Overreacted

Goodbye, Clean Code

January 11, 2020 • ☕️ 5 min read

Translated by readers into: Español • Português do Brasil • 简体中文 • 한국어

It was a late evening.

My colleague has just checked in the code that they've been writing all week. We were working on a graphics editor canvas, and they implemented the ability to resize shapes like rectangles and ovals by dragging small handles at their edges.

The code worked.

But it was repetitive. Each shape (such as a rectangle or an oval) had a different set of handles, and dragging each handle in different directions affected the shape's position and size in a different way. If the user held Shift, we'd also need to preserve proportions while resizing. There was a bunch of math.

The code looked something like this:

```
let Rectangle = {
  resizeTopLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeTopRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
};

let Oval = {
  resizeLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeTop(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottom(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
};

let Header = {
  resizeLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
}

let TextBlock = {
  resizeTopLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeTopRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomLeft(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
  resizeBottomRight(position, size, preserveAspect, dx, dy) {
    // 10 repetitive lines of math
  },
};
```

That repetitive math was really bothering me.

It wasn't *clean*.

Most of the repetition was between similar directions. For example, `Oval.resizeLeft()` had similarities with `Header.resizeLeft()`. This was because they both dealt with dragging the handle on the left side.

The other similarity was between the methods for the same shape. For example, `Oval.resizeLeft()` had similarities with the other `Oval` methods. This was because they all dealt with ovals. There was also some duplication between `Rectangle`, `Header`, and `TextBlock` because text blocks *were* rectangles.

I had an idea.

We could *remove all duplication* by grouping the code like this instead:

```
let Directions = {
  top(...) {
    // 5 unique lines of math
  },
  left(...) {
    // 5 unique lines of math
  },
  bottom(...) {
    // 5 unique lines of math
  },
  right(...) {
    // 5 unique lines of math
  },
};

let Shapes = {
  Oval(...) {
    // 5 unique lines of math
  },
  Rectangle(...) {
    // 5 unique lines of math
  },
}
```

and then composing their behaviors:

```
let {top, bottom, left, right} = Directions;

function createHandle(directions) {
  // 20 lines of code
}

let fourCorners = [
  createHandle([top, left]),
  createHandle([top, right]),
  createHandle([bottom, left]),
  createHandle([bottom, right]),
];
let fourSides = [
  createHandle([top]),
  createHandle([left]),
  createHandle([right]),
  createHandle([bottom]),
];
let twoSides = [
  createHandle([left]),
  createHandle([right]),
];

function createBox(shape, handles) {
  // 20 lines of code
}

let Rectangle = createBox(Shapes.Rectangle, fourCorners);
let Oval = createBox(Shapes.Oval, fourSides);
let Header = createBox(Shapes.Rectangle, twoSides);
let TextBox = createBox(Shapes.Rectangle, fourCorners);
```

The code is half the total size, and the duplication is gone completely! So *clean*. If we want to change the behavior for a particular direction or a shape, we could do it in a single place instead of updating methods all over the place.

It was already late at night (I got carried away). I checked in my refactoring to master and went to bed, proud of how I untangled my colleague’s messy code.

The Next Morning

… did not go as expected.

My boss invited me for a one-on-one chat where they politely asked me to revert my change. I was aghast. The old code was a mess, and mine was *clean*!

I begrudgingly complied, but it took me years to see they were right.

It’s a Phase

Obsessing with “clean code” and removing duplication is a phase many of us go through. When we don’t feel confident in our code, it is tempting to attach our sense of self-worth and professional pride to something that can be measured. A set of strict lint rules, a naming schema, a file structure, a lack of duplication.

You can’t automate removing duplication, but it *does* get easier with practice. You can usually tell whether there’s less or more of it after every change. As a result, removing duplication feels like improving some objective metric about the code. Worse, it messes with people’s sense of identity: *“I’m the kind of person who writes clean code”*. It’s as powerful as any sort of self-deception.

Once we learn how to create abstractions, it is tempting to get high on that ability, and pull abstractions out of thin air whenever we see repetitive code. After a few years of coding, we see repetition *everywhere* — and abstracting is our new superpower. If someone tells us that abstraction is a *virtue*, we’ll eat it. And we’ll start judging other people for not worshipping “cleanliness”.

I see now that my “refactoring” was a disaster in two ways:

- Firstly, I didn’t talk to the person who wrote it. I rewrote the code and checked it in without their input. Even if it *was* an improvement (which I don’t believe anymore), this is a terrible way to go about it. A healthy engineering team is constantly *building trust*. Rewriting your teammate's code without a discussion is a huge blow to your ability to effectively collaborate on a codebase together.

- Secondly, nothing is free. My code traded the ability to change requirements for reduced duplication, and it was not a good trade. For example, we later needed many special cases and behaviors for different handles on different shapes. My abstraction would have to become several times more convoluted to afford that, whereas with the original "messy" version such changes stayed easy as cake.

Am I saying that you should write "dirty" code? No. I suggest to think deeply about what you mean when you say "clean" or "dirty". Do you get a feeling of revolt? Righteousness? Beauty? Elegance? How sure are you that you can name the concrete engineering outcomes corresponding to those qualities? How exactly do they affect the way the code is written and modified?

I sure didn't think deeply about any of those things. I thought a lot about how the code *looked* — but not about how it *evolved* with a team of squishy humans.

Coding is a journey. Think how far you came from your first line of code to where you are now. I reckon it was a joy to see for the first time how extracting a function or refactoring a class can make convoluted code simple. If you find pride in your craft, it is tempting to pursue cleanliness in code. Do it for a while.

But don't stop there. Don't be a clean code zealot. Clean code is not a goal. It's an attempt to make some sense out of the immense complexity of systems we're dealing with. It's a defense mechanism when you're not yet sure how a change would affect the codebase but you need guidance in a sea of unknowns.

Let clean code guide you. **Then let it go.**
Discuss on Twitter • Edit on GitHub

Subscribe to the Newsletter

Subscribe to get my latest content by email.

Your first name

Your email address

Subscribe

I won't send you spam.
Unsubscribe at *any* time.

Overreacted

Personal blog by Dan Abramov.
I explain with words and code.

← My Decade in Review The WET Codebase →