CS159

Nathan Sprague

February 2, 2015

Testing Happens at Multiple Levels

- Unit Testing - Test individual classes in isolation.
    - Focus is on making sure that each method works according to specification.
- Integration Testing - Test the interaction between classes.
- Validation Testing - Test the entire system in context.

Different Perspectives

- Black-box testing
  - Develop tests on the basis of class specifications and documentation.
- White-box testing
  - Develop tests on the basis of implementation.
  - Aim for high “code coverage”.

Testing Example

```java
public class Estimator
{
    public static int totalCost(boolean fast, boolean good)
    {
        int total = 0;
        if (fast)
        {
            total += 5;
        }
        if (good)
        {
            total += 10;
        }
        return total;
    }
}
```

Method Coverage

- 100% method coverage: testing code calls each method at least once.

```
    @Test
    public void totalCostTestSlowBad() {
        assertEquals(0, Estimator.totalCost(false, false));
    }
```

- Done! Reassuring?

Method Coverage

- 100% method coverage: testing code calls each method at least once.

```
    @Test
    public void totalCostTestSlowBad() {
        assertEquals(0, Estimator.totalCost(false, false));
    }
```

- Done! Reassuring?
- No, but better than NOT having 100% method coverage.

Statement Coverage

- 100% statement coverage: testing executes every statement.

```java
@Test
public void totalCostTestSlowBad() {
    assertEquals(0, Estimator.totalCost(false, false));
}

@Test
public void totalCostTestFastGood() {
    assertEquals(15, Estimator.totalCost(true, true));
}
```

- Better? Happy?

Path Coverage

- 100% path coverage: testing exercises every possible path through the code.

```
    @Test
    public void totalCostTestSlowBad() {
        assertEquals(0, Estimator.totalCost(false, false));
    }
    @Test
    public void totalCostTestFastGood() {
        assertEquals(15, Estimator.totalCost(true, true));
    }
    @Test
    public void totalCostTestSlowGood() {
        assertEquals(10, Estimator.totalCost(false, true));
    }
    @Test
    public void totalCostTestFastBad() {
        assertEquals(5, Estimator.totalCost(true, false));
    }
```

Path Coverage

- 100% path coverage is typically considered an impractical target.
- It is a useful idea to have in mind while developing tests.

Test-Driven Development

- Write tests first.
  - Helps clarify specifications.
  - Helps avoid mistakes in development.

Developing Test Cases

- To *guarantee* correctness, test every possible sequence of method calls, with every possible input value.
  - Usually not possible.
- Instead, look for boundary conditions
  - Points where the behavior of the code should change
  - Test at the boundaries and on either side.
- Also test erroneous inputs
- We'll work through an example in a few minutes...

Regression Testing

- Testing is not a one-time process.
- Ideally, unit tests are maintained along with the code.
- This makes it safer to change the code:
    - All tests can be run after every change.

Brainstorm Some Tests...

```java
/**
 * Returns the point with the smallest x-coordinate
 * among all points in the array. In the case of a tie,
 * the point that appears first will be returned.
 *
 * @param points - An array of point objects
 * @return - The leftmost point
 * @throws - IllegalArgumentException If the length
 *                                   of the array is 0.
 * @throws - NullPointerException If the array, or any
 *                                entries in the array,
 *                                are null.
 */

public static Point findLeftmost(Point[] points)
          throws IllegalArgumentException,
                 NullPointerException
```