CS 635 Advanced Object-Oriented Design & Programming
Spring Semester, 2011
Doc 3 Refactoring Intro
Jan 27, 2011

Copyright ©, All rights reserved. 2011 SDSU & Roger Whitney, 5500
Campanile Drive, San Diego, CA 92182-7700 USA. OpenContent (http://
www.opencontent.org/opl.shtml) license defines the copyright on this
document.
References

JUnit Web site: http://www.junit.org/


Brian Marick’s Testing Web Site: http://www.exampler.com/testing-com/

Testing for Programmers, Brian Marick, Available at: http://www.exampler.com/testing-com/writings.html

Refactoring: Improving the Design of Existing Code, Fowler, Addison-Wesley, 1999, chapters 1 & 3
Unit Testing
Testing

Johnson's Law

If it is not tested it does not work

The more time between coding and testing

  More effort is needed to write tests
  More effort is needed to find bugs
  Fewer bugs are found
  Time is wasted working with buggy code
  Development time increases
  Quality decreases
Unit Testing

Tests individual code segments

Automated tests
What wrong with:

Using print statements

Writing driver program in main

Writing small sample programs to run code

Running program and testing it be using it
We have a QA Team, so why should I write tests?
When to Write Tests

First write the tests

Then write the code to be tested

Writing tests first saves time

Makes you clear of the interface & functionality of the code

Removes temptation to skip tests
What to Test

Everything that could possibly break

Test values
   Inside valid range
   Outside valid range
   On the boundary between valid/invalid

GUIs are very hard to test
   Keep GUI layer very thin
   Unit test program behind the GUI, not the GUI
Common Things Programs Handle Incorrectly

Adapted with permission from “A Short Catalog of Test Ideas” by Brian Marick,
http://www.testing.com/writings.html

Strings

Empty String

Collections

Empty Collection
Collection with one element
Collection with duplicate elements
Collections with maximum possible size

Numbers

Zero
The smallest number
Just below the smallest number
The largest number
Just above the largest number
XUnit

Free frameworks for Unit testing

SUnit originally written by Kent Beck 1994

JUnit written by Kent Beck & Erich Gamma

Available at: http://www.junit.org/

Ports to many languages at:
  http://www.xprogramming.com/software.htm
XUnit Versions

3.x
Old version
Works with a versions of Java

4.x
Current version 4.8.1
Uses Annotations
Requires Java 5 or later
public class Adder {
    private int base;
    public Adder(int value) {
        base = value;
    }
    public int add(int amount) {
        return base + amount;
    }
}
Creating Test Case in Eclipse
Creating Test Case in Eclipse

Fill in dialog window & create the test cases
Test Class

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;
import org.junit.Test;

public class TestAdder {

    @Test
    public void testAdd() {
        Adder example = new Adder(3);
        assertEquals(4, example.add(1));
    }

    @Test
    public void testAddFail() {
        Adder example = new Adder(3);
        assertTrue(3 == example.add(1));
    }
}
Running the Tests
The result

![JUnit Test Result]

- **Finished after 0.028 seconds**
- **Runs:** 2/2  
- **Errors:** 0  
- **Failures:** 1

**TestAdder [Runner: JUnit 4] (0.001 s)**
- **testAdd (0.000 s)**
- **testAddFail (0.001 s)**
Assert Methods

assertArrayEquals()
assertTrue()
assertFalse()
assertEquals()
assertNotEquals()
assertSame()
assertNotSame()
assertNull()
assertNotNull()
fail()
Annotations

After
AfterClass
Before
BeforeClass
Ignore
Rule
Test
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertTrue;

import org.junit.Before;
import org.junit.Test;

public class TestAdder {
    Adder example;
    @Before
    public void setupExample() {
        example = new Adder(3);
    }

    @Test
    public void testAdd() {
        assertEquals(4, example.add(1));
    }
}
Refactoring
Refactoring

Changing the internal structure of software without changing its observable behavior

Done to make the software easier to understand and cheaper to modify
When to Refactor

Rule of three

Three strikes and you refactor
When to Refactor

- When you add a new function
- When you need to fix a bug
- When you do a code review
When Refactoring is Hard

Databases

Changing published interfaces

Major design issues
When you add a feature to a program

If needed Refactor the program to make it easy to add the feature

Then add the feature
Before you start refactoring

Make sure that you have a solid suite of tests

Test should be self-checking
Do I need tests when I use my IDEs refactoring tools?

Are your IDE refactoring tools bug free?
Code Smells
Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Duplicate Code

Thursday, January 27, 2011
Long Method - Large Class

The average method size should be less than 8 lines of code (LOC) for Smalltalk and 24 LOC for C++

The average number of methods per class should be less than 20

The average number of fields per class should be less than 6.

The class hierarchy nesting level should be less than 6

The average number of comment lines per method should be greater than 1
Long Parameter List

a.foo(12, 2, "cat", "<tr>", 19.6, x, y, classList, cutOffPoint)
Divergent Change

One class is changed in different ways for different reasons
ShotGun Surgery

When you have to make a kind of change you have to make a lot of little changes in different locations
Feature Envy

A method seems more interested in a class other than the one it is in.
Data Clumps

Same three or four data items together in lots of places
Primitive Obsession

Using primitive types instead of creating small classes
Switch Statements

How do you program without them?
Lazy Class

Class that is not doing enough to pay for itself
Data Class

Class with just fields and setter/getter methods

Data classes are like children.

They are okay as a starting point, but to participate as a grownup object, they need to take some responsibility
Inappropriate Intimacy

Classes that spend too much time delving into other classes private parts
Message Chains

location = rat.getRoom().getMaze().getLocation()
Negative Slope

if (foo) {
    if (bar) {
        if (cat = dog) {
            if (rat < 10) {
                ...
            ...
        }
    }
}
Temporary Field

Field is only used in certain circumstances

Common case
field is only used by an algorithm
Don't want to pass around long parameter list
Make parameter a field
Refused Bequest

Subclass does not want to support all the methods of parent class

Subclass should support the interface of the parent class
Eclipse Refactoring
Eclipse Refactoring Menu

<table>
<thead>
<tr>
<th>Rename...</th>
<th>Edit</th>
<th>Move...</th>
<th>Edit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change Method Signature...</td>
<td>Edit</td>
<td>Extract Method...</td>
<td>Edit</td>
</tr>
<tr>
<td>Extract Local Variable...</td>
<td>Edit</td>
<td>Extract Constant...</td>
<td></td>
</tr>
<tr>
<td>Inline...</td>
<td>Edit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Convert Anonymous Class to Nested...</td>
<td></td>
<td>Convert Member Type to Top Level...</td>
<td></td>
</tr>
<tr>
<td>Convert Local Variable to Field...</td>
<td></td>
<td>Extract Superclass...</td>
<td></td>
</tr>
<tr>
<td>Extract Interface...</td>
<td></td>
<td>Use Supertype Where Possible...</td>
<td></td>
</tr>
<tr>
<td>Push Down...</td>
<td></td>
<td>Pull Up...</td>
<td></td>
</tr>
<tr>
<td>Extract Class...</td>
<td></td>
<td>Introduce Parameter Object...</td>
<td></td>
</tr>
<tr>
<td>Introduce Indirection...</td>
<td></td>
<td>Introduce Factory...</td>
<td></td>
</tr>
<tr>
<td>Introduce Parameter...</td>
<td></td>
<td>Encapsulate Field...</td>
<td></td>
</tr>
<tr>
<td>Generalize Declared Type...</td>
<td></td>
<td>Infer Generic Type Arguments...</td>
<td></td>
</tr>
<tr>
<td>Migrate JAR File...</td>
<td></td>
<td>Create Script...</td>
<td></td>
</tr>
<tr>
<td>Apply Script...</td>
<td></td>
<td>History...</td>
<td></td>
</tr>
</tbody>
</table>
public class Foo {
    public int foo() {
        return 10;
    }
}

public class Bar {
    public int bar() {
        Foo test = new Foo();
        return test.foo() + 99;
    }
}

public class NewFoo {
    public int foo() {
        return 10;
    }
}

public class Bar {
    public int bar() {
        NewFoo test = new NewFoo();
        return test.foo() + 99;
    }
}
Eclipse Rename

```java
package edu.sdsu.cs;

/**
 * Class Foo
 */
public class Foo {
    public int foo() {
        return 10;
    }
}
```
public class Bar {
    public int helperMethod(Foo test) {
        return test.foo() + test.fooTwo();
    }

    public int callHelper() {
        Foo data = new Foo();
        return helperMethod(data);
    }
}

public class Foo {
    public int foo() { return 10; }
    public int fooTwo() { return 20; }
    public int sum() {
        return foo() + fooTwo();
    }
}

public class Bar {
    public int callHelper() {
        Foo data = new Foo();
        return data.sum();
    }
}

public class Foo {
    public int foo() { return 10; }
    public int fooTwo() { return 20; }
    public int sum() {
        return foo() + fooTwo();
    }
}
Eclipse Move

```java
package edu.sdsu.cs;

class Foo {
    public int foo() {
        return 10;
    }

    public int fooTwo() {
        return 20;
    }
}
```
Extract Class

```java
package edu.sdsu.cs;

public class BinaryTree {
    private BinaryTree root;
    private String value;
    private BinaryTree left;
    private BinaryTree right;

    public boolean hasValue(String value) {
        return hasValue(root, value);
    }

    private boolean hasValue(BinaryTree tree, String value) {
        if (0 == tree.value.compareTo(value))
            return true;
        if (-1 == tree.value.compareTo(value))
            return hasValue(tree.right, value);
        if (0 == tree.value.compareTo(value))
            return hasValue(tree.left, value);

        return false;
    }
}
```
Refactoring Tool Issue

People tend to only use the features they know
Refactoring Tool Issue

Is a tool hard to use because I am unfamiliar with it or is it just hard to use
### Refactoring by 41 Professional Programmers

<table>
<thead>
<tr>
<th>Refactoring</th>
<th>Number of Programmers used Refactoring</th>
<th>Total Times used</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntroduceFactory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>PushDown</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>UseSupertype</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>EncapsulateField</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Introduce Parameter</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>Convert Local to Field</td>
<td>5</td>
<td>37</td>
</tr>
<tr>
<td>Extract Interface</td>
<td>10</td>
<td>26</td>
</tr>
<tr>
<td>Inline</td>
<td>11</td>
<td>185</td>
</tr>
<tr>
<td>Modify Parameters</td>
<td>11</td>
<td>79</td>
</tr>
<tr>
<td>Pull up</td>
<td>11</td>
<td>37</td>
</tr>
<tr>
<td>Extract Method</td>
<td>20</td>
<td>344</td>
</tr>
<tr>
<td>Move</td>
<td>24</td>
<td>212</td>
</tr>
<tr>
<td>Rename</td>
<td>41</td>
<td>2396</td>
</tr>
</tbody>
</table>
Try In Eclipse

Rename
Move
Encapsulate Field
Extract Method
Extract Class