**Software Testing, Quality Assurance and Maintenance** Winter 2019

Lecture 18 — February 27, 2019

*Patrick Lam* *version 1*

## Engineering Test Suites

We are going to move onto the second part of the course. In the first part of the course, you learned ways to make sure that your test suites are exhaustive enough. In this part of the course, you will learn about making your test suites better as engineered artifacts.

**Why tests?** Let's start by talking about what test suites can do for you (as a developer).

Reference: Kat Busch. "A beginner's guide to automated testing."
`https://hackernoon.com/treat-yourself-e55a7c522f71`

**Anecdote: "TODO: write tests."** We're all busy, right? Surely tests are less important than writing actual code to solve problems. And it can be hard to set up the test.

When you're writing code, you clearly want to know that it can at least work. So you may have a development setup. Kat Busch describes writing a Java server to interface with an Android app. Her development setup involved manual testing:

- set up test server on dev machine;
- install app on test phone;
- manually create a test case on test phone.

She writes: "Obviously I didn't test very many code paths because it was just so tedious."

Since Dropbox (her employer at the time) used code review, she actually had to write tests to pass code review, even if it was annoying to do so.

"Lo and behold, I soon needed to fix a small bug." But, of course, it's easy to introduce even more bugs when fixing something. Fortunately, she had some tests.

> I ran the tests. Within a few seconds, I knew that everything still worked! Not just a single code path (as in a manual test), but all code paths for which I'd written tests! It was magical. It was so much faster than my manual testing. And I knew I didn't forget to test any edge cases, since they were all still covered in the automated tests.

Not writing tests is incurring technical debt. You'll pay for it later, when you have to maintain the code. Having tests allows you to move faster later, without worrying about breaking your code.

Writing tests is like eating your vegetables. It'll enable your code to go big and strong.