Continuous Delivery ltd

TDD & BDD
Design Through Testing

Exercise 4 Part 1
How Tests Improve Design

Overview

Dependency Injection isn't the only useful tool that helps us to improve testability and design. Another important idea is ***Separation of Concerns***. Driving design decisions from the perspective of always aiming to separate concerns amplifies modularity and cohesion and makes our code more testable.

This technique is particularly valuable at the "edges" of our system where it interacts with I/O devices of some form: displays, storage, remote comms, etc.

This exercise is focused on creating a program to play the kid's game ***FizzBuzz***. At its heart this is a simple problem, but the requirement to do this for all of the numbers from 1 to 100, and to display the result, makes it a little more interesting.

I have divided this exercise into two parts:

Part 1 answers the question "given a number, what's the result?".

Part 2 looks at the concerns of: iterating over a sequence of numbers; and, displaying the result. Breaking the problem this way, improves the testability, helps us to think about separation of concerns and how this improves the design.

Continuous Delivery Ltd
Registered Company No: 9094073 England & Wales
VAT: GB 190 6295 90

Goals of the Exercise

To practice ***RED, GREEN, REFACTOR!***

Use TDD to develop some code that, given a number, will return a correct FizzBuzz response.

Exercise

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

Sample output:

```
1                   Fizz
2                   13
Fizz                14
4                   FizzBuzz
Buzz                16
Fizz                17
7                   Fizz
8                   19
Fizz                Buzz
Buzz                ... etc up to 100
11
```

Advice

For this exercise, remember to focus on the FizzBuzz part, leave the iteration and display for part 2.

The overall plan is to think in terms of a design with separate concerns, but also to focus in on the concern in front of us. Don't worry too much about the rest of the design, abstracting the problem well allows us the freedom to not worry about detail elsewhere.

We could do this in any order. For some problems, picking the order that you start with may help you to understand, in more detail, what you will need from the other parts. Often a top-down approach to the design can help with this. So we could start with the iteration.

Strongest advice in this part is to ***work in tiny steps*** - never be too far from stability! Be very careful not to write code before you have a test that demands it of you.

Remember The Rules of the Game!

Make progress in very small steps.

The goal is to practice TDD, more than it is to solve the problem. Don't rush ahead to solve the problem and forget to test.

Don't add new code, unless you have a failing test that makes you.

Always refactor on a passing test, and run the tests after each, small, change to verify your changes.