Thank You to the Code PaLOUsa Sponsors

What to Avoid When Writing Unit Tests

Bob Fornal

"These tests should never have been written. They provide no or little value." -ME

A QA Engineer walks into a bar.

- Orders a beer.
- Orders 0 beers.
- Orders 999,999,999,999 beers.
- Orders a lizard.
- Orders -1 beers.
- Orders a ueicbksjdhd.

The first real customer walks in and asks where the bathroom is. The bar bursts into flames, killing everyone.

```
class BeersOnTheWall {
  beers = 99;


  sing = () => {
    while (this.beers > 0) {
      let countBeer = this.getBeers(this.beers);
      console.log(`${ countBeer } of beer on the wall, ${ countBeer } of beer.`);
      this.beers = this.beers - 1;
      countBeer = this.getBeers(this.beers);
      console.log(`Take one down and pass it around, ${ countBeer } of beer on the wall.`);
    }

    console.log('No more bottles of beer on the wall, no more bottles of beer.');
    console.log('Go to the store and by some more, 99 bottles of beer on the wall.');
  };

  getBeers = (beers) => `${ beers } ${ (this.beers === 1) ? 'bottle' : 'bottles' }`;

}

const beer = new BeersOnTheWall();
beer.sing();
```

What Are Unit Tests?

- Unit Tests exercise small parts of the application (code-under-test).
- … in complete isolation.
- … actual behavior versus expected behavior.

- Unit Tests should be fast, simple, and stable.

- Unit Tests reflect the specifications.
- … can act as documentation.

- Unit Tests are a safety net.
- … provide immediate feedback about code changes.
- … find and fix bugs earlier.
- … contribute to higher code quality and better architecture.
- … faster detection of code smells.

Talk Details

Code Repository: https://github.com/bob-fornal/what-to-avoid-when-writing-unit-tests

I can be found at …

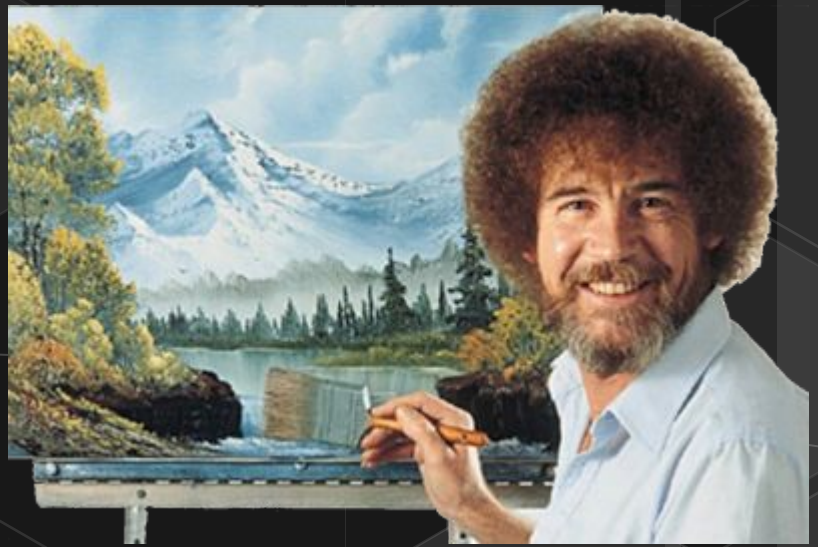
- **https://linqapp.com/conference**
- Twitter: @rfornal
- Articles https://dev.to/rfornal
- LinkedIn: https://www.linkedin.com/in/rfornal/

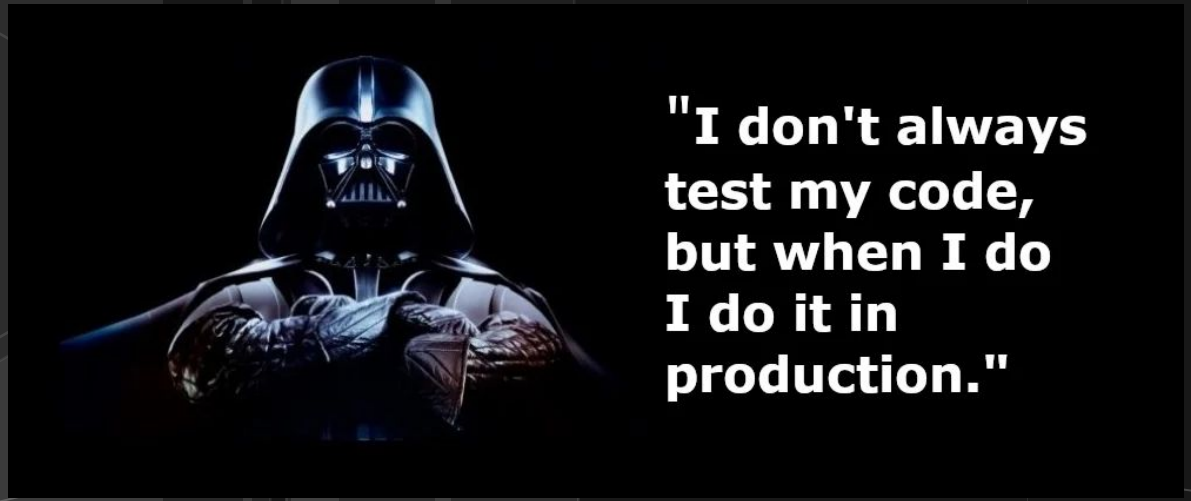
Social Media Project …

#100DaysOfCode or **#100Devs** … in one of these Twitter threads, pick three people in your field that haven't had a response and cheer them on.

Testing code has been described as an "art form."

It is, but it should not be.





Tests Should Not Be "Well-Factored"

Let's examine some examples …

- Keep the Reader in the test.
- Dare to violate the DRY Principle.

Keep the Reader in the Test

Given

A function that returns a score.

Problem

The unit test has has some part of it abstracted in a way that makes reading the test difficult.

Dare to Violate the DRY Principle

"Duplication of Code Logic"

Given

A function that adjusts a score.

Problem

The unit test has has some part of it abstracted in a way that makes reading the test difficult.

Poisoning the Codebase: Non-Deterministic Factors

Given

A function that returns a category, the time of day.

Problem

The function under test is tightly coupled to a concrete data source, violating the Single Responsibility Principle. The test cannot easily test the functionality.

Poisoning the Codebase: Side-Effects

Given

A function that updates the time of last motion and triggers something off or on.

Problem

The function under test has too many side effects that become hard to test.

Testing Too Much

Given

A function that does too many things.

Problem

The function under test has too possible paths to handle easily.

Bad Test Double

"Testing the Mock"

Given

A service is used to get some data.

Problem

Replacing the service (test double) does not correctly take into account changes in the service or returned data.

False Positives

Given

A function that processes some time delay.

Problem

The unit test does not get to the code inside the setTimeout and “does not fail” which equates to a pass.

Excessive Setup

Given

A unit test that take a lot of setup code to run.

Problem

Excessive setup is more of a CODE SMELL than something where code can be shown that is incorrect versus correct.

Just be aware that this is a case where care should be taken to examine why the setup is so lengthy.

Solution

- If there is a legitimate need, increase documentation.
- If the code under test does not adhere to the Single Responsibility Principle, maybe a refactor would be appropriate to make it more testable.

Testing Private Functionality Directly

Given

A class with private functionality that is difficult to test indirectly.

Problem

How do we test private functionality.

Solution

1. Indirectly
2. Make it public, directly.
3. Abstract it into a new service, directly.

Unit Recommendations

1. Unit Tests should be Readable.
2. One Assert per Test Method.
3. Avoid Test Interdependence.
4. Keep it Short, Sweet, and Visible.
5. Add them to the Build.


WE'RE HIRING!
APPLICATION DEVELOPERS
Fullstack/Frontend
Java, .NET, Angular, React.js, Vue.js...
Software/Web/Mobile/Cloud
Check us out at leadingedje.com
Made with PosterMyWall.com