Unit Tests

Frank Sholey

3 Laws of TDD

1. You may not write production code until you have written a failing unit test.
2. You may not write more of a unit test than is sufficient to fail, and not compiling is failing.
3. You may not write more production code than is sufficient to pass the currently failing test.

Keep tests clean!

- “Test code is just as important as production code.”
- Tests must constantly change as the code evolves.
- If tests are dirty, they are harder to change and will begin to fail as the code evolves.
- More time spent trying to fix tests -> no tests

  -> defective code -> bad software

Keep tests clean!

Hard to read, too many details

```
public void testGetPageHieratchyAsXml() throws
        Exception {
        crawler.addPage(root, PathParser.parse("PageOne"));
        crawler.addPage(root,
        PathParser.parse("PageOne.ChildOne"));
        crawler.addPage(root, PathParser.parse("PageTwo"));
        request.setResource("root");
        request.addInput("type", "pages");
        Responder responder = new SerializedPageResponder();
        SimpleResponse response =
        (SimpleResponse) responder.makeResponse(
        new FitNesseContext(root), request);
        String xml = response.getContent();
        assertEquals("text/xml", response.getContentType());
        assertSubString("<name>PageOne</name>", xml);
        assertSubString("<name>PageTwo</name>", xml);
        assertSubString("<name>ChildOne</name>", xml);
}
```

Clean and explanatory

```
public void testGetPageHierarchyAsXml() throws Exception {
        makePages("PageOne", "PageOne.ChildOne", "PageTwo");
        submitRequest("root", "type:pages");
        assertResponseIsXML();
        assertResponseContains(
          "<name>PageOne</name>", "<name>PageTwo</
        name>",       "<name>ChildOne</name>"
        );
}
```

Don’t be tedious and redundant

Tedious and redundant

```
@Test
public void turnOnLoTempAlarmAtThreashold()
        throws Exception {
    assertTrue(hw.heaterState());
    assertTrue(hw.blowerState());
    assertFalse(hw.coolerState());
    assertFalse(hw.hiTempAlarm());
    assertTrue(hw.loTempAlarm());
}
```

Easy to read and understand

```
@Test
public void turnOnLoTempAlarmAtThreshold()
        throws Exception {
    assertEquals("HBchL", hw.getState());
}

// Uppercase = on
// Lowercase = off
```

How should I write Unit Tests?

One Assert per Test?

- Single conclusion
- Quick and easy to understand

- Could cause repeated code
- Sometimes easier to merge assertions into the same test

Single Concept per Test?

- Everything goes well together
- Doesn't jump around from one miscellaneous thing to another.
- Easy to understand

Good Rule of thumb: Minimize the number of asserts per concept and test just one concept per test function.

F.I.R.S.T.

- **F**ast - Tests should be fast and run quickly
- **I**ndependent - Tests should not depend on each other. One test should not set up the conditions for the next test. Any order.
- **R**epeatable – Tests should be repeatable in any environment.
- **S**elf-Validating – The tests should have a boolean output. Either they pass or fail.
- **T**imely – The tests need to be written in a timely fashion. Unit tests should be written *just before* the production code that makes them pass.

Summary

- Tests are just as important as production code.
- Tests preserve and enhance the flexibility, maintainability, and reusability of the production code.
- Keep tests constantly clean.
- “If you let the tests rot, then your code will rot too.”
- Keep the 3 laws of TDD and the FIRST rules in mind when writing tests.