Automated Detection of Test Fixture Strategies and Smells

Michaela Greiler, Arie van Deursen, Margaret-Anne Storey
xUnit tools offer up huge opportunities to programming teams

- powerful regression test lead to less risk;

- re-think the design process with Test-Driven Development.

(Martin Fowler)
“But with these opportunities come new problems. Like any tool, the xUnit family can be used well or badly.

Thoughtful people have figured out various ways to use xUnit, to organize the tests and data effectively.”

(Martin Fowler)
Keep Code Clean and Simple!

Test code is just as important as the production code.
Structure of a Test
Test Structure

Test (method)

1. Statement
2. Statement
3. Statement
4. Statement
5. Statement
6. Statement

Setup of Test Fixture
Test Fixture

The code that initializes and configures the system under test.
Test Structure

Test (method)

1. Statement
2. Statement
3. Statement
4. Statement
5. Statement
6. Statement

Setup of Test Fixture
Exercise SUT
Verify Outcome
Teardown Test Fixture
public void testGetFlightsByOriginAirport_NoFlights_inline() throws Exception {
    // Fixture setup
    NonTxFlightMngtFacade facade =\new NonTxFlightMngtFacade();
    BigDecimal airportId = facade.createTestAirport("10F");
    try {
        // Exercise System
        List flightsAtDest = facade.getFlightsByOriginAirport(airportId);
        // Verify Outcome
        assertEquals(0, flightsAtDest.size());
    } finally {
        // Fixture teardown
        facade.removeAirport(airportId);
    }
}
How to structure the test fixture and where to place it?
Fixture Types

• Inline (in test method)
• Delegate (use helper methods)
• Implicit (use framework)
public void testGetFlightsByOriginAirport_NoFlights_inline() throws Exception {
    // Fixture setup
    NonTxFlightMngtFacade facade = new NonTxFlightMngtFacade();
    BigDecimal airportId = facade.createTestAirport("1OF");
    try {
        // Exercise System
        List flightsAtDest = facade.getFlightsByOriginAirport(airportId);
        // Verify Outcome
        assertEquals(0, flightsAtDest.size());
    } finally {
        // Fixture teardown
        facade.removeAirport(airportId);
    }
}
private NonTxFlightMngtFacade facade = new NonTxFlightMngtFacade();
private BigDecimal airportId;

protected void createTestAirportOriginTest() {
    MngFacade facade = new NonTxMngFacade();
    airportId = facade.createTestAirport("10F");
}

public void testGetFlightsByOriginAirport_NoFlights_implicit() throws Exception {
    createTestAirportOriginTest();
    List flightsAtDest = facade.getFlightsByOriginAirport(airportId);
    // Verify Outcome
    assertEquals(0, flightsAtDest.size());
    deleteTestAirportOriginTest();
}

protected void deleteTestAirportOriginTest() {
    facade.removeAirport(airportId);
}
private NonTxFlightMngtFacade facade = new NonTxFlightMngtFacade();
private BigDecimal airportId;

protected void setUp() throws Exception {
    // Fixture setup
    super.setUp();
    airportId = facade.createTestAirport("1OF");
}

public void testGetFlightsByOriginAirport_NoFlights_implicit() throws Exception {
    // Exercise SUT
    List flightsAtDest = facade.getFlightsByOriginAirport(airportId);
    // Verify Outcome
    assertEquals(0, flightsAtDest.size());
}

protected void tearDown() throws Exception {
    // Fixture teardown
    facade.removeAirport(airportId);
    super.tearDown();
}
“Most of the complexity of writing tests involves how to write the Test Methods; what to include inline and what to factor out into Test Utility Methods, and so on.”

(Meszaros)
Fixture Types

• In-line
  – code duplication
  – obscure test

• Delegate (helper methods)
  – explicit invocation needed

• Implicit (framework)
  – invocations per test, class, suite, group
  – must fit the test groupings
Balance is the Key to Life
Choices can lead to...

Test Smells
What is a test smell?

A symptom of a problem
• Can originate from several causes
• Refactoring needed

Refactoring Test Code

Arie van Deursen  Leon Moonen  
CW1  The Netherlands  
http://www.cwi.nl/~arie.leon/  
{arie,leon}@cwi.nl

Alex van den Bergh  Gerard Kok  
Software Improvement Group  
The Netherlands  
http://www.software-improvers.com/  
{alex,gerard}@software-improvers.com

ABSTRACT
Two key aspects of extreme programming (XP) are unit testing and merciless refactoring. Given the fact that the ideal test code / production code ratio approaches 1:1, it is not surprising that unit tests are being refactored. We found that refactoring test code is different from refactoring production code in two ways: (1) there is a distinct set of bad smells involved, and (2) improving test code involves additional test-specific refactorings. To share our experiences with other XP practitioners, we describe a set of bad smells that indicate trouble in test code, and a collection of test refactorings to remove these smells.
Smells can lead to Problems

- Slow tests
- Maintenance overhead
- Fragile tests
- Obscure tests
- Unclear cause-effect relations
How to measure fixture smells?

Usage of class fields and variable declarations!
class BlobStorageExampleTest extends GitTestCase {

    // setup field
    Repository repository;
    Repository repository2;
    // header initialization
    Storage store = new Storage();
    // ad hoc field
    Directory dir, gitDir;

    @Before public void setUp() throws Exception {
        super.setUp();
        gitDir = new Directory(".");
        // repository depends on gitDir
        repository = new FileRepository(gitDir);
        repository2 = new FileRepository(gitDir);
    }

    ...

    @Test public void testRepository() {
        dir = new Directory(".");
        loadFile();
        ...
    }

    private void loadFile(){
        repository.getFile("testfile");
        ...
    }
}
```java
class BlobStorageExampleTest extends GitTestCase {
    // setup field
    Repository repository;
    Repository repository2;
    // header initialization
    Storage store = new Storage();
    // ad hoc field
    Directory dir, gitDir;

    @Before public void setUp() throws Exception {
        super.setUp();
        gitDir = new Directory(".");
        // repository depends on gitDir
        repository = new FileRepositoryRepository(gitDir);
        repository2 = new FileRepositoryRepository(gitDir);
    }

    ... 

    @Test public void testRepository() {
        dir = new Directory(".");
        loadFile();
        ...
    }

    private void loadFile(){
        repository.getFile("testfile");
        ...
    }
}
```

Field declaration
Field initialization
Field usage (via helper)
class BlobStorageExampleTest extends GitTestCase {
    //setup field
    Repository repository;
    Repository repository2;
    //header initialization
    Storage store = new Storage();
    //ad hoc field
    Directory dir, gitDir;

    @Before public void setUp() throws Exception {
        super.setUp();
        gitDir = new Directory(".");
        //repository depends on gitDir
        repository = new FileRepositoryRepository(gitDir);
        repository2 = new FileRepositoryRepository(gitDir);
    }

    ... @Test public void testRepository() {
        dir = new Directory(".");
        loadFile();
        ...
    }

    private void loadFile(){
        repository.getFile("testfile");
        ...
    }

    Michaela Greiler
Test Fixture Smells

• General Fixture
• Test Maverick
• Dead Fields
• Lack of Cohesion of Test Methods
• Vague Header
• Obscure In-line setup
General Fixture...

...occurs if test classes contain broad functionality in the implicit setup, and different tests only access part of the fixture.

\[
\text{When you have a new hammer, everything looks like a nail!}
\]
Test Mavericks...

...are test methods that make no use of the implicit setup procedure.
Lack of Cohesion of Test Methods...

...occurs if test methods within a class do not belong together.
Vague Header

...occurs when fields are initialized in the header of a class, but not in the implicit setup.
Dead Fields

...are fields that are initialized but never used within the test class.
Obscure In-line Setup

...within the test method to much setup code is place which obscures the focus of the test.

Michaela Greiler
Evaluation

Case Studies

Interviews & Questionnaire

HealthCare system
Evaluation

Case Studies

Do fixture-related test smells occur in practice?

HealthCare system

eclipse Mylyn

EGit

Interviews & Questionnaire

Do developers recognize these test smells as potential problems?

Michaela Greiler
TestHound

Detects test smells
Guides test refactoring
# Smell Summary Report

<table>
<thead>
<tr>
<th>Testcase class</th>
<th>No. Tests</th>
<th>No. Flds (Setup/ All)</th>
<th>Dead Flds</th>
<th>LCOM</th>
<th>General Fixture</th>
<th>Test Maverick</th>
<th>Obscure Inline</th>
<th>Vague Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.eclipse.egit.core.test.op.AddOperationTest</td>
<td>6</td>
<td>5/5</td>
<td></td>
<td>0.0</td>
<td></td>
<td></td>
<td></td>
<td>![Icon]</td>
</tr>
<tr>
<td>org.eclipse.egit.core.GitMoveDeleteHookTest</td>
<td>19</td>
<td>4/8</td>
<td></td>
<td>0.04</td>
<td></td>
<td>![Icon]</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>org.eclipse.egit.core.test.indexDiff.IndexDiffCacheTest</td>
<td>1</td>
<td>5/5</td>
<td>![Icon] 1 deads super: 1/3</td>
<td>0.0</td>
<td></td>
<td>![Icon]</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>org.eclipse.egit.core.test.op.RemoveFromIndexOperationTest</td>
<td>5</td>
<td>5/5</td>
<td></td>
<td>0.0</td>
<td></td>
<td>![Icon]</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>org.eclipse.egit.core.securestorage.GitSecureStoreTest</td>
<td>13</td>
<td>2/2</td>
<td></td>
<td>0.0</td>
<td></td>
<td>![Icon]</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>org.eclipse.egit.core.synchronize.ThreeWayDiffEntryTest</td>
<td>9</td>
<td>9/9</td>
<td>![Icon] 5 deads super: 5/8</td>
<td>0.0</td>
<td></td>
<td>![Icon]</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>org.eclipse.egit.core.test.GitProjectSetCapabilityTest</td>
<td>4</td>
<td>3/3</td>
<td></td>
<td>0.0</td>
<td></td>
<td>![Icon]</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>org.eclipse.egit.core.test.op.CommitOperationTest</td>
<td>7</td>
<td>7/7</td>
<td></td>
<td>0.12</td>
<td></td>
<td>![Icon]</td>
<td>![Icon]</td>
<td></td>
</tr>
<tr>
<td>org.eclipse.egit.core.internal.storage.BlobStorageTest</td>
<td>6</td>
<td>4/4</td>
<td></td>
<td>0.5</td>
<td>![Icon] 2 methods</td>
<td>![Icon] 1 detached</td>
<td>![Icon] 1 methods</td>
<td>![Icon]</td>
</tr>
</tbody>
</table>
# Detail Report

## General Fixture

2 test methods use less than 70% of the fields set during test setup. Methods affected: `testFailNotFound`, `testFailWrongType`.

## Minimal Fixture

After removing detached methods from class, refactor the large fixture to a minimal fixture. Minimal Fixture: `repository gitDir`

Hint: Only half of the fields are shared. Consider an Extract Class refactoring.

Extract Class 1: `testOk`
Extract Class 2: `testFailNotFound testFailWrongType testFailCorrupt testFailCorrupt2`

## Setup Detail 4 member fields, 0 static fields

<table>
<thead>
<tr>
<th>Fields set in Setup</th>
<th>Usage</th>
<th>Inherited</th>
</tr>
</thead>
<tbody>
<tr>
<td>repository</td>
<td><img src="https://example.com/green.png" alt="Green" /> 5 out of 6 test methods use this field</td>
<td>GitTestCase</td>
</tr>
<tr>
<td>project</td>
<td><img src="https://example.com/orange.png" alt="Orange" /> 3 out of 6 test methods use this field</td>
<td>GitTestCase</td>
</tr>
<tr>
<td>gitDir</td>
<td><img src="https://example.com/green.png" alt="Green" /> 5 out of 6 test methods use this field</td>
<td>GitTestCase</td>
</tr>
<tr>
<td>testUtils</td>
<td><img src="https://example.com/red.png" alt="Red" /> 1 out of 6 test methods use this field</td>
<td>GitTestCase</td>
</tr>
</tbody>
</table>
Case Studies

130K LOC
87 test classes
479 test methods

750K LOC
36 test classes
933 test methods

500K LOC
232 test classes
1644 test methods
Fixture Organisation

95% @Setup
75% @Setup
70% @Setup

the package structure of the test code closely followed the package structure of the SUT

often class to test class mapping
Do fixture-related test smells occur in practice?

- General Fixture
- Test Mavericks
- LCOTM
- Dead Fields
- Obscure Inline
- Vague Header

Legend:
- Health Care
- Mylyn
- eGit
Do developers recognize these test smells as potential problems?
Wrong test fixture management can lead to code quality problems. Improving test code quality is important. We regularly engage in maintenance tasks of test code. The report will primarily show irrelevant information. Using a fixture management tool could help me improve the test code quality. I expect to look at the reports of this tool on a regular basis.

- Strongly disagree
- Disagree
- Neither agree nor disagree
- Agree
- Strongly agree
“I really like the tool. I think it presents a lot of useful information. I think it can definitely be very beneficial for our company.”

“The report was definitely very useful. It triggered a lot of ideas to improve and discussions.”
“We do not have the option to say ’Oh, that’s ugly, I’ll spend a day to clean it up’ if it does not give us immediate business.”

“It’s not enough to have the tool run. It should be part of the infrastructure and result in a failed build.”
“It is all about low-hanging fruit: what can you do easily and quickly.”

Strategies for Avoiding Test Fixture Smells During Software Evolution

Michaela Greiler, Andy Zaidman and Arie van Deursen
Delft University of Technology, The Netherlands
{m.s.greiler,a.e.zaidman,arie.vandeursen}@tudelft.nl

Margaret-Anne Storey
University of Victoria, Canada
mstorey@uvic.ca

Abstract—An important challenge in creating automated tests is how to design test fixtures, i.e., the setup code that initializes the system under test before actual automated testing can start. Test designers have to choose between different approaches for the setup, trading off maintenance overhead with slow test execution. Over time, test code quality can erode and test smells can develop, such as the occurrence of overly complex code and dead fields. In this fixture-related test smells evolve in thousand revisions of open source systems, we previously developed a tool that helps developers identify and refactor code when test smells are present.

MSR 2013
Lessons learned

• Grouping: class to test class.
• Frequency of Vague Headers.
• Violation of the Single Responsibility Principle.
• Inheritance Structure.
More information? Interested?

Michaela Greiler
m.s.greiler@tudelft.nl
@mgreiler