Test-Driven Development and Refactoring

CPSC 315 – Programming Studio

Testing

- Discussed before, general ideas all still hold
- Test-Driven Development
  - Generally falls under Agile heading
  - A style of software development, not just a matter of testing your code
  - Enforces testing as part of the development process

Test Driven Development Overview

- Repeat this process:
  1. Write a new test
  2. Run existing code against all tests; it should generally fail on the new test
  3. Change code as needed
  4. Run new code against tests; it should pass all tests
  5. Refactor the code

Test Writing First

- Idea is to write tests, where each test adds some degree of functionality
- Passing the tests should indicate working code (to a point)
- The tests will ensure that future changes don't cause problems

Running Tests

- Use a test harness/testing framework of some sort to run the tests
  - A variety of ways to do this, including many existing frameworks that support unit tests
  - JUnit is the most well-known, but there is similar functionality across a wide range of languages

Test framework

- Specify a test *fixture*
  - Basically builds a state that can be tested
  - Set up before tests, removed afterward
- Test suite run against each fixture
  - Set of tests (order should not matter) to verify various aspects of functionality
  - Described as series of assertions
- Runs all tests automatically
  - Either passes all, or reports failures
    - Better frameworks give values that caused failure

Mock Objects

- To handle complex external queries (e.g. web services), random data, etc. in testing
- Implements an interface that provides some functionality
  - Can be complex on their own – e.g. checking order of calls to some object, etc.
  - Can control the effect of the interface

Example Mock Object

- Remote service
  - Interface to authenticate, put, get
  - Put and Get implementations check that authentication was called
  - Get verifies that only things that were “put” can be gotten.
- As opposed to an interface that just returned valid for authenticate/put, and returned fixed value for get.

Successful Tests

- Tests should eventually pass
- You need to check that **all** tests for that unit have passed, not just the most recent.

Refactoring

- As code is built, added on to, it becomes messier
- Need to go back and rewrite/reorganize sections of the code to make it cleaner
- Do this on a regular basis, or when things seem like they could use it
- Only refactor after all tests are passing
  - Test suite guarantees refactoring doesn't hurt.

Refactoring Common Operations

- Extract Class
- Extract Interface
- Extract Method
- Replace types with subclasses
- Replace conditional with polymorphic objects
- Form template
- Introduce "explaining" variable
- Replace constructor with "factory" method
- Replace inheritance with delegation
- Replace magic number with symbolic constant
- Replace nested conditional with guard clause

Resources

- *Test-Driven Development By Example*
  - Kent Beck; Addison Wesley, 2003
- *Test-Driven Development A Practical Guide*
  - David Astels; Prentice Hall, 2003
- *Software Testing A Craftsman's Approach (3rd edition)*
  - Paul Jorgensen; Auerback, 2008
- Many other books on testing, TDD, also