Refactoring Fundamentals

Code Smells: Environment and Test Smells

Steve Smith
Ardalis.com
@ardalis

In This Course

- ~~What is Refactoring?~~
- ~~Why do it?~~
- ~~What's the process?~~
- ~~What are some tools that can assist with it?~~
- ~~What is a *Code Smell*?~~
- What are some examples of Code Smells?
- What are some common refactorings?
- How does one apply them correctly?

Organizing Code Smells

- **Taxonomy proposed by Mäntylä, M. V. and Lassenius, C.**
  - http://www.soberit.hut.fi/~mmantyla/ESE_2006.pdf

- **Organization of Code Smells into 5 Groups**
  - ~~The Bloaters~~
  - ~~The Object-Orientation Abusers~~
  - ~~The Change Preventers~~
  - ~~The Dispensables~~
  - ~~The Couplers~~

- **I've added three more:**
  - ~~The Obfuscators~~
  - Environment Smells
  - Test Smells

Code Smells: Environment

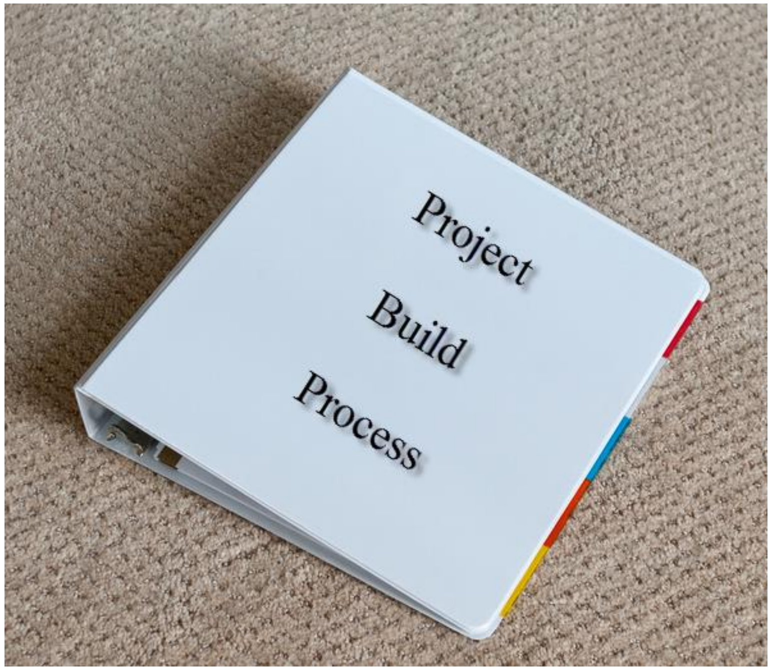
- Smells in your programming process
- Increase friction
- Reduce velocity

http://flic.kr/p/sQpCa

Environment Smells: Build Requires Multiple Steps

- **Building your project should be trivial**
  - Pull from source control
  - Run build script

- **If it takes more steps than this, consider streamlining it**



Corrective Action

- **Build Script**
- **Continuous Integration**

http://flic.kr/p/br93pH

Environment Smells: Tests Requires Multiple Steps

- Running tests should be trivial and obvious
- Make running tests the default
- Combine with build script

Corrective Action
- Build Script
- Continuous Integration

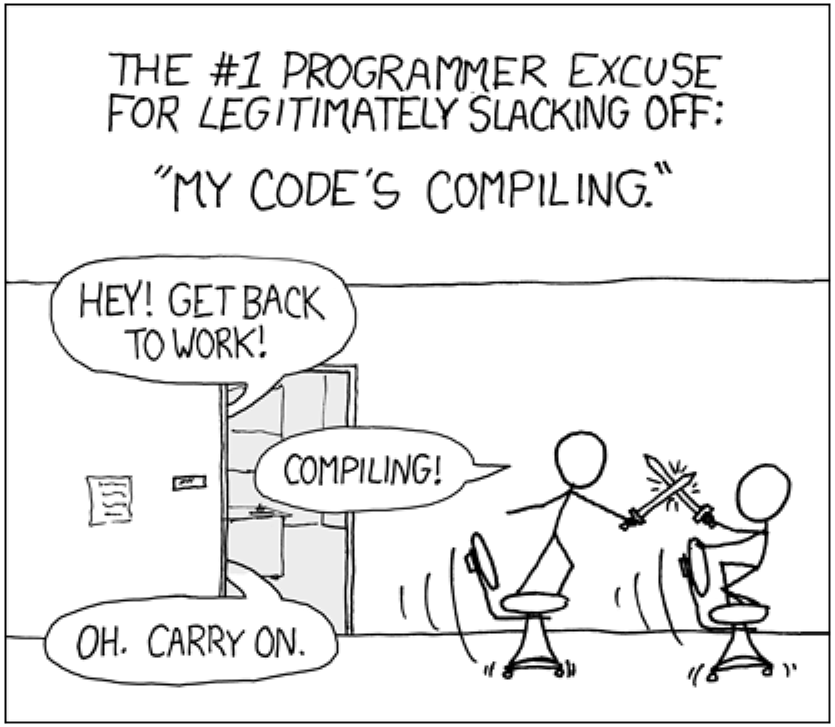
http://flic.kr/p/br93pH

Environment Smells: Builds Take Too Long

- Builds (and tests) should be FAST
- Use modern hardware
- Optimize
- Favor small, fast unit tests

Corrective Action

- Upgrade hardware
- Shift Integration Tests to Unit Tests
- Shift Slow Tests to Build Server



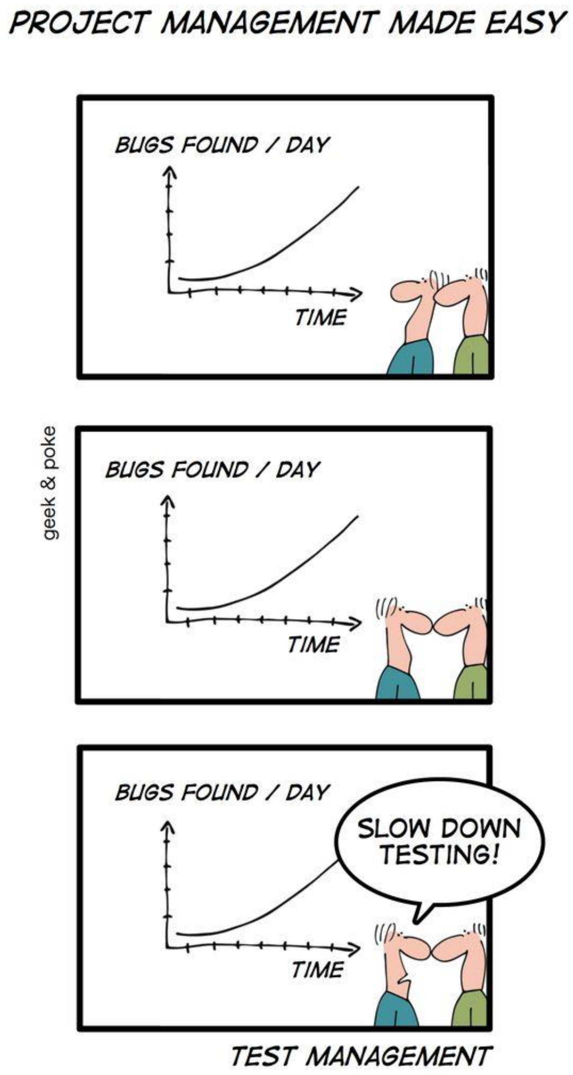
http://xkcd.com/303/

Test Smells

- Poor tests hurt productivity

Kinds of poor tests:

- Slow
- Brittle
- Overly-coupled
- Unhelpful when failing
- Inconsistent



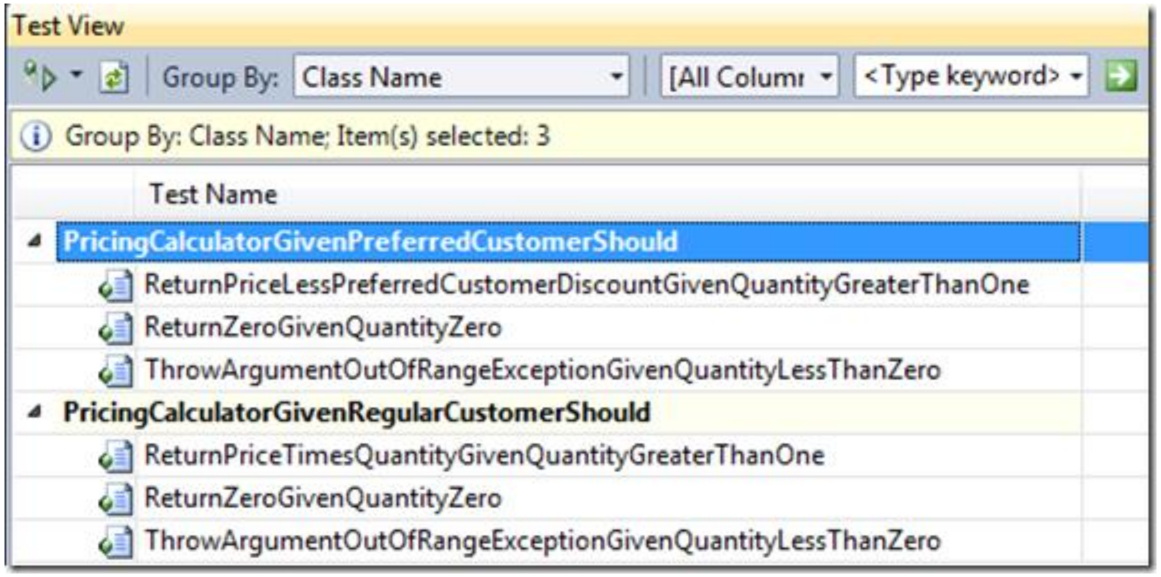
http://geek-and-poke.com

Test Smells : Not Enough Tests

- Test everything that can break
- Use a coverage tool to identify gaps
- Use Cyclomatic Complexity tools to areas that need more testing
- Write tests to document how the API should work
- Test Boundary Conditions
- Test both success and failure paths
- Test bugs

Test Smells : DRY vs. DAMP

- Don’t Repeat Yourself (DRY)
- Descriptive And Meaningful Phrases (DAMP)
- You can have good test names and still avoid duplication


Test View

Group By: Class Name [All Columı <Type keyword>

Group By: Class Name; Item(s) selected: 3

Test Name
PricingCalculatorGivenPreferredCustomerShould
ReturnPriceLessPreferredCustomerDiscountGivenQuantityGreaterThanOne
ReturnZeroGivenQuantityZero
ThrowArgumentOutOfRangeExceptionGivenQuantityLessThanZero
PricingCalculatorGivenRegularCustomerShould
ReturnPriceTimesQuantityGivenQuantityGreaterThanOne
ReturnZeroGivenQuantityZero
ThrowArgumentOutOfRangeExceptionGivenQuantityLessThanZero

http://ardalis.com/unit-test-naming-convention

Test Smells : Fragility

- Fragile or Brittle tests break too easily
- Small changes in the system break many tests
- More time is spent fixing tests than making the actual change to the system
- Tests provide negative value; slow productivity
- Gives tests a bad name
- Solution: DRY your tests

Test Smells: The Liar

- Test that appears valid…
- But doesn't really test the target at all

Examples

- Test only tests mocks or fakes, not system under test
- Integration test doesn't test infrastructure
- Errors are ignored

Test Smells: Excessive Setup

- Too hard to set up

- Setup “noise” drowns out tested behavior

To correct

- Reduce coupling in system under test
- Move test setup to helper methods

Test Smells: The Giant

- **Long, complex, and tests many things at once.**
- **Similar to the God Object in production code**
  - Also violates Long Method smell
- **May indicate that the system under test *is* a God Object**

To correct

- **Break up the responsibilities of the system under test**
- **Break up the test into separate tests**

Test Smells: The Mockery

- Mocking can be a useful tool for testing system behavior
- Confusion with multiple mock objects may result in only mocked behavior being verified
- System under test isn’t tested at all
- Special case of The Liar

To correct

- Verify the test can fail and that the system under test is being exercised correctly as part of the test

Test Smells: The Inspector

- **Usually an attempt to achieve 100% code coverage**
- **Violates encapsulation**
- **Breaks with any refactoring of the object**
- **Exacerbated by duplication**
  - Usually can be ignored unless there is a lot of it spread across tests

Test Smells: Generous Leftovers

- One test leaves some state for another

- Results in temporal coupling

To correct

- Ensure tests are completely self-contained

Test Smells: Poisonous Leftovers

- One test leaves some state that derails another test

- Results in temporal coupling

To correct

- Ensure tests are completely self-contained

Test Smells: The Local Hero

- The test depends on something specific to the local development environment

- It passes on one machine, but fails anywhere else

- Results in environment coupling

To correct

- Use a build server
- Ensure all dependencies and setup scripts are in source control

Test Smells: The Nitpicker

- A test that compares more state than necessary
- For instance, instead of checking one field, checks entire output of a web page
- Any minor change will break the test

To correct

- Isolate test assertions to those the test cares about

Test Smells: The Secret Catcher

- A test that appears to do nothing…

- But is actually relying on an exception to be thrown

To correct

- Be explicit about expected exceptions
- Be sure to explicitly fail if an expected exception does not occur

Test Smells: The Dodger

- A test dodges its primary responsibility
- Sets up and tests side effects, but never the core behavior

To correct

- Write simpler tests whenever possible
- Refactor setup code to helper methods

Test Smells: The Loudmouth

- A test that generates too much output, even when passing
- Fills up console, log files, event logs, etc. with unnecessary chatter
- Usually left over from debugging the test when it was authored

To correct

- Remove unnecessary messages
- Consider showing them only when the test fails

Test Smells: The Greedy Catcher

- Test that catches exceptions and “swallows” important details
- May replace raw exception with less informative one
- May ignore exception but log the details (a la The Loudmouth)

To correct

- Handle exceptions consistently
- Make sure failing tests contain all data that’s useful for debugging

Test Smells: The Sequencer

- Test that depends on order of unordered list(s) to pass

To correct

- Ignore order *or*
- Ensure order by explicitly sorting in the test

Test Smells: The Hidden Dependency

- **Related to Local Hero; test depends on some data being populated somewhere**
- **If the data isn't set, the test fails, leaving little clue why**

To correct

- **Ensure all dependencies are in source control and local**
- **Make non-local dependencies very explicit**
  - Write tests specifically to verify the remote dependency is working

Test Smells: The Enumerator

- **A test name smell**

- **Refers to tests whose names are simply:**
  - Test1
  - Test2
  - Test3

- **Provide no value or insight into the test's intent or expected behavior**

To correct

- **Use descriptive, intention-revealing test names**

Test Smells: The Stranger

- **Also known as: The Distant Relative**

- **Tests an object that isn't even part of the test case**

- **Most likely, the object tested is a collaborator, included by mistake**
  - May indicate Excessive Setup

To correct

- **Move the test to a fixture focusing on the object being tested**
- **Simplify the test and its setup**

Test Smells: The Operating System Evangelist

- Similar to Local Hero

- Test depends on the specific operating system it runs on to pass

- For example, a test that requires Windows-style newline sequences in its assertions (which then fail on Unix-based systems)

To correct

- Avoid depending on the local environment;
- In the case of NewLine, use a framework utility like Environment.NewLine()

Test Smells: Success Against All Odds

- **A test that cannot fail**

- **Typically written pass-first rather than fail-first**

- **Remember when writing tests:**
  - Red
  - Green
  - Refactor

To correct

- **Verify the test is exercising the system under test**
- **Change the system under test to make the test fail**
- **Fix the failing test (and refactor if necessary)**

Test Smells: The Free Ride

- An extra test, tagging along inside another one

- Over time, can result in The Giant

To correct

- Keep tests small and focused on testing one thing
- Break the extra test out into its own test case

Test Smells: The One

- One test to rule them all…
- There can be only one…
- Combines aspects of The Giant and the Free Ride
- A test fixture, with one method, that tests the entire set of functionality of a given object

To correct

- Use small, well-factored tests to test each unit of behavior

Test Smells: The Peeping Tom

- Another shared state smell
- Also known as The Uninvited Guests
- Refers to tests that can “see” results of other tests
- May fail even though the system under test is correct

To correct

- Avoid depending on shared or global state whenever possible
- Ensure any such state is returned to a known state after every test

Test Smells: The Slow Poke

- **An extremely slow test**

- **As a result, an infrequently run test**

- **Time to run the test = break time**
  - Grab some coffee
  - Chat with coworkers
  - Head home for the day

- **Slow down productivity**

To correct

- **Speed up the test**
- **Move slow tests to the continuous integration server**

Test Smells: The Contradiction

- A test whose message contradicts reality
- Typically the message states what should have happened, not what did happen

```
Assert.AreEqual(4, result, “2 +2 equals 4”);
```

- Cause confusion when analyzing test results

To correct

- State the failure that occurred.

```
Assert.AreEqual(4, result, “Expected 2+2=4, got ” + result);
```

Test Smells: Roll the Dice

- **A test that uses random test data, and only fails some of the time**

- **Tests should be repeatable and consistent**
  - Avoid random test data

- **Random systems under tests should have random values set by tests**

To correct

- **Replace random values in tests with specific values**
- **Inject “random” values into system under test**

Test Smells: Hidden Tests

- **Avoid hiding test logic in base classes**

- **Favor clarity over absolute DRYness**
  - Keep as much setup / teardown logic in the test fixture class as possible
  - Make all Assertions made by a test explicit

- **Avoid hiding asserts in teardown or cleanup methods, especially in base classes**

To correct

- **Keep setup and assertion logic in the test class**

- **Read more:**
- **http://www.ademiller.com/blogs/tech/2007/11/tdd-anti-pattern-inherited-test/**

Test Smells: Second Class Citizens

- **Test code isn't treated with the respect of production code**

- **Tests may not:**
  - Be in source control
  - Follow coding standards and conventions
  - Be refactored to keep quality high

- **This leads to poor quality tests that either hurt productivity or are abandoned**

To correct

- **Treat test code like production code**

Test Smells: Wait and See

- **A test that explicitly waits for an action to occur**
- **Frequently contribute to very slow test suites**
- **Avoid whenever possible**

To correct

- **Eliminate delays in system under test whenever possible**
  - *E.g. parameterize or inject Thread.Sleep()s that are needed in production but not during tests*
- **Use callbacks rather than polling**

Test Smells: Inappropriate Test Group

- **Avoid grouping too many tests per fixture**

- **Tests that use none of the fixture's setup and teardown logic probably belong somewhere else**
    - Especially if the setup time is non-trivial

- **Avoid having The Giant or Long Class smells in your test fixtures**

To correct

- **Use as many test fixtures as necessary to cleanly organize tests by responsibility and feature being tested**

Test Smells: The Optimist

- Also known as The Happy Path

- Only “happy paths” are tested – no “sad paths”

To correct

- Ensure tests cover success, failure, and boundary conditions of the system under test

Test Smells: The Sleeper

- A test that is guaranteed to fail after a certain point in the future
- Often caused by hard-coding a specific date in the test
- May also occur at a very specific time of day (e.g. midnight) each day

To correct

- Don’t rely on system clock in production code
- Use only relative dates/times in test code when possible

Quote

This code will be replaced long before the year 2000.

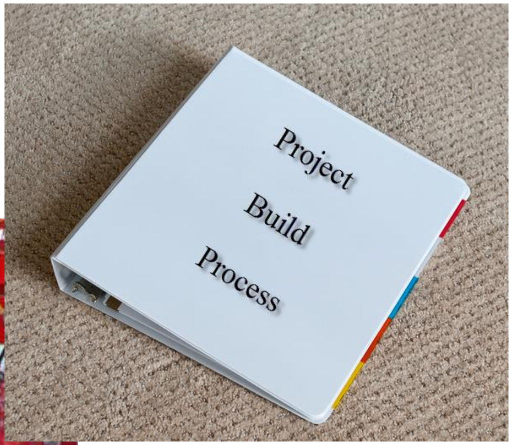
1970s programmers

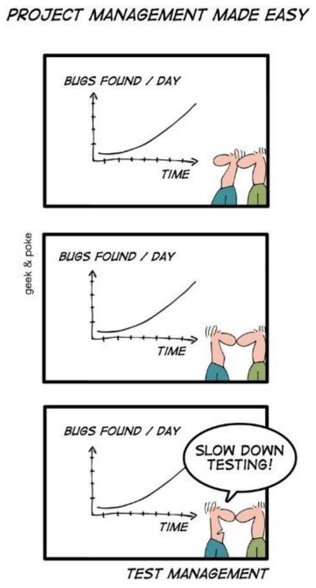
Test Smells: The Void

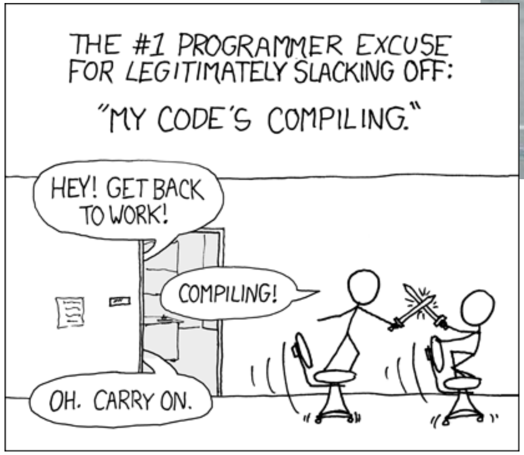
No Tests At All

Summary

Project
Unit Test Running
Process


Project
Build
Process


PROJECT MANAGEMENT MADE EASY
BUGS FOUND / DAY
TIME
BUGS FOUND / DAY
TIME
BUGS FOUND / DAY
SLOW DOWN TESTING!
TIME
TEST MANAGEMENT
geek & poke


THE #1 PROGRAMMER EXCUSE FOR LEGITIMATELY SLACKING OFF:
"MY CODE'S COMPILING."
HEY! GET BACK TO WORK!
COMPILING!
OH. CARRY ON.

References

Related Pluralsight Courses

SOLID Principles of Object Oriented Design http://bit.ly/rKbR9a

Books

Clean Code http://amzn.to/YjUDI0

Web

- TDD AntiPatterns http://blog.james-carr.org/2006/11/03/tdd-anti-patterns/
- On Stackoverflow http://stackoverflow.com/questions/333682/tdd-anti-patterns-catalogue
- Unit Test Naming Convention
  http://ardalis.com/unit-test-naming-convention

Thanks!

Steve Smith

Ardalis.com

Twitter: @ardalis

To Teach Is To Learn Twice

pluralsight
hardcore developer training