On the Diffusion of Test Smells in Automatically Generated Test Code: An Empirical Study

Fabio Palomba*, Dario Di Nucci*, Annibale Panichella+, Rocco Oliveto°, Andrea De Lucia*

SBST 2016
May, 16th 2016
Austin, TX, USA

* University of Salerno, Italy
+ Delft University of Technology, The Netherlands
° University of Molise, Italy
Automatic Generation of Test Code

Production Class

- method1()
- method2()
- ...
- methodN()

Test Suite

- test_method1()
- test_method2()
- ...
- test_methodN()
Automatic Generation of Test Code

Usability of Automatic Generation Tools in Practice

[Rojas et al. - ISSTA'15]
What about the characteristics of test code produced by such tools?
Test Smells in Test Code

Refactoring Test Code

Arie van Deursen  
Leen Moonen  
CWI  
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.

Keywords
Refactoring, unit testing, extreme programming.

1 INTRODUCTION
"If there is a technique at the heart of extreme programming (XP), it is unit testing" [1]. As part of their programming activity, XP developers write and maintain (white box) unit tests continually. These tests are automated, written in the same programming language as the production code, considered an explicit part of the code, and put under revision control.

The XP process encourages writing a test class for every class in the system. Methods in these test classes are used to verify complicated functionality and unusual circumstances. Moreover, they are used to document code by explicitly indicating what the expected results of a method should be for typical cases. Last but not least, tests are added upon receiving a bug report to check for the bug and to check the bug fix [2]. A typical test for a particular method includes: (1) code to set up the fixture (the data used for testing), (2) the call of the method, (3) a comparison of the actual results with the expected values, and (4) code to tear down the fixture. Writing tests is usually supported by frameworks such as JUnit [3].

The test code / production code ratio may vary from project to project, but is ideally considered to approach a ratio of 1:1. In our project we currently have a 2.5 ratio, although others have reported a lower ratio.1 One of the cornerstones of XP is that having many tests available helps the developers to overcome their fear for change: the tests will provide immediate feedback if the system gets broken at a critical place. The downside of having many tests, however, is that changes in functionality will typically involve changes in the test code as well. The more test code we get, the more important it becomes that this test code is as easily modifiable as the production code.

The key XP practice to keep code flexible is "refactor mercilessly": transforming the code in order to bring it in the simplest possible state. To support this, a catalog of "code smells" and a wide range of refactorings is available, varying from simple modifications up to ways to introduce design patterns systematically in existing code [5].

When trying to apply refactorings to the test code of our project we discovered that refactoring test code is different from refactoring production code. Test code has a distinct set of smells, dealing with the ways in which test cases are organized, how they are implemented, and how they interact with each other. Moreover, improving test code involves a mixture of refactorings from [5] specialized to test code improvements, as well as a set of additional refactorings, involving the modification of test classes, ways of grouping test cases, and so on.

The goal of this paper is to share our experience in improving our test code with other XP practitioners. To that end, we describe a set of test smells indicating trouble in test code, and a collection of test refactorings explaining how to overcome some of these problems through a simple program modification.

This paper assumes some familiarity with the JUnit framework [3] and refactorings as described by Fowler [5]. We will refer to refactorings described in this book using Nure

1 This project started a year ago and involves the development of a product called Decline [4]. Development is done by a small team of five people using XP techniques. Code is written in Java and we use the JUnit framework for unit testing.

Test Smells represent a set of a poor design solutions to write tests

[Van Deursen et al. - XP 2001]
public void test12 () throws Throwable {
    JSTerm jSTerm0 = new JSTerm();
    jSTerm0.makeVariable () ;
    jSTerm0.add((Object) "");
    jSTerm0.matches(jSTerm0);
    assertEquals (false, jSTerm0.isGround () );
    assertEquals (true, jSTerm0.isVariable());
}
Test Smells in Test Code

```java
public void test12 () throws Throwable {
    JSTerm jSTerm0 = new JSTerm();
    jSTerm0.makeVariable();
    jSTerm0.add((Object) "");
    jSTerm0.matches(jSTerm0);
    assertEquals(false, jSTerm0.isGround());
    assertEquals(true, jSTerm0.isVariable());
}
```

The test method checks the production method isGround()
public void test12 () throws Throwable {
    JSTerm jSTerm0 = new JSTerm();
    jSTerm0.makeVariable ();
    jSTerm0.add((Object) "");
    jSTerm0.matches(jSTerm0);
    assertEquals (false, jSTerm0.isGround ());
    assertEquals (true, jSTerm0.isVariable());
}

But also the production method isVariable()
public void test12 () throws Throwable {
    JSTerm jSTerm0 = new JSTerm();
    jSTerm0.makeVariable ();
    jSTerm0.add((Object) "");
    jSTerm0.matches(jSTerm0);
    assertEquals (false, jSTerm0.isGround ());
    assertEquals(true, jSTerm0.isVariable());
}

This is an Eager Test, namely a test which checks more than one method of the class to be tested, making difficult the comprehension of the actual test target.
Test Smells in Test Code

A test case is affected by a Resource Optimism when it makes assumptions about the state or the existence of external resources, providing a non-deterministic result that depend on the state of the resources.

An Assertion Roulette comes from having a number of assertions in a test method that have no explanation. If an assertion fails, the identification of the assert that failed can be difficult.
Who cares about Test Smells?

Test Cases can be re-generated!
Who cares about Test Smells?
Test Cases can be re-generated!

True
Who cares about Test Smells?

Test Cases can be re-generated!

True

BUT
Who cares about Test Smells?
Test Cases can be re-generated!

True
BUT

Developers modify and remove test code
Developers add tests when automatic tools leave uncovered branches
Developers combine generated with manually written tests

Usability of Automatic Generation Tools in Practice
[Rojas et al. - ISSTA’15]
On the Diffusion of Test Smells in Automatically Generated Test Code: An Empirical Study

Empirical Study Design
Empirical Study Design

8 test smell types

“Refactoring Test Code”
[Van Deursen et al. - XP 2001]
Empirical Study Design

8 test smell types

“A Refactoring Test Code”
[Van Deursen et al. - XP 2001]

110 software projects

“A Large Scale Evaluation of Automated Unit Test Generation using Evosuite”
[Fraser and Arcuri - TOSEM 2014]
Empirical Study Design

8 test smell types
“Refactoring Test Code”
[Van Deursen et al. - XP 2001]

110 software projects
“A Large Scale Evaluation of Automated Unit Test Generation using Evosuite”
[Fraser and Arcuri - TOSEM 2014]
Empirical Study Design

16,603

JUnit classes

“A Large Scale Evaluation of Automated Unit Test Generation using Evosuite”

[Fraser and Arcuri - TOSEM 2014]
Data Extraction

Test Suite

test_method1()
test_method2()
...
test_methodN()

Test Smell Detector

"Are Test Smells Harmful? An Empirical Study"
[Bavota et al. - EMSE 2015]
Data Extraction

Test Suite

test_method1()
test_method2()
...
test_methodN()

Test Smell Detector

“Are Test Smells Harmful? An Empirical Study”
[Bavota et al. - EMSE 2015]

75% precision 100% recall

Sample size: 378 JUnit classes
RQ1: To What Extent Test Smells are Spread in Automatically Generated Test Classes?
Research Questions

RQ1: To What Extent Test Smells are Spread in Automatically Generated Test Classes?

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?
Research Questions

RQ1: To What Extent Test Smells are Spread in Automatically Generated Test Classes?

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

RQ3: Which Test Smells Co-Occur Together?
Research Questions

**RQ1:** To What Extent Test Smells are Spread in Automatically Generated Test Classes?

**RQ2:** Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

**RQ3:** Which Test Smells Co-Occur Together?

**RQ4:** Is There a Relationship Between the Presence of Test Smells and the Project Characteristics?
On the Diffusion of Test Smells in Automatically Generated Test Code: An Empirical Study

Analysis of the Results
Results of the Study

RQ1: To What Extent Test Smells are Spread in Automatically Generated Test Classes?

13,791 smelly JUnit classes
Results of the Study

RQ1: To What Extent Test Smells are Spread in Automatically Generated Test Classes?

83% of the JUnit classes analyzed
Results of the Study

RQ1: To What Extent Test Smells are Spread in Automatically Generated Test Classes?

RQ1
Test Smells are highly diffused in the automatically generated test suites
Results of the Study

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

Assertion Roulette 54%
RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

Results of the Study

- **Assertion Roulette**: 54%
- **Test Code Duplication**: 33%
Results of the Study

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

- Assertion Roulette: 54%
- Test Code Duplication: 33%
- Eager Test: 29%
Results of the Study

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

```java
public void test8 () throws Throwable {
    Document document0 = new Document();
    assertNotNull(document0);
    document0.procText.add((Character) "s");
    String string0 = document0.stringify();
    assertEquals("s", document0.stringify());
    assertNotNull(string0);
    assertEquals("s", string0);
}
```
Results of the Study

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

Assertion Roulette

What is the behavior under test?

Are the generated assertions valid?
Results of the Study

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

These problems have a huge impact on developers’ ability to find faults

The Impact of Test Case Summaries on Bug Fixing Performance
An Empirical Investigation
[Panichella et al. - ICSE’16]
RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

Test Code Duplication

```java
public void test8 () throws Throwable {
    GenericProperties generic0 = new GenericProperties();
    boolean boolean0 = generic0.isValidClassname();
    ...
}
```

```java
public void test9 () throws Throwable {
    GenericProperties generic0 = new GenericProperties();
    boolean boolean0 = generic0.isValidClassname();
    ...
}
```
Results of the Study

RQ2: Which Test Smells Occur More Frequently in Automatically Generated Test Classes?

Test Code Duplication

This problem can be avoided by generating test fixtures!
Results of the Study

RQ3: Which Test Smells Co-Occur Together?

- Assertion Roulette
- Eager Test
- Assertion Roulette
- Sensitive Equality
- Resource Optimism
- Mystery Guest
Results of the Study

RQ3: Which Test Smells Co-Occur Together?

- Assertion Roulette
- Eager Test
- Assertion Roulette
- Sensitive Equality
- Resource Optimism
- Mystery Guest

Automatic tools have as main goal that of maximize coverage, without considering test code quality.
Results of the Study

RQ4: Is There a Relationship Between the Presence of Test Smells and the Project Characteristics?

Yes! The higher the LOC to be tested, the higher the probability to produce a smelly test case!
Results of the Study

RQ4: Is There a Relationship Between the Presence of Test Smells and the Project Characteristics?

Yes! The higher the LOC to be tested, the higher the probability to produce a smelly test case!

The higher the LOC of the JUnit class, the higher the probability to produce a smelly test case!
Lesson 1: Current implementations of search-based algorithms for automatic test case generation do not consider code quality, increasing the probability to introduce smells!
Lesson 1: Current implementations of search-based algorithms for automatic test case generation do not consider code quality, increasing the probability to introduce smells!

NB: Considering test code quality is important not only to avoid the introduction of smells, but also because the coverage can be increased!

Summarizing

**Lesson 2:** Automatic test case generation tools do not produce text fixtures during their computation, and this implies the introduction of several code clones in the resulting JUnit classes.

Future research should spend effort in the automatic generation of test fixtures!
From now on…

Challenge 1: Evaluating Test Smells in Test Cases automatically generated by other tools
From now on...

**Challenge 1:** Evaluating Test Smells in Test Cases automatically generated by other tools

**Challenge 2:** Defining new algorithms able to solve the design problems analyzed (e.g., test fixtures).
On the Diffusion of Test Smells in Automatically Generated Test Code: An Empirical Study

Fabio Palomba*, Dario Di Nucci*, Annibale Panichella+, Rocco Oliveto°, Andrea De Lucia*

SBST 2016
May, 16th 2016
Austin, TX, USA

* University of Salerno, Italy
+ Delft University of Technology, The Netherlands
° University of Molise, Italy