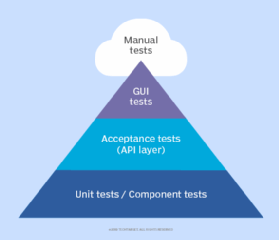
**How agile teams use test automation pyramid**

The main discourse in this blog is to understand the automation testing pyramid and to match that with agile work model. Automation testing pyramid defines the types of tests to be conducted at various levels, their scope and how intense the tests should be. Let’s take a moment to understand about the testing pyramid and how automation testing is performed under the governance of testing pyramid.

**Test Automation pyramid and its principles**

On a brighter note, Automation testing pyramid is nothing newer than what we have studied in v-model that is mapping a corresponding testing in accordance with the development readiness. But the only difference that we see is the v-model strategy doesn’t tell how intensified a test type should be, but that is very well explained in testing pyramid theory also it emphasizes the value o automation in each area. Testing pyramid says that it’s best way to conduct through testing in the early life cycle such as in unit and integration tests in order to prevent the bugs then a few end-to-end tests that cover most customer likely scenarios.



Let’s deep dive in about each level of testing in a hierarchical fashion and see what testing type should kick start and how much focused that test should be and what’s the value that automation is adding.

**Unit testing**

We know that in agile model the development and testing should go in parallel. In agile the development is managed in small component level initially then they integrate and after they ensure the system is developed to ensure the feature is developed.

The best way to ensure the most of the unit tests are automated since that’s the test which can give us the feedback very early also they are easier to fix, because the bug that is seen is likely happening due to the code error present in only that specific module. These tests can be run more frequently i.e. as and when there are deployments of that module we can start execute the test cases. The degree of dependency exists for this test is relatively small.

**Integration testing**

In the midst development team integrates these smaller components to perform business logic. To test the communication between the services we must consider writing some powerful integration tests. Basically these tests are conducted at API layer. This integration testing phase is vital because in the previous stage we have ensured that a component or service works better as a standalone but it could break when it is being integrated, so in order to verify that the communication is proper through API calls, business logic is performed and finally the information has stored in database. We need to write these tests. Automation is useful to perform more frequent execution also it helps validating very accurately. In this phase just to mitigate the risks due to delay in developing any other components we might rely on placing mock services for the ones which are not ready. Since it’s early in life cycle the bugs that are identified here are not too much costly to fix also it doesn’t take too long to triage then fix them. Also these tests never takes time to execute and feedback us he results, because we have no UI so no platform to add delay.

**System testing (end-to-end)**

Now we have done testing at two levels, ensured the designed services were working intended when they were as standalone also they were performing needed business logics after successful integration. If we think this testing is needed we are missing something great here. For a moment when we go back to fundamentals of testing the high level definition says, the goal is to verify that the functionality is working as intended and it should serve the purpose of the requirement.

We need to consider the user journeys that are key and what we think that are executed in the field as often as possible. Automation can add great value here as well, because the execution in the end to end phase most likely happens on UI layer and it can help us find any defect before the application goes live. Being in line with the test automation pyramid there shouldn’t be too many end to end tests because these tests however can help finding defects, they add significant delay to feedback results due to the fact that these tests have to interact with so many elements and pages.

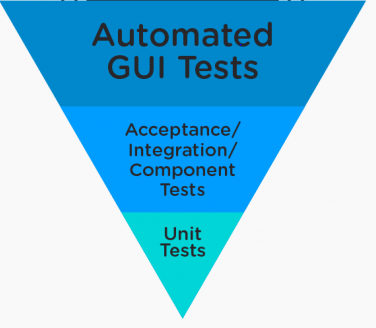
The fact is that though we find a defect it’s very costly to fix. When we say it’s expensive, it is mainly due to two things those are effort needed to identify where that particular bug occurs then understand the bug then fix it, the second one is it is found very late in the cycle. After given a fix for the identified bug a thorough regression testing is needed as we never know that the new code change will impact any other system which would have been working before.

**Testing pyramid followed in manual testing (less automation)**

In case of manual testing the approach is followed as per the below figure, where developers write unit tests to ensure the module works, but these tests are very minimal in number. As we go past different level of testing levels we intensify the testing we start writing more tests. Finally we end up conducting through end-end testing by testing all the customer journeys or use cases manually.

The main drawback of this approach is that, we are finding the defects but not preventing. Yes, you read it correct, the meaning for that statement can be explained as since we are not conducting through testing there could be fair chance for a particular micro service or component that is being developed may break on certain edge cases and this was hidden and when it was found later the developer had a lot of code base to look into to understand what causes the bug it apparently takes more effort than it ideally should have taken during initial stage. Moreover we require lot of regression testing to ensure the given fix hasn’t impacted any other working module

Holistically by following the anti-cone approach we are welcoming lot of redundancy by choosing the nook to corner testing at end- end phase. The amount of effort needed to write automated cases for end-to-end is vague, also the amount of time that they take to execute and provide results is relatively higher. Its lot of overhead to do that at this phase as it has high chances to skew the deadlines towards right.



**Conclusion**

If we were to brief the and re-iterate the chronicles of the test automation pyramid

1. Start with an intensified automated testing early in the life cycle probably at unit testing and integration testing levels
2. Don’t just limit the automation to the UI layer, we must consider automating at API layer as much as we can, given the TDD approach of development in agile SDLC model
3. Don’t overload the test pack with too many end to end test cases unless they think they are likely executed by customer and it has some important business logic to perform, believing that the other requirements would have been covered under user story based testing in earlier phases.
4. The reveres cone approach is only helpful to find the defects but not to prevent them for occurring from earlier life cycles.
5. Devops integration is best matched with the test automation pyramid it helps both the development and testing teams to trigger their tests.

Thanks for walking through the content, hope there would have been some learning ☺