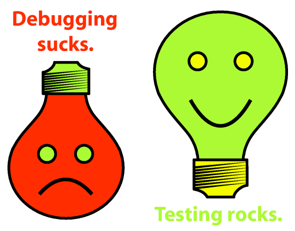

Testing on the Toilet

Mar. 20, 2008

~~Testing~~ TestNG on the Toilet

Recently, somewhere in the Caribbean Sea, you implemented the PirateShip class. You want to test the cannons thoroughly in preparation for a clash with the East India Company. This requires that you run the crucial **`testFireCannonDepletesAmmunition()`** method many times with many different inputs.

TestNG is a test framework for Java unit tests that offers **additional power and ease of use** over JUnit. Some of TestNG's features will help you to write your PirateShip tests in such a way that you'll be well prepared to take on the Admiral. First is the **`@DataProvider`** annotation, which allows you to add parameters to a test method and provide argument values to it from a data provider.

```
public class PirateShipTest {
  @Test(dataProvider = "cannons")
  public void testFireCannonDepletesAmmunition(int ballsToLoad, int ballsToFire,
      int expectedRemaining) {
    PirateShip ship = new PirateShip("The Black Pearl");
    ship.loadCannons(ballsToLoad);
    for (int i = 0; i < ballsToFire; i++) {
      ship.fireCannon();
    }
    assertEquals(ship.getBallsRemaining(), expectedRemaining);
  }
  @DataProvider(name = "cannons")
  public Object[][] getShipSidesAndAmmunition() {
    // Each 1-D array represents a single execution of a @Test that refers to this
    // provider. The elements in the array represent parameters to the test call.
    return new Object[] {
      {5, 1, 4}, {5, 5, 0}, {5, 0, 5}
    };
  }
}
```

Now let's focus on making the entire test suite run faster. An old, experienced pirate draws your attention to TestNG's capacity for running tests in parallel. You can do this in the definition of your test suite (described in an XML file) with the **`parallel`** and **`thread-count`** attributes.

```
<suite name="PirateShip suite" parallel="methods" thread-count="2">
```

A great pirate will realize that this parallelization can also help to expose race conditions in the methods under test.

Now you have confidence that your cannons fired in parallel will work correctly. But you didn't get to be a Captain by slacking off! You know that it's also important for your code to **fail** as expected. For this, TestNG offers the ability to specify those exceptions (and only those exceptions) that you expect your code to throw.

```
  @Test(expectedExceptions = { NoAmmunitionException.class })
  public void testFireCannonEmptyThrowsNoAmmunitionException() {
    PirateShip ship = new PirateShip("The Black Pearl");
    ship.fireCannon();
  }
```

More information, discussion, and archives:
http://googletesting.blogspot.com