Refactoring

smarty

By Your
Favorite Dev

```
git clone
git@github.com:easton873/refactoring.git
```

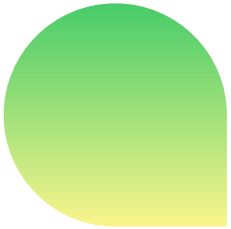
Why Refactor?

- Improves the Design of Software
- Makes Software Easier to Understand
- Helps You Find Bugs
- Helps You Program Faster

First step to Refactoring...

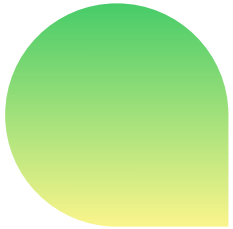
Tests

How to write tests



By hand

- Write down the inputs
- Work out the answer by hand
- Write a test expecting it



By trusting

- Create inputs
- Run test with those inputs
- See failed test and copy the output you got to be the expected output in the test

Bad Code Smells

Two things to keep in mind

Code and then refactor

- Don’t worry about refactoring the first time through
- Thinking about refactoring can hamper you from actually writing working code in the first place
- We refactor code because it makes it more maintainable, it is okay if it doesn’t start that way
- It is a way bigger problem to never refactor

These aren’t hard and fast rules

- There are exceptions to almost all of these so don’t get mad at yourself or others if you see this in code
- There is more than one right way to write code
- There are too many of these to always keep track of, these are good to review every once and a while

Bad Code Smells

Mysterious Name

People often think it isn't worth the trouble to rename something so they keep the mystery going

Can be a sign your design is too complicated if you can't think of a good name

Duplicated Code

Causes you to need to maintain the code in two spots

Long Function

Delegating to other functions to make your big function smaller is good if your smaller functions have good, descriptive names

Long Parameter List

Replacing parameters with queries (the data to derive it is in another parameter), or you can introduce a parameter object (Config objects are something I've used at Smarty)

Bad Code Smells

Global Data

Causes problems because nobody owns it, any part of the code base can modify it. Also painful to test it

Global data is better in small doses

Mutable Data

Mutable Data that can be calculated elsewhere should just be generated using a query when it is needed

Divergent Change

When one module changes for multiple reasons

For example, if there was one module I edited when we added new address fields and plans

Shotgun Surgery

When many modules need change in many places because we added a feature

Bad Code Smells

Feature Envy

When a module uses other modules more than its own internal code

Modules should communicate outside of themselves minimally

Data Clumps

Data that only exists with other data

A good test is to delete one of the data values and see if the other data still makes sense

Fixing this leads to finding **Feature Envy**

Primitive Obsession

Using primitive types for things that shouldn't be the same type

If millimeters and inches are both just a double, they can be added, but shouldn't be

Repeated Switches

Use polymorphism to eliminate a lot of conditional logic

Switches are okay to use too, just not duplicated ones

Bad Code Smells

Loops

Replace with built in functions like filter and map

Loops are also okay to use

Lazy Element

When you have too much structure for what needs to be done

Usually a type with just one method or a function that never grew to do more

Speculative Generality

The product of somebody saying “we’ll need this functionality someday”

Found as functions with lots of unused parameters, abstract classes that are never used, etc.

Temporary Field

When member variables in a struct are only used sometimes, or only when you go down certain paths of logic it remains unused

Bad Code Smells

Message Chains

When a client asks an object of one object and that object asks for another object and so forth

This means the client is coupled to the structure of the navigation

Middle Man

Some objects don't do anything but talk to other objects and sometimes it is better to talk to the object in question directly

Insider Trading

Failing to encapsulate. Make sure your modules don't talk to each other more than they need to

If two modules have common interests, try making a third module to help facilitate

Large Class

When a struct tries to do too much it often has too many fields and that means duplicated code cannot be far behind

Make a composite to start

Bad Code Smells

Alternative Classes w/ Different Interfaces

When two things could share an interface but they don't

Names might not match or arguments could be slightly tweaked to make it the same

Data Class

Structs with no methods

Usually a sign that some of your code needs to move from the client to the struct itself

Refused Bequest

Inheriting stuff you don't need/use

Comments

Are good, but sometimes they exist just to justify bad code

Rename things or rethink things if you think they need a comment. Move the comments into the code to make it self documenting

John’s 2 Refactoring Rules:

- Decouple
- Make it simpler

Remember:

“Clear is better than clever”

– Rob Pike: Go Proverbs

Exercise: Hotel Simulator

Yippee