Test-Driven Design
for Introductory OO Programming

Viera Krňanová Proulx

Northeastern University

vkp@ccs.neu.edu

SIGCSE 2009, Chattanooga, TN
Disclaimer

This is not TDD

TeachScheme/ReachJava curriculum

• Curriculum that focuses on data driven design
• Design Recipe enforces examples/tests first
• Design of tests reinforces student learning
• Design of tests improves program structure
• Impossible without software support for testing

The tester library team:

Viera K. Proulx, Weston Jossey
Outline:

Design Recipe: Tests/Examples first

• Anatomy of a Unit test
• JUnit is not the answer

The Prima tester library

• Extensibility
• Future Plans -- Avanti
• Our experiences
Why think about testing?

Testing is hard
Java does not support comparing data by value
Defining such equality is hard for a novice
It increases the program complexity
Detracts from the focus on the program design

Learning to design tests, equality comparison, test reporting
• is a topic on its own
• we need pedagogy for that too

But: testing should be integrated into program design early
Testing Support

Tester library

Tests are written as a part of the program design

Test library suitable for the beginner

- Tests compare data by their values
  - handle collections of data
  - handle circularity
  - handle random choice
  - handle tests of Exceptions
  - ... and more

- Test evaluation is automatic - compares data by their values
Designing a Method: The DESIGN RECIPE

Follow these steps for every method:

• 1: Problem analysis and data definition
• 2: Purpose statement and the header
• 3: Examples with expected outcomes
• 4: Inventory/Template of available data fields and methods
• 5: Method body
• 6: Tests

Pedagogical advantages:

Each step is well defined
-- with a tangible result
-- with a guidance on what questions to ask
Test-First Advantages

Design tests first

• understand what data is needed for the method
• understand what are the expected outcomes
• gain insight into how the method behaves

Evaluating the tests

• define additional tests based on method design
• know that the expected behavior works
• for failed tests see what went wrong

Benefits

• simple methods -- simple tests
Anatomy of a Unit Test

How to set up, what to compare
• define examples of data and expected results
• invoke the method
• compare actual and expected

How to evaluate the test cases
• define methods to compare actual and expected

How are the results reported
• display which tests failed
• provide a link to failed tests
• show why the tests failed
Why not JUnit?

Defining the tests

• *extends* `TestCase` before you see inheritance
• no access to private methods, fields
• new syntax, language

Evaluation of the test cases

• define your own `equals` method

Reporting of the results

• JUnit bar: red or green
• links to the line where the test failed
• expected:<BTNode@c66e698d> but was:<BTNode@52d68153>
Why not JUnit?

The key problems for a novice:

• extra language, syntax
• the need to define `equals` method
• Test result reporting uninformative

of course, we can teach students to do this...
they should learn how to do this...

... but not in their first week of Java
Let's compare:

Binary Search Tree: ABST, Node, Leaf
• Test the add method - build a tree
• we want to make sure the tree is built correctly
• the test should compare two trees

Defining equals method
• Three classes: needs to use `getClass`
• Should override `hashCode`
• ... and test that both work correctly ...

Define toString method
• to make sure the results are meaningful
Define equals only:

```java
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Node<T> other = (Node<T>) obj;
    if (data == null) {
        if (other.data != null)
            return false;
    } else if (!data.equals(other.data))
        return false;
    if (left == null) {
        if (other.left != null)
            return false;
    } else if (!left.equals(other.left))
        return false;
    if (right == null) {
        if (other.right != null)
            return false;
    } else if (!right.equals(other.right))
        return false;
    return true;
}
```
Test case definition:

```java
// test the method add that builds ABST
public void testInsert(){
    assertEquals(tree3, tree3a);
    assertEquals(tree4, tree4a);
    assertEquals(tree4, tree4b);
}

// test the method add that builds ABST
public void testTreeAdd(Tester t){
    t.checkExpect(tree3, tree3a, "same trees");
    t.checkExpect(tree4, tree4a, "same trees");
    t.checkExpect(tree4, tree4b, "different trees");
}
```
Test results:

<table>
<thead>
<tr>
<th>Failure Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>junit.framework.AssertionFailedError: expected: <a href="mailto:Node@2206331a">Node@2206331a</a> but was: &lt;</td>
</tr>
<tr>
<td>at TreeTests.testInsert(TreeTests.java:51)</td>
</tr>
</tbody>
</table>
Test results:

Examples:

new Examples:1:
  this.cms = Book: Old Man and the Sea, by Hemingway
new Book: 2:
  this.title = "Old Man and the Sea"
  this.author = "Hemingway"
this.eos = Book: Elements of Style, by EBW
new Book: 3:
  this.title = "Elements of Style"
  this.author = "EBW"
this.htdp = Book: HtDP, by MF
new Book: 4:
  this.title = "HtDP"
  this.author = "MF"
this.lil = Book: Little Lisper, by MF
new Book: 5:
  this.title = "Little Lisper"
  this.author = "MF"
this.booksByTitle =
  new Book$BookByTitle: 6()
this.leaf =
  new Leaf: 7(
    this.comp = Book$BookByTitle: 6)
this.tree3 =
  new Node: 8(
    this.comp = Book$BookByTitle: 6
    this.data = Book: 4
    this.left =
      new Node: 9(
        this.comp = Book$BookByTitle: 6
        this.data = Book: 3
        this.left = Leaf: 7
        this.right = Leaf: 7)
    this.right =
      new Node: 10(
        this.comp = Book$BookByTitle: 6
        this.data = Book: 2
        this.left = Leaf: 7
        this.right = Leaf: 7))
this.tree4 =
  new Node: 11(
    this.comp = Book$BookByTitle: 6
    this.data = Book: 4
    this.left =
      new Node: 12(
        this.comp = Book$BookByTitle: 6
        this.data = Book: 2
        this.left =
          new Node: 13(
            this.comp = Book$BookByTitle: 6
            this.data = Book: 5
            this.left = Leaf: 7
            this.right = Leaf: 7)
        this.right = Leaf: 7))
this.tree3a = Node: 8
this.tree4a = Node: 11
this.tree4b =
  new Node: 14(
    this.comp = Book$BookByTitle: 6
    this.data = Book: 4
    this.left = Node: 9
    this.right =
      new Node: 15(
        this.comp = Book$BookByTitle: 6
        this.data = Book: 2
        this.left = Leaf: 7
        this.right = Node: 13))

-----------------------
Found 2 test methods
-----------------------
-----------------------
Ran 10003 tests.
1 test failed.
Test results:

--- END OF TEST RESULTS ---
Prima tester library

Functionality

Uses reflection to compare arbitrary objects

Easy setup for a novice: **Examples** class

No equals - all equality evaluation is done by the library

No **toString** method needed - done automatically

A variety of reporting options

• failed tests only

• all tests

• pretty-print all data

always shows user-defined **toString** with pretty-print
Prima tester library

A wide range of test scenarios

• compare any two objects, including circularly defined
• compare two `Iterable` objects
• compare two `Map` objects
• `checkOneOf` a random set of values
• `checkRange` value within the given range: `Comparator`
• `checkNumRange` mixed numeric ranges
• test if a method throws exception with the given message
• `checkFail` for test we want to fail
Prima tester library

User options

- user can define several classes with test methods
- user can implement own equality:
  ```java
  interface ISame<T>{ boolean same(T t); }
  ```
- `checkEquivalence` user implements `Equivalence` interface
  ```java
  interface Equivalence<T>{
      boolean equivalent(T t1, T t2); }
  ```
- user can annotate any method to be a test method
- user can include test methods within class definition
  - this provides access to `private` fields and methods
- `Printer.print(Object obj)` pretty-prints any object
Avanti tester library

Goal: support seasoned programmer

... learning ground for defining own tests, evaluation ...

includes a coverage tool

special support for testing effects

special support for comparing data structures
Our Experiences

Tester library

Classroom trials:

Spring 2008 -- the first prototype
Fall 2008 -- beta version used at five institutions
  - Northeastern University
  - Worcester Polytechnic Institute, Worcester, MA
  - Seton Hall University, South Orange, NJ
  - duPont Manual High School, Louisville KY
  - Millard Public Schools, Omaha, NE
Spring 2009 -- fully deployed, new users added
  - Vassar College, Poughkeepsie, NY -- in a regular Java course
Our Experiences

Tester library

Used with hundreds of students throughout the semester

• Students get real feedback on validity of their programs
• Students believe testing matters
• Students understand why smaller methods are better
• Students explore the design of the tester
• New appreciation of the meaning of equality
• Some get excited about testing!

Colleague reports uniformly positive
Our Experiences -- curriculum overall

Yearly surveys done for over 10 years:

Coop employers report higher expectations of students
Students exceed even the higher expectations

Instructors in follow-up courses: students are better prepared
on pretest 30 percent failure reduced to 1 percent

Very low attrition rate (<5%)

Students are much more confident in their understanding of program design

Dissemination:

Workshop in summer 2007, 2008, 2009 at four US locations

A growing number of followers
Summer Workshops

Redesigning Introductory Computing: The Design Discipline

NSF supported: DUE-0618543

One week, hands on workshops for faculty

Free travel, room and moard, materials

In Boston, four levels: TS/RJ, TS2, RJ, Bootstrap

Join us ...

Garden City (Long Island), NY  Adelphi University  June 22-26
San Luis Obispo, CA  Cal Poly  June 28-July 2
Boston, MA  Northeastern/WPI/Brown  July 20-24
Salt Lake City UT  University of Utah  August 3-7

For info:  http://www.teach-scheme.org
THANK YOU

Resources

Tester library:

Tester library, World libraries:  http://www.ccs.neu.edu/javalib
Tester library:  http://code.google.com/p/nutester/

Resources: TeachScheme/ReachJava curriculum:

Main site for the TeachScheme/ReachJava! project:
• http://www.teach-scheme.org

Lab materials, lecture notes, assignments:
• http://www.ccs.neu.edu/home/vkp/HtDC.html

(World libraries support the design of interactive games with graphics, timer, and key events)