLD). This gives us insight on how the framework strategy should be and that way we end up designing best framework for agile approach.

Let’s understand quickly as what are the development strategies and how testing is conducted in each model

**TDD**

Test driven development is a test first approach, it is done in a way that, the tests are designed and executed. Initially we see the most of the test cases will fail as we didn’t have the code written. The strategy would be to implement the code in order to ensure the tests are passed. This is one of the widely used approaches given its advantages as we can avoid writing duplicate code, we can build an efficient system by covering all the possible conditional flows

One major advantage of this approach is more regression testing at module and integration level is conducted as we have tests written on module wise. This even helps when we integrate the test pack with CI/CD tools, that way we can run tests only related to a particular service or module once that gets deployed in the environment. This is gives a great opportunity for early testing.

**TLD**

Test last approach is conventional approach that’s being followed in waterfall model kind of software development life cycle. In this approach the testing is conducted once the whole system is available. In this approach we have the feedback latency because any bug that’s identified has to be triaged then find the root cause and give a fix. It is mandatory from a testing perspective that for every change there has to be regression testing to ensure nothing broke in the system. This is covered at system level.

**Automation testing approach**

Given the expectation that the testing has to go hand in hand with automation we must have a proper plan of developing scripts. Whether n-sprint approach or n-1 sprint approach to be followed? Let’s quickly look at how those sprints work

**n-sprint**

In this model, we will have to consider all user stories that are listed must be automated

**n-1 sprint**

In this model we likely trail by a sprint, whatever the stories those were developed in previous sprint are automated in current sprint.

Despite the model, we should always make sure that the regression pack is kept up to the date in the same fashion we have to maintain the sanity pack too.

**Types of Automation testing**

When in agile, we build the system in many iterations and we call them as sprints. It is obvious that we will have only smaller chunks built in linear fashion. The question here, is how do we accommodate the space for automation when UI and the whole system is not available? Because when automation is said, 90% of the spectators talk about automating at UI layer as that has ROI which is better.

If we think automation is best deployed to test UI, not others we are missing greatly the ideology. What we are meaning to say here is that we should think of automating at different layers such as API , DB validations to ensure that the system is doing the right thing. If we can ensure that at the initial stage our job would be lot easier, also it helps the business greatly as we are uncovering the bugs early in lifecycle and which are not that expensive to fix.

For an illustration let’s consider that the development approach is a TDD one, we will not have the complete system to test, rather we will have smaller modules/ micro services would be developed in an incremental fashion. To support the quick testing at that level we should consider designing a framework that can have tests based on module wise. We also need to understand what are the services offered by the component and accordingly write type of tests.

Write powerful integrated API tests in order to verify the micro service – micro service communication

Perform some data base table’s validation to see the business logic has been processed successfully and information stored at right table

We will apparently not have UI developed initially so as soon as that gets ready we should be ready in testing the flows that starts from UI

**Framework selection**

As per industry standards when we are getting into automation its mandatory to conduct feasibility study to understand the need then better select the tool, framework and any other required infrastructure. In order to better work with agile selection of BDD framework would be more helpful. This is also called advanced test driven development model (ATDD). Let’s quickly understand what those process all about and how to design framework of that sort.

**ATDD**

This is also known as behavior driven development approach, this is also a test first approach. The specialty of this model is that tests are driven by the spec/ feature file. Spec files have requirements documented in a plain English language called gherkin and in a clear step by step fashion. The main ideology behind this strategy is to make the tests simpler to decipher. Even a non-technical person when looked at the test, can easily identify what the test is doing.

If we understand that the approach is BDD, then we should more focus on designing the tests at a system level also the tests should appear in a way that they are understandable to everyone. We should get the requirement split into smaller minces then write every single action as a step then design the functionality to achieve that point.

When the spec file actually driving the test script, the test case carries more meaning and is very easy to understand by any non-technical stakeholders. Also it is easy to extend these tests as we move forward from one level of testing to another testing.

This is also giving us scope to perform early testing even testing at API layer can be done through this approach. It also provides a good reporting structure.

**Framework best practices**

We learnt that BDD framework is the best solutions for agile approach. There are certain framework standards which will make the framework more versatile. The framework should be built in a manner that it can be extended from modular level during initial stages to system level at lateral stages. Let’s look at certain components which will help us to make the framework n extendable one

**Write page objects**

Page object model implementation is truly the best solution and will reduce the maintenance to a great extent. We must store all the required page locators in a POJO class or in separate properties file basis the choice then write methods for each and every action.

The reason behind the approach was that if at all there is any change of elements or the functionality it is very easy to inculcate those changes to the test case. This way the scope of the test can be leveraged

**Separate test code from wrapper methods**

In any test automation framework for the benefit of reusability we build common methods then we connect them using the OOPs techniques. The test script shouldn’t contain any logic rather it must contain only the method calls as much as possible. So the reason that we are separating test code from the base code was that, we can best alter the test flow easily when there are changes to the framework. Also we can ensure the change of logic according to the new flow at one place would help reflecting the change into all test cases.

This indeed helps when we want to leverage the functionality of the test cases too. Suppose if we have tests written initially at module level and in future as we move to the system level if we want to extend the test this is very much possible.

**Relying on external resources for data**

Test data is essential for any test case to be run. Injecting the test data into the scripts or into the classes within methods is not a good practice. We should always consider reading them from the external resources such as excel workbooks, JDBC resources, properties file..etc. the advantage of having them outside is they don’t need to build every time we run a test also it is very easy to maintain and handle the data.

**Maintain separate packs/ group related tests**

As we discussed above that in TDD model, we will have smaller minces of the application developed then eventually it is matured. So if at all the design is a service or component based, the tests likely appear to be at unit/ module level. Writing all related tests for service together or mapping them with a common tag to ensure we only run those tests when that service alone is redeployed after some enhancements or bug fixes.

During initial stage we expect more deployments to happen, so if we can test more frequently that individual service alone rather running the entire suite that saves lot of time and gives quick feedback. This is also considered as rework if proper understanding is missing between the development and test teams. In a summary it helps us to conduct more testing at initial stage so that the component would never have bigger problems as we move to the further test cycles.

The another approach is that maintaining different packs for sanity and smoke validations and one complete one or regression testing. This will help us support for in sprint testing as well as to trigger a whole application test to ensure the new change didn’t break the existing thing.

**Conclusion**

Automation industry has taken a big leap from documenting the manually tested functionality as scripts for future execution to the purpose of serving the need for in sprint automation due to the agile parallel test execution expectation. With following the discussed practices and understanding the strategy of development and testing need we can design a matching framework that serves the need of the hour. There is no standard or hard coded practice as such, rather it fully depends on the context of a specific project.

These are some insights about customizing the automation framework so that it meets agile practices, things are quite debatable and documented basis the lore by the experience. Hope this has some learning inside. Thank you very much for reading this through ☺