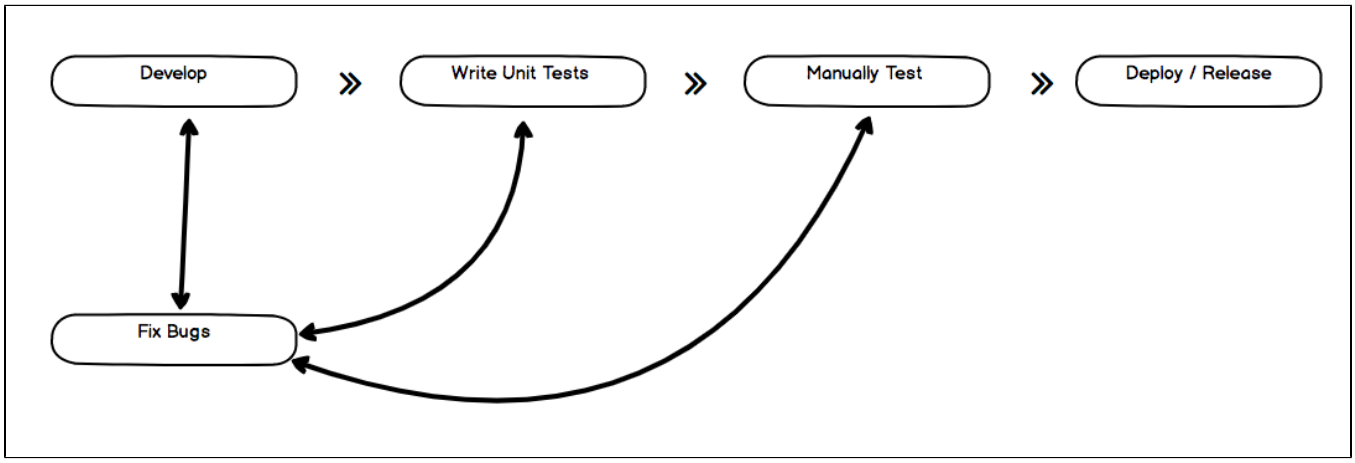
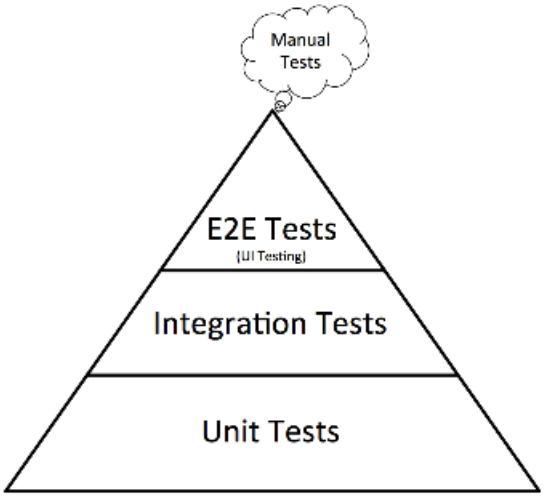
Automated Testing OCL Roadmap (Acceptance and Smoke Testing)

Manual software testing is performed by a human sitting in front of a computer carefully going through application screens, trying various usage and input combinations, comparing the results to the expected behavior and recording their observations. Manual tests are repeated often during development cycles for source code changes and other situations like multiple operating environments and hardware configurations. This is our current approach to doing verification in our OpenMRS OCL product.

An automated testing tool can playback pre-recorded and predefined actions, compare the results to the expected behavior and report the success or failure of these manual tests to a test engineer. Once automated tests are created they can easily be repeated and they can be extended to perform tasks impossible with manual testing. Because of this, automated software testing is an essential component of successful development projects. Automated testing does not aim to replace manual testing but it does aim at making the process scalable and efficient even as the product grows and it is released to users. This is what the current cycle looks like.

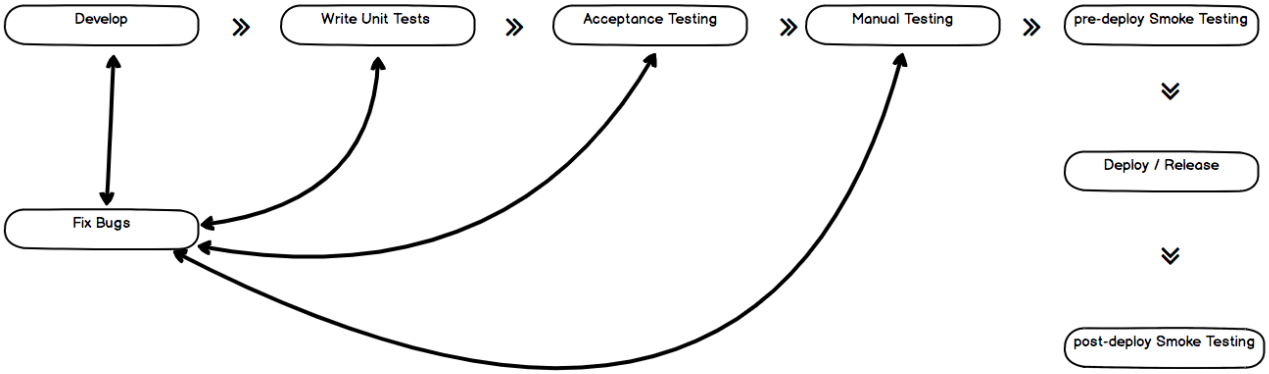


Currently, our bulk of tests sit at the manual level, where apart from testing single units, regressive testing is done using the manual testing approach. While testing manually is recommended, it is not able to capture all scenarios and that should be left for only the smoke test scenarios commonly known as 'happy paths' A well-tested product needs to have three levels of testing, Unit-tests, Service tests, and User interface tests. This is according to the test pyramid below. Unit Tests should cover individual units and actions and all the time use mocked data to perform the basic testing. Integration which can also be referred to as acceptance sit in between unit tests and End to End tests. We can add smoke testing just before End to End Testing to just verify our happy paths are there then do simple End to End Scenarios covering the requirement scope of OCL as it is.



In other words, we need to test OCL at all levels, To do that we need to get to a cycle that is somewhat similar to the diagram below. As shown in the diagram, we need to extend our test coverage to include acceptance testing and smoke testing which will reduce our current load when it comes to manual testing. Acceptance testing will cover most of the scenarios using a backendless approach. In this, we will be running our application and simulating responses from our backend using puppeteer and a mocked server. We will then provide mocked API responses and we can then check the different behaviors when the application is supplied with different combinations of data. This simply means that our test will run with our application launched by puppeteer but instead of using actual requests to the API we will mock the requests and provide a response while checking the changes on the application user interface. With this, we have the leverage to perform tests such as what happens when the network call to an API endpoint does not work, how do we handle error responses, what happens when we provide invalid data to the user interface. What happens when a user does an unexpected action say adding a dictionary and doing an accidental page reload.

FYI: **Integration tests are higher in the testing pyramid than unit-tests and have a goal of ensuring that processes and not units are tested. Writing Integration tests might involve mocking responses using fixtures, trying out user-centric scenarios such as reloads, intentional error introduction and even testing out edge-case scenarios during usage.** **Quote Source**



At the acceptance level, there is flexibility to make assertions that various elements are present in the application and we can reduce the number of assertions we have to write by using visual regression where we can take image snapshots of various screens from our diverse test just to ensure consistency when the application changes over time. With acceptance testing, we can provide all kinds of data types that we anticipate and others that we do not anticipate and that way we can be able to get bugs before someone else introduces them or reports them.

Smoke testing, on the other hand, will help us verify that all the scenarios that are in our OLC product description are correctly working and that we can be able to verify that before we deploy our releases and after we deploy our releases to the OpenMRS users. Post-deploy smoke testing is lightweight and can even be replaced by manual testing. Given an automated approach, bugs can be easily caught before they get to production and this will guarantee the robustness and confidence of OCL even before we release it to the users. Cypress framework is lightweight and is the most suitable tool too use for the smoke-tests.

To ensure that the application works as expected we would need to create and run the acceptance tests when code is pushed to GitHub and our smoke tests after deploy to master and over specific time periods say a scheduled night run when most resources are not in use.

Consistently doing this we can be able to achieve writing a framework and sample tests within a period of one month before we start writing actual tests with it and setting up the CI/CD.