6. Testing & Refactoring

How we create classes?
- We think about what a class must do
- We focus on its implementation
- We write fields
- We write methods
- We may write a few test cases to see if it works
- We hand it off to users of our code
- We then wait for them to come back with feedback (problems)
Test First Coding

- How about starting with a test case even before we have any code for our class?

- How about first write test that fail because the code to support it does not exist?

- How about adding functionality to our system by adding tests incrementally and then adding code to make those tests succeed?

Test First Coding Benefits

- It would
  - completely revert the way we develop
  - We think about how our class will be used first
    - Helps us develop better interfaces that are easier to call and use
  - Would change the way we perceive things
  - Will have code that verifies operations
  - Will increase robustness of code
  - Will verify changes we make
  - Will give us more confidence in our code
Test First Coding Benefits...

• Forces us to make our code testable

• Tests decouple the program from its surroundings

• Serves as invaluable form of documentation
  – Shows others how to use our code

Test Isolation – Mock Objects

• How do we create a test when our system may depend on
  – A database to persist information
  – A third party simulator to perform calculations/functions
  – A printer to print output
  – A scanner or device to read input?

• We may implement our system with Mock Objects
Mock Objects

- A Mock Object
  - Provides the expected functionality
  - Isolates the code from details that may be filled in later
  - Speeds up development of test code
  - Can be refined incrementally by replacing with actual code

```
Test Code -> Service

Mock Service
```

Unit Testing

- Unit testing
  - Is more of an act of design than verification
  - Is more of an act of documentation than verification
  - Provides excellent feedback
Types of Tests

- White-box testing
  - Knows and depends on internal structure of modules being tested
  - Unit Testing
    - Drives the design
    - Validates changes made
    - Insufficient as verification tool however

- Black-box testing
  - Does not know and depend on internal structure of modules being tested
  - Acceptance testing
    - Written by customers, QA
    - Focuses on functionality of the system

Acceptance Testing

- Manual testing is not the preferred way

- Need to find ways to automate this as well

- Promotes separation of business logic from UI

- May be written using scripts, XML, etc.
Continuous Integration

- What good are the test cases if they are not run
- How often should we run them?
- Every night at least
- How about once every hour?
- Or better still when ever code change is checked in
- When code is checked in the code is compiled automatically and all tests cases are executed
  - If a test fails the team is alerted
  - When test fails, nothing else important/high priority
    - Fix the code to make the test succeed
    - Or modify the test to fit the changes if appropriate

Tools for Testing

- A number of tools are available
- A number of them are open source as well

- For Java and .NET
  - JUnit/NUnit
    - Automated Unit testing tool
  - Ant/NAnt
    - Automated build system
  - Cruise control/Cruise Control .NET
    - Continuous integration
A Test Driven Exercise

• Problem Statement
• Test code generation
• Coding and Design
• Testing and continuous integration
• Change

What is Refactoring?

• The Process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure

• Why fix what’s not broken?
  -- A software module
    • Should function its expected functionality
      -- It exists for this
    • It must be affordable to change
      -- It will have to change over time, so it better be cost effective
    • Must be easier to understand
      -- Developers unfamiliar with it must be able to read and understand it
A Refactoring Exercise

• Revisiting the code

• Improvements to be made

• Reasoning

• Benefits