Code Craft Kick-Start

Exercises & Links

Day #2

Test Doubles

Exercise #4

CD Warehouse++

As always happens, our customer has added some requirements for a CD Warehouse:

```
When someone buys a CD, we need to notify the charts of the sale, telling
them the artist and title of the CD bought, and how many copies were sold.

We offer a price guarantee for albums that are in the Top 100. We guarantee
to beat the lowest competitor's price by £1.
```

How to tackle this exercise:

1. *Use test doubles to enable you to retrieve test data (such as the CD's current chart position and the lowest competitor price) and to test that messages were sent to external systems*
2. *Don't forget to work backwards from assertions (or the mock object equivalent of an assertion), and don't declare interfaces until the test doubles require them.*

Specification By Example

Exercise #5

Solve A Problem With Software

In your pair, specify a set of features for a simple software product you believe could solve a real-world problem.

How to tackle this exercise:

1. *Elect someone in your pair as the **customer***
2. *Working with your customer, define a real-world **problem** (NOT A SOLUTION!) in a single problem statement (e.g., it's very hard to find good drummers in my local area)*
3. *Envision a single **headline feature** that you imagine will solve the problem (e.g., drummers within an hour of me are listed in order of proximity)*
4. *List up to 5 **supporting features** that would be required to make the headline feature work (e.g., drummers can register their location, travel time is calculated, etc)*
5. *Working one feature at a time, work with the customer to precisely capture how they envision the feature working in key **scenarios** using **examples***
6. *Do not write any code. This can all be captured in e.g., a spreadsheet*

Outside-In TDD

Exercise #6

Solve A Problem With Software II

In a test-driven way, implement software that passes your customer's tests.

How to tackle this exercise:

1. *Work one feature, one scenario and one example at a time*
2. *Create a test fixture for each scenario. Write a unit test for each unique outcome of each example. Copy and paste the data from your example directly into the automated test.*
3. *Name your automated tests to easily identify which feature, which scenario and which outcome they are testing.*
4. *If a scenario has multiple examples, consider using a parameterized test.*
5. *Work outside-in, stubbing or mocking any dependencies you wish to make separate concerns (e.g., a database, or a web service, or another component of your system you wish to have its own tests later)*
6. *When a layer of your design is passing its tests, moving inwards to TDD the next layer (the parts you stubbed and mocked), and keep moving inwards until you have a working end-to-end core implementation that excludes any external dependencies.*
7. *Refactor when your code passes the tests but doesn't conform to the rules of Simple Design, Modular design or Conceptual Correlation. Keep extracted elements (e.g., helper methods or classes) hidden from the test code where possible.*
8. *Use version control to ensure you can easily get back to a working version if you mess up.*
9. *DON'T FORGET WHO YOUR CUSTOMER IS! Any questions about requirements should be directed to them.*