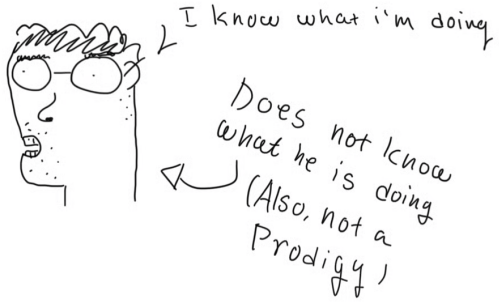
Lol, why does testing matter?

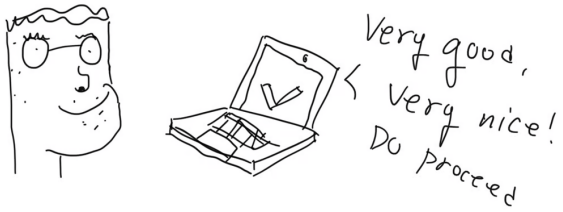
Email from non existent person: “why does unit/integration/whatever testing matter? I’m 21 years old, and everything I write works because I’m a goddamn prodigy.“



This could very well have been a mail written by me, when I had just started my new company Codeboom. I wasn’t in this alone, and had a good friend of mine by my side. We had a lot of disputes, many of which made me a better developer, but we agreed on one thing: “Lol, testing does not matter, I can just test it out manually to see if everything works as intended”. Also for legal reasons, I wasn’t a goddamn prodigy. But the amount of self-confidence you can have after having been the “kid that knows computers” while growing up, is staggering (actually it’s enough to start a software consultancy company, despite having close to no experience writing production ready code).

But alas, you don’t know what you don’t know, and I now know some of the stuff, I didn’t know back then.

Testing is of course about verifying everything works as intended. But to me, It’s as much about being able to edit code with a calm mind. Without tests, I get slightly anxious, that whenever I make a change, it might break something. While you’re probably feeling very confident, that you know what all of the side-effects of the code you just wrote are, you are most likely wrong. If you created your application using some tool that initialises it with all sorts of dependencies and utilities for you, your codebase is already immensely more complex than what a single person can comprehend. Somebody recently asked on hacker news why create-react-app needs +1400 dependencies, just to initialise a project. I don’t know the answer, and I know that there exists tools that can do it with less. But it paints a picture. If you change something in your application that relies on one of these dependencies, and they in turn rely on some other dependency etc. etc. You suddenly made a change to a complex system. And I swear to god, you don’t have the full overview of it. So having that little which verifies that what you just wrote works as intended, might just be able to stop you waking up in the middle of the night bathed in sweat thinking, “DID I JUST BREAK X Y Z?”



Also another reason is that it is a real time saver. The alternative to automated testing is manual testing, if you want to verify everything works as intended. And I’ve done this, as it was “no big deal”. That quickly changed when my company eventually died (do recall, that I had no experience writing production ready code), and I got a more normal nine to five. Here I was forced to work with developers significantly better than myself. They had a very elaborate testing setup: unit tests, integration tests, end-to-end

had a very elaborate testing setup: unit tests, integration tests, end-to-end tests, Q&A testing and all that jazz. This is when I realised how much time I've wasted manually verifying my changes didn't break anything and bug hunting blindfolded.

Why do I say I was bug hunting blindfolded? Whenever you have a bug without a testing setup, there's nothing else to do, than listen to where your gut feels the problem is, and then go investigate. But with a good testing setup you will get one of these bad boys, signifying that an assert failed: □. This makes it much easier to see the full picture. Let's say that your function which maps millisecond timestamps to a hh:mm:ss format broke. Now you just have a string of gibberish in your interface. Where do you start looking? You probably have some idea of which functions are included in this process. You can then start from the top, and see if they all work as intended. But with a good testing setup, you might get an error message telling you, that the function pulling timestamps from the database is broken. You can then simply go fix that, to get everything back on track.

The above was a very simple example, and in that case you could probably debug it without the use of tests... But it grows more useful, the more complex a system is. So let me finish with a final note, which is that I don't always use tests. Whenever I create simple projects for myself or do some sort of prototyping I often can't be bothered. But if any of these projects eventually grow to a size where i lose the overview of what is going on, then I start adding them in.

And sometimes even before that, just for good measure.

Best,
Mads
Ps. All of the benefits of writing tests relies on being able to write good tests. This is a topic for another time, but as a starter, here are some guidelines I wrote for an open-source project: https://github.com/kirbydesign/designsystem/wiki/The-Good:-Test/f9db01165ba6f6a98cdba324a7000129bc220b9a

Newer Older

10th February 2024
Program@Stibo Gym

2024-02-06
How to profile a ts-node script f...

Mads Buchmann Frederiksen © 2022-2025