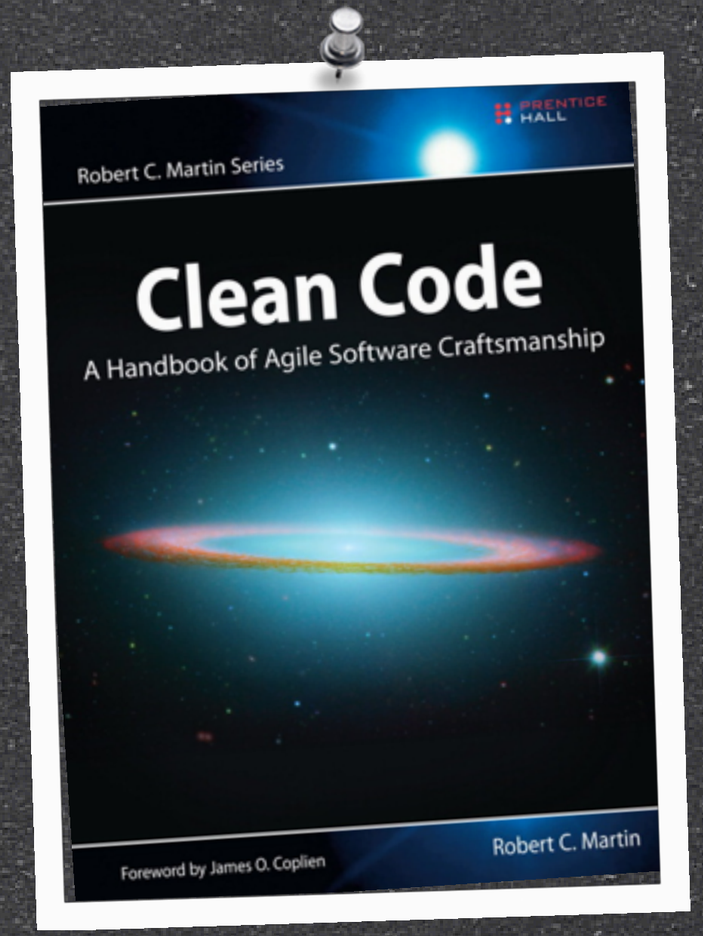
Clean Code
“Unit Tests”



Mike Christianson http://codeaweso.me

The Three Laws of TDD

The Three Laws of TDD

- First Law - You may not write production code until you have written a failing unit test.

The Three Laws of TDD

- First Law - You may not write production code until you have written a failing unit test.
- Second Law - You may not write more of a unit test than is sufficient to fail, and not compiling is failing.

The Three Laws of TDD

- First Law - You may not write production code until you have written a failing unit test.
- Second Law - You may not write more of a unit test than is sufficient to fail, and not compiling is failing.
- Third Law - You may not write more production code than is sufficient to pass the currently failing test.

TDD’s “Fallout 3”

TDD's "Fallout 3"

- Every Day - Dozens of tests

TDD's "Fallout 3"

- Every Day - Dozens of tests
- Every Month - Hundreds of tests

TDD's "Fallout 3"

- Every Day - Dozens of tests
- Every Month - Hundreds of tests
- Every Year - Thousands of tests

TDD's "Fallout 3"

- Every Day - Dozens of tests
- Every Month - Hundreds of tests
- Every Year - Thousands of tests

THAT'S A LOT OF CODE!

Keeping Tests Clean

- Test code is just as important as production code. Perhaps more so.
- Remember the “broken windows” theory.

Clean Tests

- Readability
- Readability
- Readability

Clean Tests

- Readability
- Readability
- Readability
- Clarity
- Simplicity
- Density of Expression

Build-Operate-Check

- Build up test data
- Operate on test data
- Check for expected results

Build-Operate-Check

```
public void testGetDataAsXML() throws Exception {
  makePageWithContent(“TestPageOne”, “test page”);

  submitRequest(“TestPageOne”, “type:data”);

  assertResponseIsXML();
  assertResponseContains(“test page”, “<Test”);
}
```

Build-Operate-Check

```
public void testGetDataAsXML() throws Exception {
  makePageWithContent(“TestPageOne”, “test page”);

  submitRequest(“TestPageOne”, “type:data”);

  assertResponseIsXML();
  assertResponseContains(“test page”, “<Test”);
}
```

ANY OTHER “CLEAN CODE” PRACTICES?

Which do you prefer?

```
@Test
public void turnOnLoTempAlarmAtThreshold() {
  hw.setTemp(WAY_TOO_COLD);
  controller.tic();
  assertTrue(hw.heaterState());
  assertTrue(hw.blowerState());
  assertFalse(hw.coolerState());
  assertFalse(hw.hiTempAlarm());
  assertTrue(hw.loTempAlarm());
}
```

```
@Test
public void turnOnLoTempAlarmAtThreshold() {
  wayTooCold();
  assertEquals(“HBchL”, hw.getState());
}
```

Dual Standards

- Memory or CPU efficiency usually matter less within a test.
- But, always, cleanliness remains.

One Assert per Test

- It’s more what you'd call a "guideline" than actual rule.
- Strategies for removing duplicate code with the goal of following the “rule” can complicate tests.
  - Template Method pattern
  - Separate class, use @Before

Single Concept per Test

- An actual rule, not a guideline
- A continuation of prior “clean code” concepts
- Violation causes confusion and delay for the reader

Single Concept per Test

```
@Test
public void test() throws ParseException {
  GroupValidator validator = new GroupValidator(LOCATION, "WIII");
  LinkedHashMap<Group,String> groups =
    ReportParser.parse("METAR WIII 172145Z AUTO VRB02KT 9999 +SHRA +FC FZDZ +TSRAGR " +
        " +BKN030 FEW075 SCT100 OVC250 11/06 Q1021", validator);
  assertEquals("172145Z", groups.get(TIMESTAMP));
  assertEquals("AUTO", groups.get(AUTO_COR));
  assertEquals("VRB02KT", groups.get(WIND));
  assertEquals("9999", groups.get(VISIBILITY));
  assertEquals("+SHRA +FC FZDZ +TSRAGR", groups.get(WEATHER));
  assertEquals("BKN030 FEW075 SCT100 OVC250", groups.get(SKY_CLOUD));
  assertEquals("11/06", groups.get(TEMP_DEWPOINT));
  assertEquals("Q1021", groups.get(PRESSURE));
  assertEquals("METAR AUTO", groups.get(FULL_TYPE));
}
```

Single Concept per Test

```
@Test
public void test() throws ParseException {
  GroupValidator validator = new GroupValidator(LOCATION, "WIII");
  LinkedHashMap<Group,String> groups =
    ReportParser.parse("METAR WIII 172145Z AUTO VRB02KT 9999 +SHRA +FC FZDZ +TSRAGR " +
        " +BKN030 FEW075 SCT100 OVC250 11/06 Q1021", validator);
  assertEquals("172145Z", groups.get(TIMESTAMP));
  assertEquals("AUTO", groups.get(AUTO_COR));
  assertEquals("VRB02KT", groups.get(WIND));
  assertEquals("9999", groups.get(VISIBILITY));
  assertEquals("+SHRA +FC FZDZ +TSRAGR", groups.get(WEATHER));
  assertEquals("BKN030 FEW075 SCT100 OVC250", groups.get(SKY_CLOUD));
  assertEquals("11/06", groups.get(TEMP_DEWPOINT));
  assertEquals("Q1021", groups.get(PRESSURE));
  assertEquals("METAR AUTO", groups.get(FULL_TYPE));
}
```

THOUGHTS? IDEAS? IS IT "CLEAN"?

F.I.R.S.T

- Fast - Tests should be fast and run quickly.
- Independent - Test should not depend on each other and be able to run in any order.
- Repeatable - Test should be repeatable in any environment.
- Self Validating - Tests should have a boolean output: pass or fail.
- Timely - The tests need to be written in a timely fashion, that is, just before production code.