Your engineering team will love automated testing eventually

Why do we test our code?

Many small teams and young companies cut corners on testing. When senior management sees a working product, it's often difficult to explain the importance of testing coverage when you could be running off to build the next cool feature.

The reality is that as you build more features, your product becomes more complex. As your product becomes more complex, it becomes harder to test. As your product becomes harder to test, crashes become more frequent. You can see the problem.

If a feature is important for our product, it must be tested automatically. Apart from ensuring that no product value is lost over time, we invested in automated testing for the following reasons:

- **Testing can become boring**. When you've just built a new feature, tapping every button or field can be a pleasure — but it's unlikely to be a pleasure after the 100th or 1000th time. We use automation to reduce repetition and keep our team fresh and motivated

- **We build tests to save our time.** It's easy to launch a new instance of the app to test the login screen. But how would you test a popup that is presented to a user only under complex conditions that must be recreated manually?
- **We build tests so that we can focus on things that matter.** 80% of our testing time is devoted to automation. The remaining time is devoted to testing the hardest cases manually, so that we don't repeat.
- **We build tests to make our code clean**. Building automated tests requires a modularised, loosely coupled project structure. You need to write code that makes any dependency easy to replace and any data easy to mock. This makes your code easy to test and maintain by multiple engineers working in parallel.

How to test

There are many ways to test a big project. We use a variety of approaches, depending on the situation:

Unit testing

The most basic method of testing your code. Unit tests are most helpful in parts of the code that affect business logic. They verify that algorithms work correctly, identify edge cases and ensure graceful failure.

We use a basic toolset to build unit tests: JUnit (default testing framework), Mockito (passing fake objects into tested classes), Jacoco (test coverage report). Our code is written in MVP (Model-View-Presenter) architecture, and uses Dependency Injection pattern (Dagger 2).

Integration tests

While unit tests focus mostly on plain Java code, we often need to test a whole component like an application screen or background service.

In this situation we don't build detailed, Robolectric-powered unit tests that check every single behaviour (*is this button disabled under given condition? Is the text set correctly?*). Instead, QA and software engineers build a controlled environment for the tested component and then validate whether it behaves in the way we expect.

Our transaction status screen, for example, has many different UI states. Instead of preparing dozens of tests, we prepare API or Database mocks for every state and provide them via **DaggerMock** to screen dependencies. Then Android Instrumentation tests, together with Espresso, do assertions on every single screen state.

End-to-end testing

End-to-end testing is the final stage of our QA process. When we finish working on a new feature or app release, we perform end-to-end testing to simulate the experience of a real user.

We use similar tools for integration and end-to-end testing (DaggerMock, Android Instrumentation Tests, Espresso), but this time there is no hermetic environment for every app component. Instead, we assume that each place in the app has to be reached in the same way that a user would do it — by clicking on the interface.

The price for end-to-end testing is time — the most complex scenarios can take up to a couple of minutes to test. But in return we cover real use cases (features, navigation flows, third party libraries and more), connect to the real API and work on real data.

This content is from
https://medium.com/azimolabs/automated-testing-will-set-your-engineering-team-free-a89467c40731