COMP 354
TDD and Refactoring

Greg Butler

Computer Science and Software Engineering
Concordia University, Montreal, Canada

Office: EV 3.219 Email: `gregb@cs.concordia.ca`

Winter 2015

Course Web Site:
http://users.encs.concordia.ca/˜gregb/home/comp354-w2015.html

Test-Driven Development (TDD)

Write tests first!

Why does it make sense to write tests first?

TDD in its pure form

Do not add *any* (product) code unless a test is failing.

Advantages of TDD

- Writing tests captures your understanding ofd requirements
- Tests actually get written
- Programmer satisfaction when all tests pass (why?)
- Repeatable and automated validation of correctness
- Confidence to change/refactor

TDD Enabling Technologies

JUnit

A framework for automated unit testing of Java code

Written by Erich Gamma (of Design Patterns fame) and Kent Beck (creator of XP methodology)

Uses Java 5 features such as annotations and static imports

Download from www.junit.org

TestCase Class Usage: Test Methods

Write a method for each test.

Method should be declared `public void`

Convention: method name starts with *"test"*

```
public void testGetTitle()
```

By following the naming convention, individual tests will be picked up automatically (by reflection)

A Test That Always Fails

```
public void testFailure() {
   fail();
}
```

Yields ...

```
junit.framework.AssertionFailedError
      at junit.framework.Assert.fail(Assert.java:47
      at junit.framework.Assert.fail(Assert.java:53)
      at MovieTest.testFailure(MovieTest.java:24)
      at java.lang.reflect.Method.invoke(Native Method)
```

Testing for Expected Values — A common test

```
public void testEmptyVectorSize() {
      Vector v = new Vector();
      assertEquals(
          "size should be 0", // msg
          0,                  // expected value
          v.size()            // actual value
     );
```

Other Assert Methods — no message provided

```
assertEquals(expected_value, actual_value)

assertNull(reference_type_expr)

assertNotNull(reference_type_expr)

assertTrue(boolean_expr)

assertFalse(boolean_expr)
assertSame(expected_ref, actual_ref)
```

Refactoring

A refactoring is a **change** made to the internal structure of S/W to make it easier to understand and less expensive to modify, **without changing its observable behavior**.

Behavior preserving (code) transformation

Eclipse has very effective (code) refactoring capabilities

Martin Fowler, Refactoring: improving the design of existing code, Addison-Wesley, 2000

Goal: Eliminate “Bad Smells” in Code and Design

Refactoring can help us gradually eliminate bad smells such as

Generic name
Duplication
Long method
Comments
Large class
Data class
Switch statements
Long parameter list

Martin Fowler, Refactoring: improving the design of existing code, Addison-Wesley, 2000

Refactoring — Examples from Larman [p.390]

Extract Method
Transform a long method into a shorter by factoring out a portion into a private helper method

Extract Constant
Replace a (hard-coded) literal constant with a constant variable

Introduce Explaining Variable
Put the result of the expression, or parts of it, in a temporary variable with a name that explains the purpose

Martin Fowler, Refactoring: improving the design of existing code, Addison-Wesley, 2000

Refactoring — Some advice from Fowler

When should I refactor? How often? How much time should I dedicate to it?

It is not something you should dedicate 2 weeks for every 6 months
... rather, you should do it as you develop!

Refactor ...

- when you recognize a warning sign (a *“bad smell”*)
- when you add a function
  — likely not an island unto itself
- when you fix a bug
  — is the bug symptomatic of a design flaw?
- when you do a code review
  — a good time to re-valuate your design, share opinions

Martin Fowler, Refactoring: improving the design of existing code, Addison-Wesley, 2000

TDD and Refactoring

What is the link between TDD and Refactoring?

Refactor, only, if all test cases have been written and all test cases pass

By re-running the test cases, the developer can be confident that refactoring is not damaging any existing functionality

Refactoring is part of a TDD micro-iteration

- An executable test for a particular feature, is designed, written and shown to fail
- The code is extended to pass the test while continuing to pass all other existing tests
- The code and tests are refactored to eliminate redundancy