Unbearable Test Smells

Steven Mak
steven@odd-e.com
www.odd-e.com
twitter: stevenmak

Who am I?

- Name: Steven Mak
- Agile Coach at Odd-e
- Lives in Hong Kong
- Agile/Scrum, TDD Coaching
- I love coding - Java, C/C++, PHP, Perl, C#, VB, and some weird languages

Copy and Paste Code

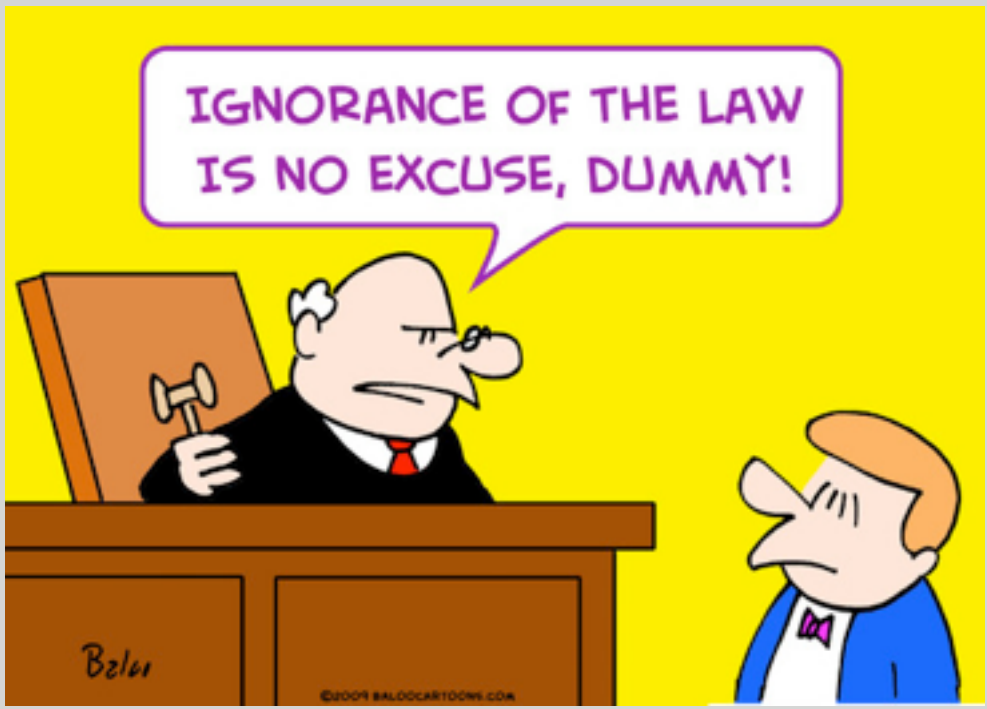
Long test codes are copied and pasted somewhere else with only a few lines changing

DRY

Don’t Repeat Yourself!
Don’t Repeat Yourself!
Don’t Repeat Yourself!
Don’t Repeat Yourself!
Don’t Repeat Yourself!
Don’t Repeat Yourself!
Don’t Repeat Yourself!
Don’t Repeat Yourself!

Not knowing the fixtures

Some initialisation and clean up codes that are repeated in each tests...



What is fixture?

```
TEST_GROUP (TEST_thisObject)
{
        void setup() {
        }
        void teardown() {
        }
};
```

Duplication causing fragile tests

Where is the duplication?

EXPECT_LOG(“ABC error”);

Duplication causing fragile tests

Where is the duplication?

EXPECT_LOG("ABC error");

So there is a line in code that prints this log message

Duplication causing fragile tests

Put it under centralise header file:

#define ABC_ERROR_WITH_EC “ABC error”

The test will then look like:

EXPECT_LOG(ABC_ERROR);

Over-Optimism?

Tests that forgot to cover exceptional cases or just covered the easiest condition

```
if (aaa() || bbb() || ccc() {
        ...
  } else {
        ...
  }
```

Tests don’t have assertions

```
TEST(TEST_GROUP, TEST_THIS)
{
        runThisFunctionLaLaLa();
}
```

What does it mean by 80% Unit Test Coverage?

Why xUnits don't have CHECK_NOT_EQUAL?

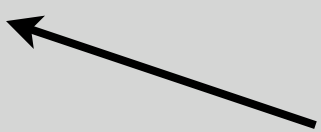
What is the problem with:

CHECK(TRUE, xxx != 3);

Why xUnits don't have CHECK_NOT_EQUAL?

What is the problem with:

CHECK(TRUE, xxx != 3);



Is there any good reason why you cannot know the output value?

So, tell me what it is then.

OK, fine, so I use CHECK with a specific output value, what now?

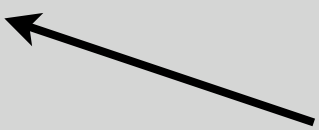
What is the problem with:

CHECK(TRUE, xxx == 4);

OK, fine, so I use CHECK with a specific output value, what now?

What is the problem with:

CHECK(TRUE, xxx == 4);

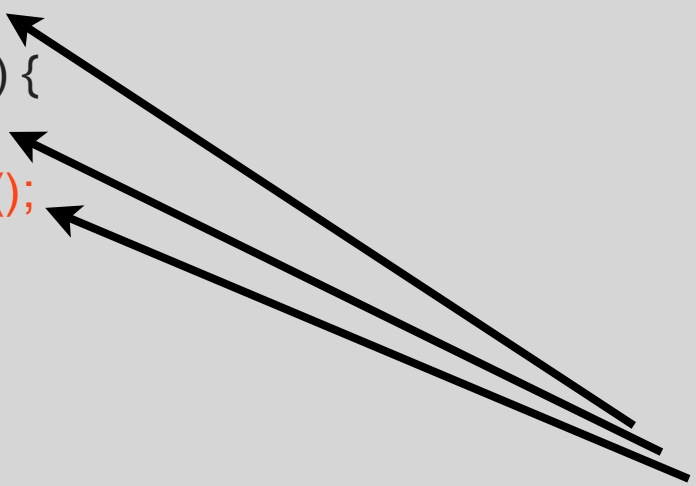


In most xUnits, we have LONGS_EQUAL telling you the actual value when it goes wrong instead of a "false"

Do you know your xUnit harness?

Further example

```
try {
        readConfigurationFile();
        assertTrue(true);
} catch (IOException e) {
        assertTrue(false);
        e.printStackTrace();
}
```



These are the places you know your team does not know the test harness.

Some xUnit harness

- Java: JUnit
- .Net: NUnit
- C/C++: CppUTest
- PHP: PHPUnit

What's wrong?

What is the problem with:

TEST(TEST_AIH, FAIL_BAD_PARAM)

Names don't really tell

What is the problem with:

TEST(TEST_AIH, FAIL_BAD_PARAM)

Be more precise about how it triggered the failure

What names tell us?

- Who
  - Name of the SUT class
  - Name of the method or feature being exercised
- Input
  - Important characteristics of any input values
  - Anything relevant about the state
- Output
  - The outputs expected
  - The expected post-exercise state

Conditional Test Logic?

```
// verify Vancouver is in the list
actual = null;
i = flightsFromCalgary.iterator();
while (i.hasNext()) {
        FlightDto flightDto = (FlightDto) i.next();
        if (flightDto.getFlightNumber().equals(
                expectedCalgaryToVan.getFlightNumber()))
        {
                actual = flightDto;
                assertEquals("Flight from Calgary to Vancouver",
                        expectedCalgaryToVan,
                        flightDto);
                break;
        }
}
```

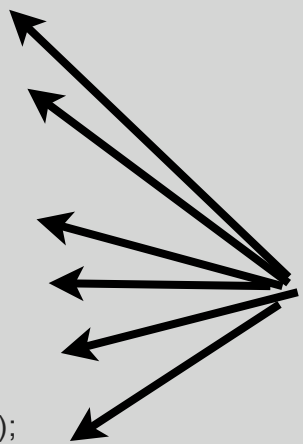
Tests that crash 50% of the time?!!

Testing everything at a time

```
public void testFlightMileage_asKm2() throws Exception {
        // set up fixture
        // exercise constructor
        Flight newFlight = new Flight(validFlightNumber);
        // verify constructed object
        assertEquals(validFlightNumber, newFlight.number);
        assertEquals("", newFlight.airlineCode);
        assertNull(newFlight.airline);
        // set up mileage
        newFlight.setMileage(1122);
        // exercise mileage translator
        int actualKilometres = newFlight.getMileageAsKm();
        // verify results
        int expectedKilometres = 1810;
        assertEquals( expectedKilometres, actualKilometres);
        // now try it with a canceled flight
        newFlight.cancel();
        try {
                newFlight.getMileageAsKm();
                fail("Expected exception");
        } catch (InvalidRequestException e) {
                assertEquals( "Cannot get cancelled flight mileage",
                e.getMessage());
        }
}
```

Testing everything at a time

```
public void testFlightMileage_asKm2() throws Exception {
        // set up fixture
        // exercise constructor
        Flight newFlight = new Flight(validFlightNumber);
        // verify constructed object
        assertEquals(validFlightNumber, newFlight.number);
        assertEquals("", newFlight.airlineCode);
        assertNull(newFlight.airline);
        // set up mileage
        newFlight.setMileage(1122);
        // exercise mileage translator
        int actualKilometres = newFlight.getMileageAsKm();
        // verify results
        int expectedKilometres = 1810;
        assertEquals( expectedKilometres, actualKilometres);
        // now try it with a canceled flight
        newFlight.cancel();
        try {
                newFlight.getMileageAsKm();
                fail("Expected exception");
        } catch (InvalidRequestException e) {
                assertEquals( "Cannot get cancelled flight mileage",
                e.getMessage());
        }
}
```



Comments as deodorant

Testing everything at a time

```
public void testFlightMileage_asKm2() throws Exception {
        // set up fixture
        // exercise constructor
        Flight newFlight = new Flight(validFlightNumber);
        // verify constructed object
        assertEquals(validFlightNumber, newFlight.number);
        assertEquals("", newFlight.airlineCode);
        assertNull(newFlight.airline);
        // set up mileage
        newFlight.setMileage(1122);
        // exercise mileage translator
        int actualKilometres = newFlight.getMileageAsKm();
        // verify results
        int expectedKilometres = 1810;
        assertEquals( expectedKilometres, actualKilometres);
        // now try it with a canceled flight
        newFlight.cancel();
        try {
                newFlight.getMileageAsKm();
                fail("Expected exception");
        } catch (InvalidRequestException e) {
                assertEquals( "Cannot get cancelled flight mileage",
                e.getMessage());
        }
}
```

Duplications with application logic?

Inappropriate dependencies

- Test setup depending on other tests files
- A test file depending on another test file
- Stub functions depending on other tests

extern int reg_ecx; // in the stub program

int reg_exc; // in SUT

What can we do?

Try: one test group per file

But why can't? is it because of... ?

Test initialisation hard to read and shared among test groups in the same test file

- Fixtures
- Test Data Builder
- Parameterised Creation
- make-it-easy

Dont forget fixtures

```
TEST_GROUP (TEST_thisObject)
{
   void setup()
   {
   }

   void teardown()
   {
   }
};
```

Test Data Builder

```
eth_data_buf
->setControl(2)
->withParameterA(3)
->build();
```

Parameterised Creation

```
@Before
public void setUp() throws Exception {
        alice = new Person();
        alice.setId(1L);
        alice.setFirstname("Alice");
        alice.setLastname("Adams");
        alice.setSsn("111111");
        billy = new Person();
        billy.setId(2L);
        billy.setFirstname("Billy");
        billy.setLastname("Burke");
        billy.setSsn("222222");
        clark = new Person();
        clark.setId(3L);
        clark.setFirstname("Clark");
        clark.setLastname("Cable");
        clark.setSsn("333333");
        alice.isInLoveWith(billy);
}
```

Parameterised Creation

```
public class ParameterizedCreationMethodExample {
     private Person alice, billy, clark;
     @Before
     public void setUp() throws Exception {
          clark = createPerson("Clark", "Cable");
          billy = createPerson("Billy", "Burke");
          alice = createPerson("Alice", "Adams");
          alice.isInLoveWith(billy);
     }
     private Person createPerson(String firstName, String lastName) {
          Person person = new Person();
          person.setFirstname(firstName);
          person.setLastname(lastName);
          person.setId(UniqueNumber.next());
          person.setSsn(String.valueOf(UniqueNumber.next()));
          return person;
     }
     @Test
     public void aliceShouldAcceptWhenProposedToByBilly()
          throws Exception {
          billy.proposeTo(alice);
          assertTrue(alice.isEngagedWith(billy));
     }
}
```

make-it-easy

```
Maker<Apple> appleWith2Leaves = an(Apple, with(2, leaves));
Maker<Apple> ripeApple = appleWith2Leaves.but(with(ripeness, 0.9));
Maker<Apple> unripeApple = appleWith2Leaves.but(with(ripeness, 0.125));

Apple apple1 = make(ripeApple);
Apple apple2 = make(unripeApple);

Banana defaultBanana = make(a(Banana));
Banana straightBanana = make(a(Banana, with(curve, 0.0)));
Banana squishyBanana = make(a(Banana, with(ripeness, 1.0)));
```

http://code.google.com/p/make-it-easy/

Try: One assertion per test

Customised Assertions

```
#define CHECK_OBJ(a,b) CHECK_OBJ(a,b, __FILE__,__FILE__)

void CHECK_OBJ(struct* yourObj, struct* myObj, const char* file, int line)
{
     if (structs are not equal) {
          SimpleString errorMessage = StringFromFormat(
               “My struct: %d, %p, %s”, myObj->d, myObj->p, myObj->s);
          FAIL_LOCATION(errorMessage.asCharString(), file, line);
     }
}
```

At least: One concept per test

Hamcrest

- Framework for writing declarative match criteria

```
String s = "yes we have no bananas today";

Matcher<String> containsBananas = new StringContains("bananas");
Matcher<String> containsMangoes = new StringContains("mangoes");

assertTrue(containsBananas.matches(s));
assertFalse(containsMangoes.matches(s));
```

Or even better

```
assertThat(s, containsString("bananas"));
assertThat(s, not(containsString("mangoes"));
```


http://code.google.com/p/hamcrest/

Meaningful Assertion Messages

- Don’t repeat what the built-in test framework outputs to the console (e.g. name of the test method)
- Don’t repeat what the test name explains
- If you don’t have anything good to say, you don’t have to say anything
- Write what should have happened, or what failed to happen, and possibly mention when it should have happened

It’s Design Smell!!!

Extra Constructor

```
public class LogFileMerge {
    private URL logFileA, logFileB;

    public LogFileMerge() {
        this(new URL("http://server1/system.log"),
        new URL("http://server2/system.log"));
    }

    LogFileMerge(URL a, URL b) {
        this.logFileA = a;
        this.logFileB = b;
    }
}
```

Test-Specific SubClass

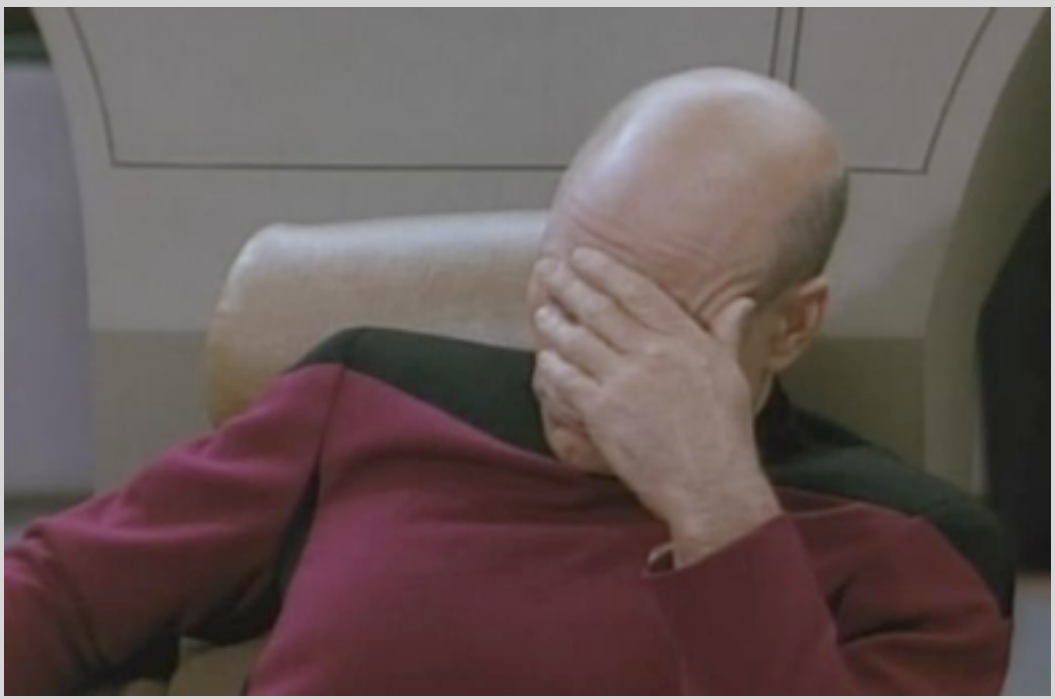
```
public class CreditCardProcessing {

    public boolean isValid(String cardnumber) {
        return validationCodeMatches(cardnumber)
            && cardIsActive(cardnumber);
    }

    protected boolean validationCodeMatches(String cardnumber) {
        // validation logic omitted for brevity...
    }

    protected boolean cardIsActive(String cardnumber) {
        // access to merchant system's web service
        // omitted for brevity...
    }
}
```

Still not testable?



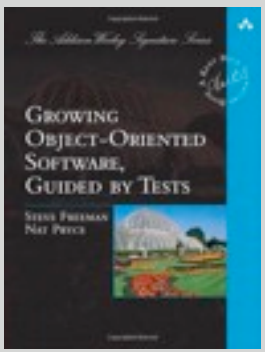
- Do you follow good design principles?

Thinking

- Test code is not second class citizen
- Good design principles apply:
  - Responsibility
  - Dependency
  - Low Coupling
  - High Cohesion
  - Indirection
  - Protected Variations
- Watch out for organisational dysfunction!

References

- Practical TDD and ATDD for Java Developers - Lasse Koskela
- Growing OO Software, guided by tests - Steve Freeman
- xUnit Test Patterns - Gerard Meszaros



Steven Mak
steven@odd-e.com
www.odd-e.com
twitter: stevenmak