Generalisation through complexity

The benefits of Test Driven Design Triangulation

By Stuart Hopwood

The problem

When attempting Test Driven Design methodologies, developers often attempt to write the design up front.

The problem

This can lead to the following issues:

- Tests that are tightly coupled to implementation, causing refactoring issues and bugs.
- Tests based on testing implementation and not based on behaviour.
- Loss of productivity while trying to implement too many steps ‘down the road’.

The solution

How can we prevent this?

With the Test Driven Design cycle with the triangulation pattern.

The solution

The Test Driven Design cycle and the triangulation pattern are two practices that go hand in hand, and if used correctly results in code that is cleaner, more reusable , and that is easier to expand upon.

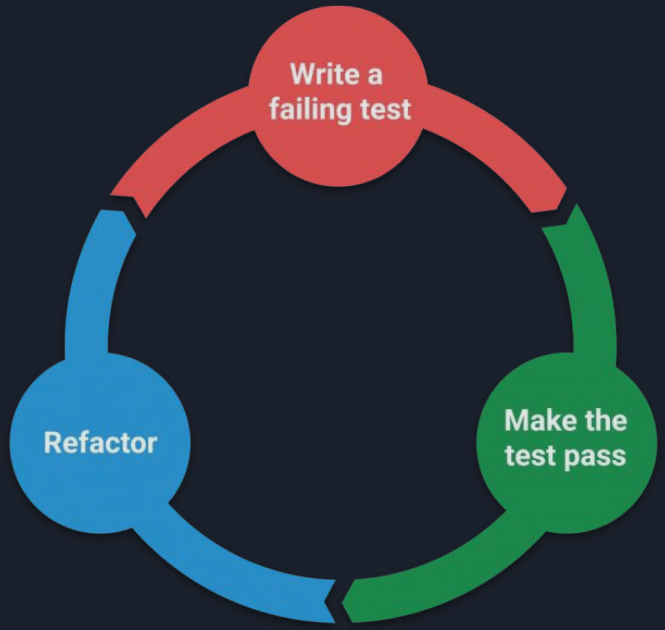
The solution

Let’s talk about the first part...The TDD Cycle.

TDD Cycle

What is the Test Driven Design (TDD) Cycle?

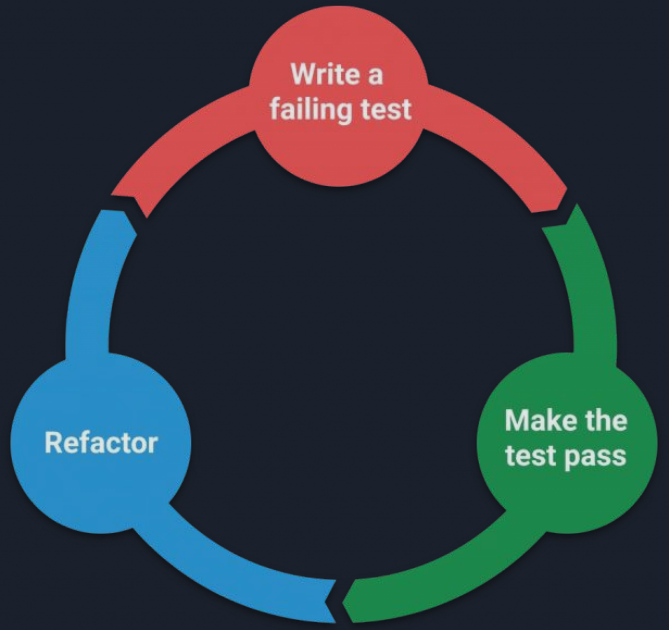
The TDD Cycle



TDD Cycle - Write a failing test.

Step 1: Write a failing test

- Tests should be written to test ‘public’ behaviour, i.e. “When user inputs 1 return “I”
- The test should fail when ran.
- Tests should assert one ‘logical’ behaviour not many.
  For example: ‘input a number and expected a string in return’



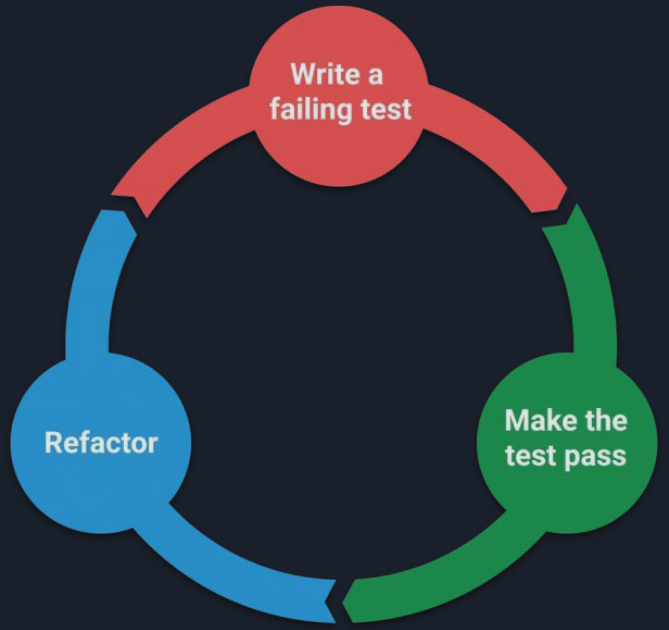
TDD Cycle - Write a failing test.

Step 2: Make the test pass

- Write code that makes the test pass in the simplest manner. For example:

```
public string Convert(){
    return "I";
}
```

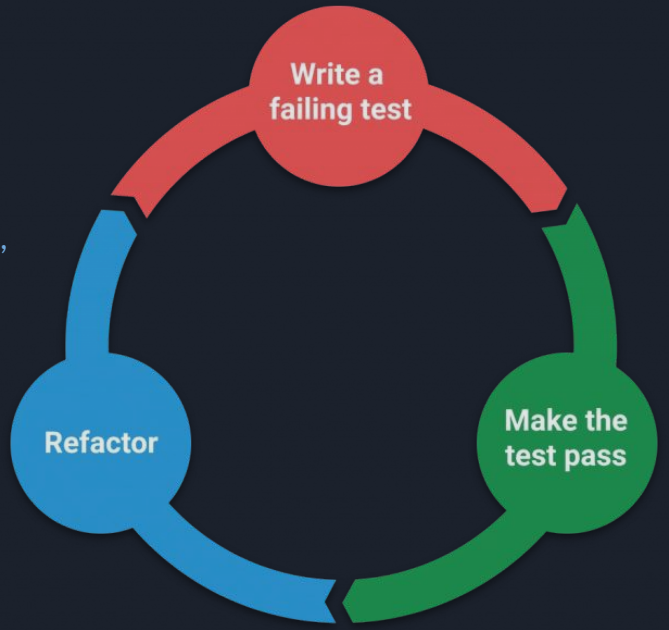
- The test should pass without modification.
- Write just enough code the pass the test



TDD Cycle - Write a failing test.

Step 3: Refactor

- When you have repeated the same piece of code three times, reactor.
- Introduce variables, constants, collections, methods, etc, that abstract repeated blocks of code into reusable code that is more generalised and less specific.
- Follow TTP (Transformation Priority Premise) where simpler refactors are preferred over more complex ones.
- Ensure the tests still pass.



The solution

Now let’s talk about the second part...The Triangulation Pattern.

The Triangulation Pattern

What is the Triangulation Pattern?

The Triangulation Pattern

According to Kent Beck, the Triangulation Pattern is the practice of generalising code, one test at a time, doing the simplest, least general thing at each step and making the code more general as we go.

The Triangulation Pattern

In our Roman Numeral example - We start with a test that expects the Roman Numeral for ‘1 = I’.

So we just return ‘I’

```
public string Convert(){
    return "I";
}
```

Test passed.

The Triangulation Pattern

Then we write a test for ‘2 = II’.

To make it pass we can just add a branch for the second case using an If statement.

```
public string Convert(int input){
    if(input == 2)
        return "II";

    return "I";
}
```

The Triangulation Pattern

Then we get to ‘3 = III’

This simplest way to make the test pass is to add another branch.

```
public string Convert(int input){
    if(input == 2)
        return "II";
    if(input == 3)
        return "III";
    return "I";
}
```

The Triangulation Pattern

But we have just repeated ourselves three times, so we refactor!

```
public string Convert(int input)
{
    var numerals = string.Empty;
    for (int i = 1; i <= input; i++)
    {
        numerals = $"{numerals}I";
    }
    return numerals;
}
```

Adding a loop allows our code to be more generalised and less specific, while still passing the tests.

The Triangulation Pattern

And so on....

As we add tests we ‘triangulate’ to the desired behaviour and the resulting code becomes more generalised.

The tests themselves become more specific as we try to cover the edge cases and boundaries, but they aren’t tied to any one implementation reducing refactoring and debugging issues later on.

The Triangulation Pattern

Although the example of the Roman Numeral Convertor was simplistic hopefully it has demonstrated how you can achieve better code through the methodologies of TPP + Triangulation.

Many Thanks.