Does Refactoring of Test Smells Induce Fixing Flaky Tests?

Fabio Palomba & Andy Zaidman

TU Delft

Does Refactoring of Test Smells Induce Fixing Flaky Tests?

***Abstract*—Regression testing is a core activity that allows developers to ensure that source code changes do not introduce bugs. An important prerequisite then is that test cases are deterministic. However, this is not always the case as some tests suffer from so-called *flakiness*. Flaky tests have serious consequences, as they can hide real bugs and increase software inspection costs. Existing research has focused on understanding the root causes of test flakiness and devising techniques to automatically fix flaky tests; a key area of investigation being concurrency. In this paper, we investigate the relationship between flaky tests and three previously defined test smells, namely *Resource Optimism*, *Indirect Testing* and *Test Run War*. We have set up a study involving 19,532 JUnit test methods belonging to 18 software systems. A key result of our investigation is that 54% of tests that are flaky contain a test code smell that can cause the flakiness. Moreover, we found that refactoring the test smells not only removed the design flaws, but also fixed all 54% of flaky tests causally co-occurring with test smells.**

***Index Terms*—Test Smells; Flaky Tests; Refactoring;**

I. Introduction

Test cases form the first line of defense against the introduction of software faults, especially when testing for regression faults [1], [2]. As such, with the help of testing frameworks like, for example, JUnit developers create test methods and run these periodically on their code [1], [3]. The entire team relies on the results from these tests to decide on whether to merge a pull request [4] or to deploy the system [5], [6], [7]. When it comes to testing, developer productivity is partly dependent on both (i) the ability of the tests to find real problems with the code being changed or developed [5], [8] and (ii) the cost of diagnosing the underlying cause in a timely and reliable fashion [9].

Unfortunately, test suites are often affected by bugs that can preclude the correct testing of software systems [10], [11]. A typical bug affecting test suites is *flakiness* [5]. Flaky tests are tests that exhibit both a passing and a failing result with the same code [5], *i.e.,* unreliable test cases whose outcome is not deterministic. The relevance of the flaky test problem is well-known and highlighted by both researchers and practitioners. For instance, dozens of daily discussions are opened on the topic on social networks and blogs [12], [13], [14], [15], [5].

Some key issues that are associated with the presence of flaky tests are that: (i) they may hide real bugs and are hard to reproduce due to their non-determinism [16], (ii) they increase maintenance costs because developers may have to spend substantial time debugging failures that are not really failures, but just flaky [17]. Perhaps most importantly, from a psychological point of view flaky tests can reduce a developer's confidence in the tests, possibly leading to ignoring actual test failures [15]. Because of this, the research community has spent considerably effort on trying to understand the causes behind test flakiness [16], [18], [19], [20] and on devising automated techniques able to fix flaky tests [21], [22], [23]. However, most of this research mainly focused the attention on some specific causes possibly leading to the introduction of flaky tests, such as concurrency [24], [23], [25] or test order dependency [20] issues, thus proposing *ad-hoc* solutions that cannot be used to fix flaky tests characterized by other root causes. Indeed, according to the findings by Luo *et al.* [16] who conducted an empirical study on the motivations behind test code flakiness, the problems faced by previous research only represent a part of whole story: a deeper analysis of possible fixing strategies of other root causes (*e.g.,* flakiness due to wrong usage of external resources) is still missing.

In this paper, we aim at making a further step ahead toward the comprehension of test flakiness, by investigating the role of so-called *test smells* [26], [27], *i.e.,* poor design or implementation choices applied by programmers during the development of test cases. In particular, looking at the test smell catalog proposed by Van Deursen *et al.* [26], we noticed that the definitions of three particular design flaws, *i.e., Resource Optimism*, *Indirect Testing* and *Test Run War*, are closely connected to the concept of test flakiness. For instance, the *Resource Optimism* smell refers to a test case that makes optimistic assumptions about the state or the existence of external resources [26], thus possibly producing a non-deterministic behavior due to the presence/absence of the referenced external resource.

Therefore, in this study we hypothesize that *test smells can represent a useful source of information to locate flaky tests*. We also hypothesize that the *refactoring of test smells may, as indirect effect, mean the resolution of the test flakiness*. To investigate our hypotheses, our investigation is steered by the following research questions:

- **RQ$_1$:** *What are the causes of test flakiness?*
- **RQ$_2$:** *To what extent can flaky tests be explained by the presence of test smells?*
- **RQ$_3$:** *To what extent does refactoring of test smells help in removing test flakiness?*

To verify our hypotheses and seek answers to our research questions, we have set up an empirical study which involves the source code of 19,532 JUnit test methods belonging to 18 large software systems. Key results from our study indicate that flaky tests are quite common: almost 45% of JUnit test methods that we analyzed are flaky. Moreover, we found that

Palomba and Zaidman
ICSME 2017
Shanghai, China

IEEE/TCSE
Distinguished Paper Award

Regression Testing

Pinto et al.
“Understanding Myths and Realities of Test-Suite Evolution”
FSE 2012

Test cases form the first line of defense against the introduction of faults

Regression Testing

Gousios et al.
"Work practices and challenges in pull-based development: The integrator's perspective"
ICSE 2015

The entire team rely on them to decide whether to merge a pull request

Regression Testing

Beller et al.
"Ooops, my tests broke the build: An explorative analysis of Travis CI with GitHub*"* - MSR 2017

Or even whether to proceed with the deployment of the system

Flaky Tests

Test cases that exhibit both
a passing and failing result
with the same code

Luo et al.
“An Empirical Analysis of Flaky Tests”
FSE 2014

Flaky Tests

Test cases that exhibit both a passing and failing result with the same code

They might hide real bugs, increase maintenance costs, and reduce developers' confidence

Luo et al.
"An Empirical Analysis of Flaky Tests"
FSE 2014

Flaky Tests

Flaky Tests are relevant for practitioners and highly discussed on the web

Flaky Tests @ Google

Flaky Tests at Google

Google has around 4.2 million tests that run on our continuous integration system. Of these, around 63 thousand have a flaky run over the course of a week. While this represents less than 2% of our tests, it still causes significant drag on our engineers.

SO HOW DO WE APPROACH TEST FLAKINESS?

Studying Test Smells

Symptoms of poor design or implementation choices in test code

Van Deursen et al.
"Refactoring Test Code"
XP 2001

Studying Test Smells

Symptoms of poor design or implementation choices in test code

Van Deursen et al.
"Refactoring Test Code"
XP 2001

Resource Optimism is a test that makes optimistic assumption about the state or the existence of an external resource

Studying Test Smells

Symptoms of poor design or implementation choices in test code

Van Deursen et al.
"Refactoring Test Code"
XP 2001

Indirect Testing is a test that exercises different classes with respect to the corresponding production class

Studying Test Smells

Symptoms of poor design or implementation choices in test code

Van Deursen et al.
"Refactoring Test Code"
XP 2001

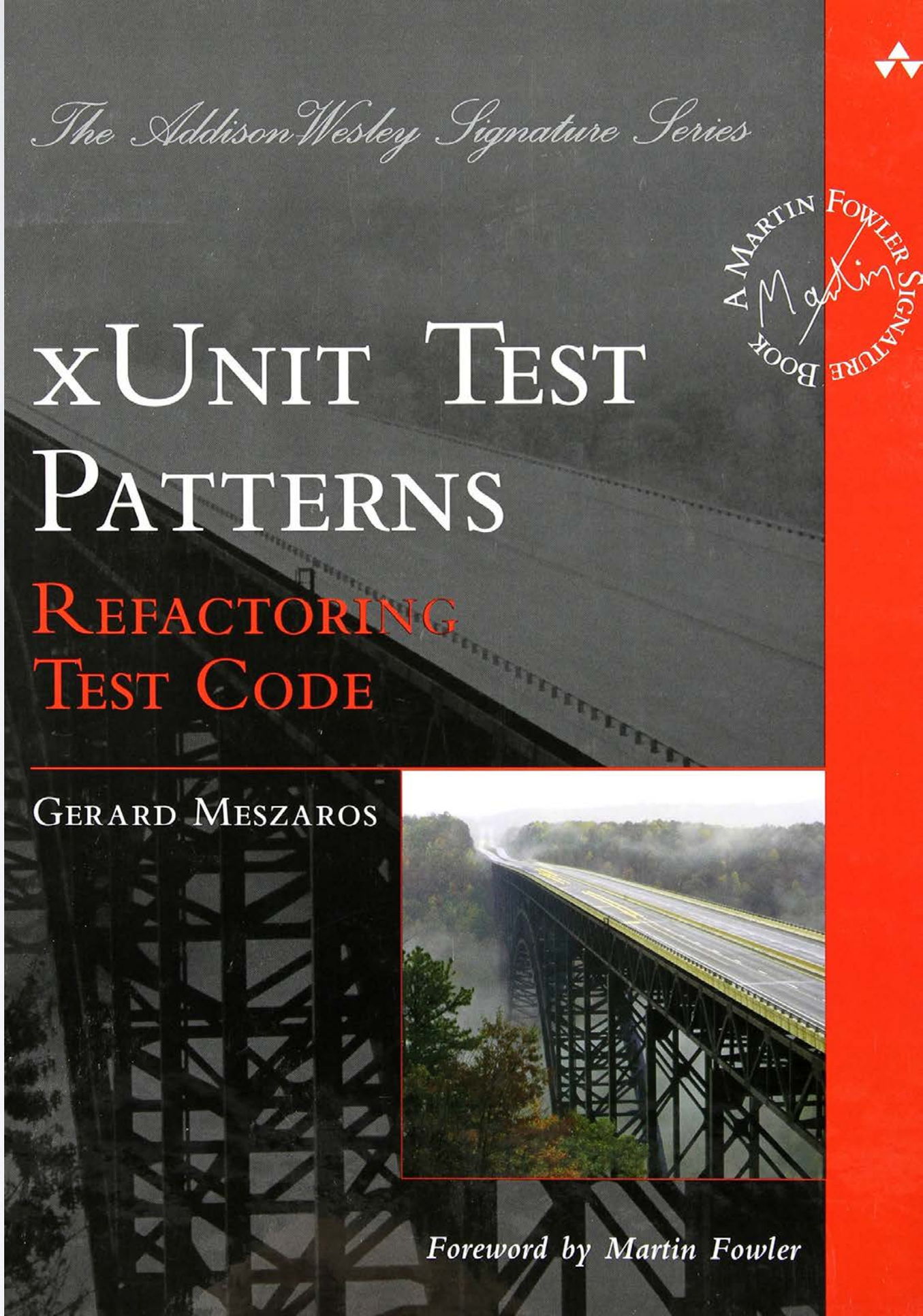
Test Run War is a test that allocates resources that are also used by other methods, possibly causing interferences

2001 research paper

→

Taken up in a 2007 book

18 test smells listed



Research Questions

Research Questions

What are the causes of test flakiness?

Research Questions

What are the causes of test flakiness?

To what extent can flaky tests be explained by the presence of tests smells?

Research Questions

What are the causes of test flakiness?

To what extent can flaky tests be explained by the presence of tests smells?

To what extent does refactoring of test smells help in removing flaky tests?

Software Projects

18

open-source systems randomly selected from Github

Detecting Test Smells

The detector exploits the definition of the smells to detect them

The detector has a precision of 88% and a recall of 100%

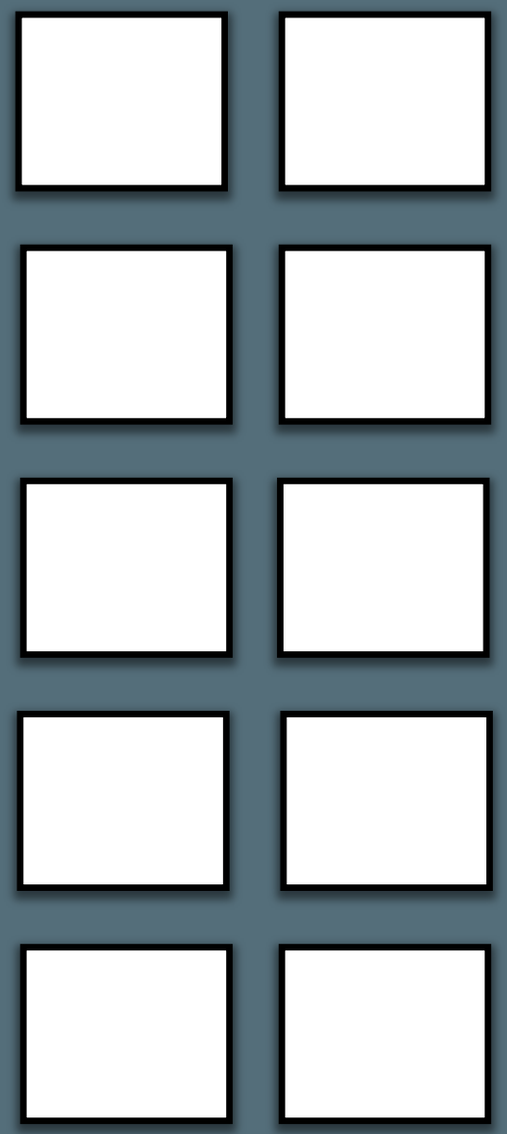
Bavota et al.
"Are Test Smells Really Harmful?"
An Empirical Study
EMSE

Palomba et al.
"On the Diffuseness of Test Smells in Automatically Generated Test Code"
SBST 2016

Detecting Flaky Tests

We ran JUnit ten times

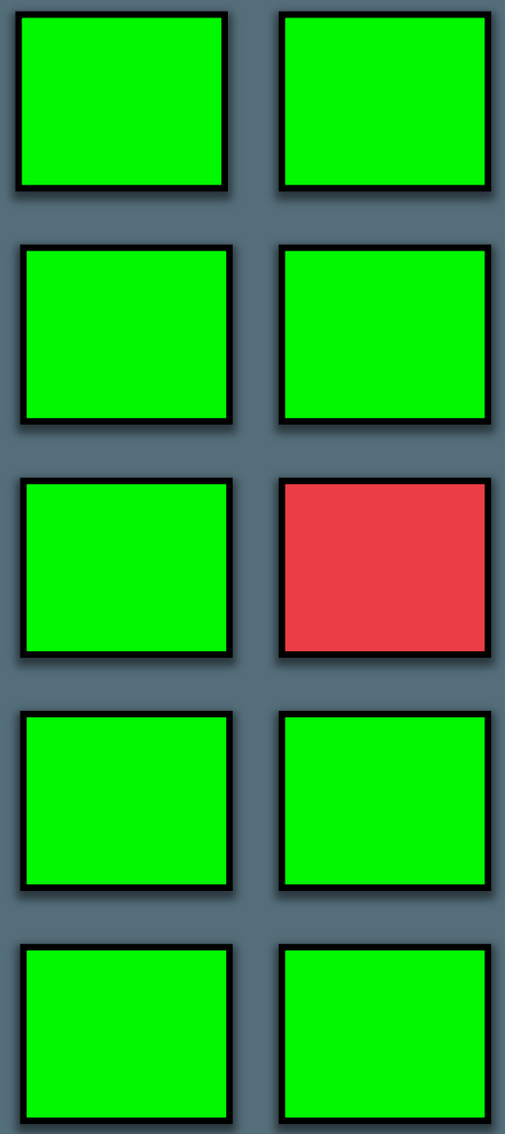
If the output of a test method was different at least once



Detecting Flaky Tests

We ran JUnit class ten times

If the output of a test method was different at least once



Test Smells and Flaky Tests Diffusion

7,812 test smells,
i.e., 40% of the tests

8,829 flaky tests,
i.e., 45% of the tests

Causes of Test Flakiness

We manually linked each flaky test onto one of the 10 root causes defined in the taxonomy by Luo et al.

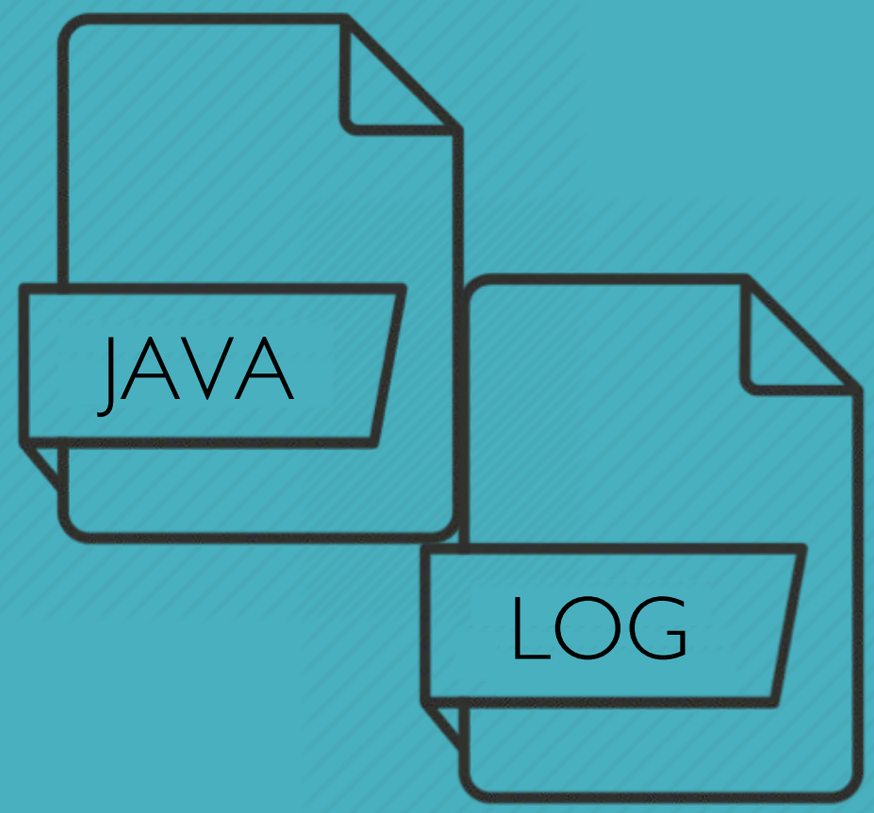
Luo et al.
“An Empirical Analysis of Flaky Tests”
FSE 2014

Causes of Test Flakiness

We manually linked each flaky test onto one of the 10 root causes defined in the taxonomy by Luo et al.

Source code of the test

Exceptions thrown



Luo et al.
"An Empirical Analysis of Flaky Tests"
FSE 2014

Causes of Test Flakiness

27%

Async Wait

A test method making an asynchronous call and that does not wait for the result of the call.

Causes of Test Flakiness

22%

IO issue

The test method does not properly acquire or release one or more to its resources

Causes of Test Flakiness

17%

Concurrency

Different threads interact in a non-desirable manner

Causes of Test Flakiness

Test Ordering

Ordering of tests execution

11%

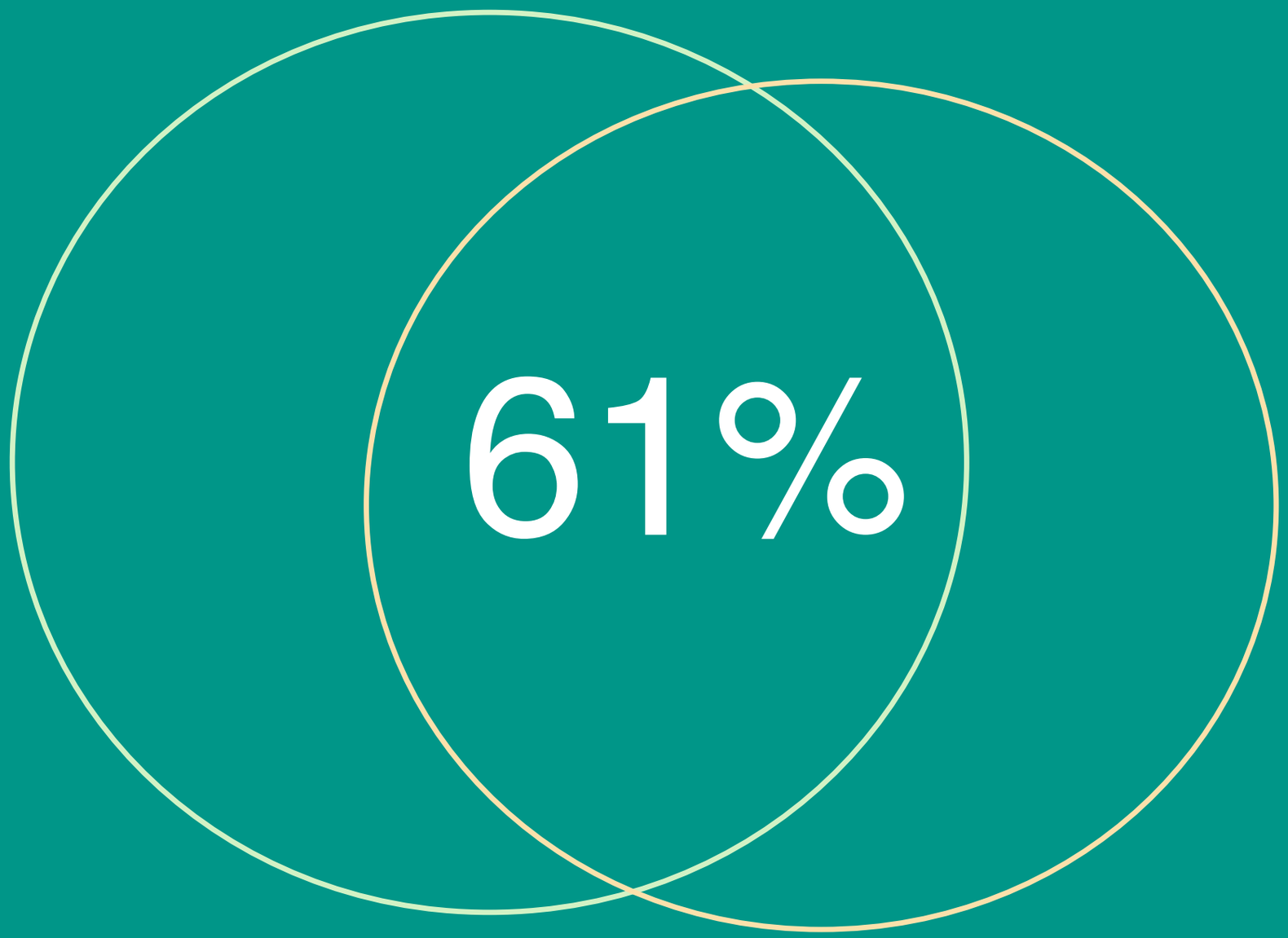
Network

Network performance

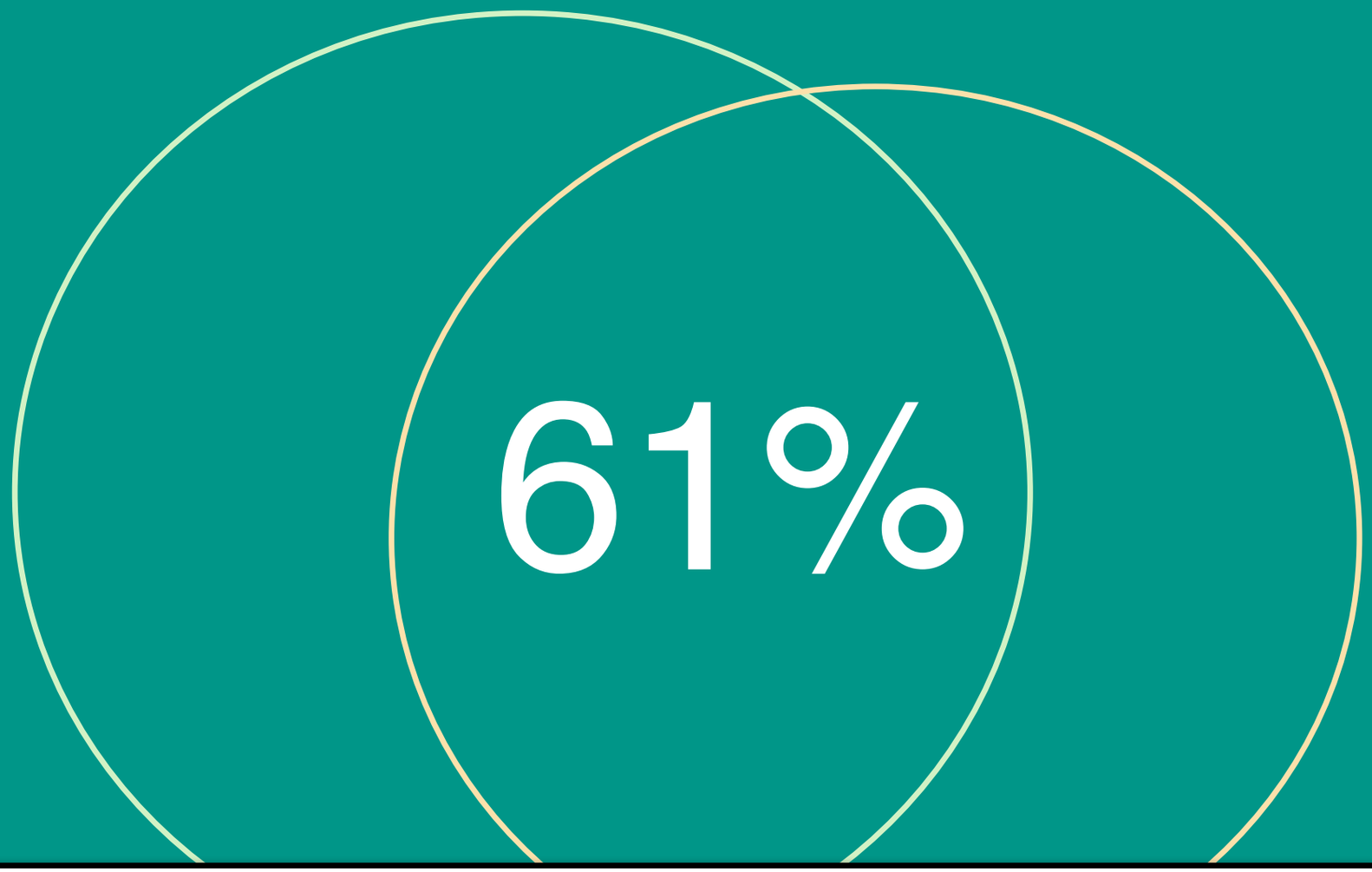
10%

Test Smells vs Flaky Tests

Test Smells vs Flaky Tests



Test Smells vs Flaky Tests



How many of them are casual co-occurrences?

Test Smells vs Flaky Tests

We manually identified the *flakiness-inducing* test smells

61%

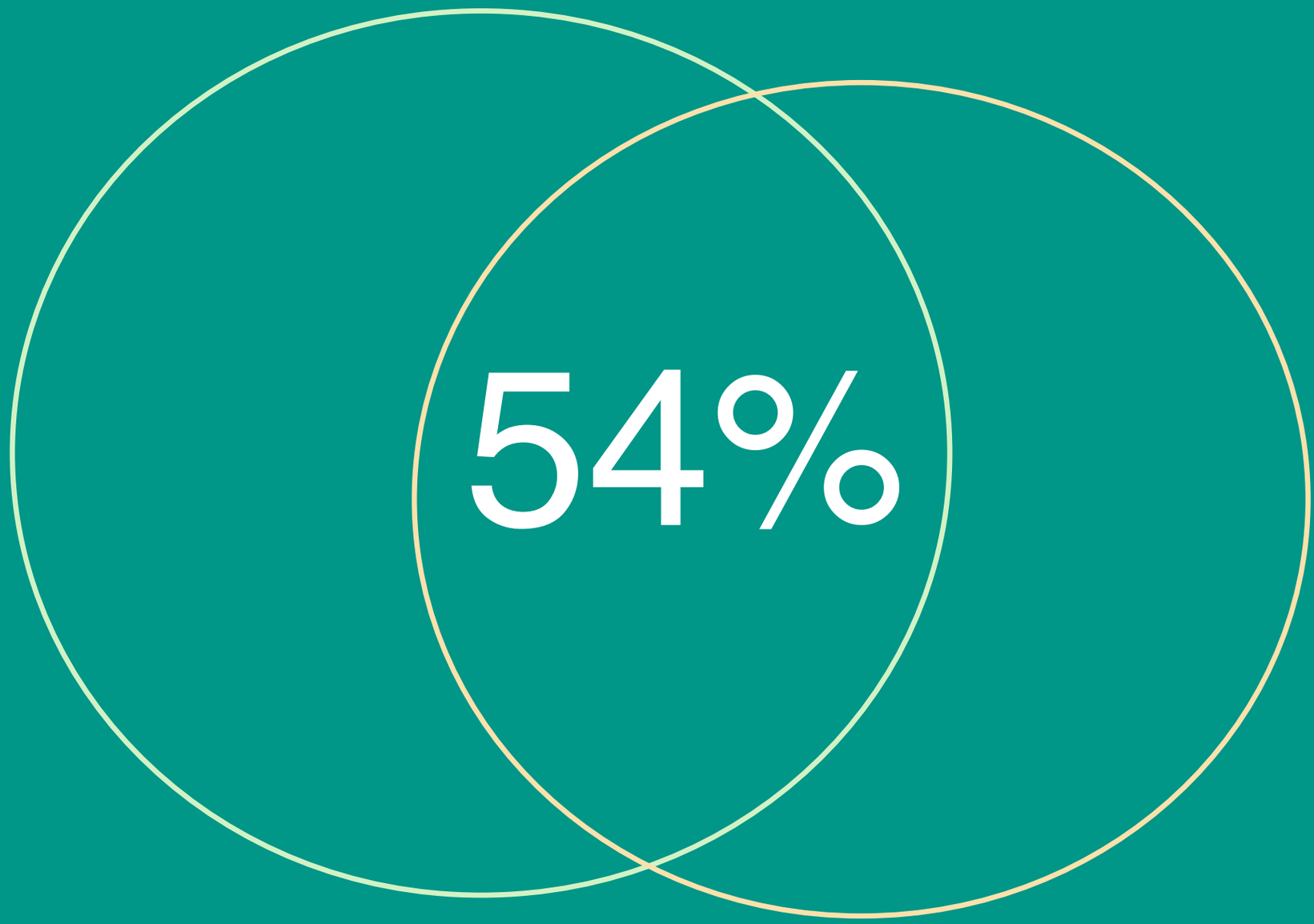
Test Smells vs Flaky Tests

61%

We manually identified the *flakiness-inducing* test smells

A Resource Optimism was casually related to a test case if the flakiness was due to issues in the management of external resources

Test Smells vs Flaky Tests



Test Smells vs Flaky Tests

Resource Optimism

IO issue

Network

54%

Test Smells vs Flaky Tests

Resource Optimism

IO issue

Network

54%

Indirect Testing

Test Ordering

Test Smells vs Flaky Tests

Resource Optimism

IO issue

Network

54%

Indirect Testing

Test Ordering

Test Run War

Concurrency

The Role of Refactoring

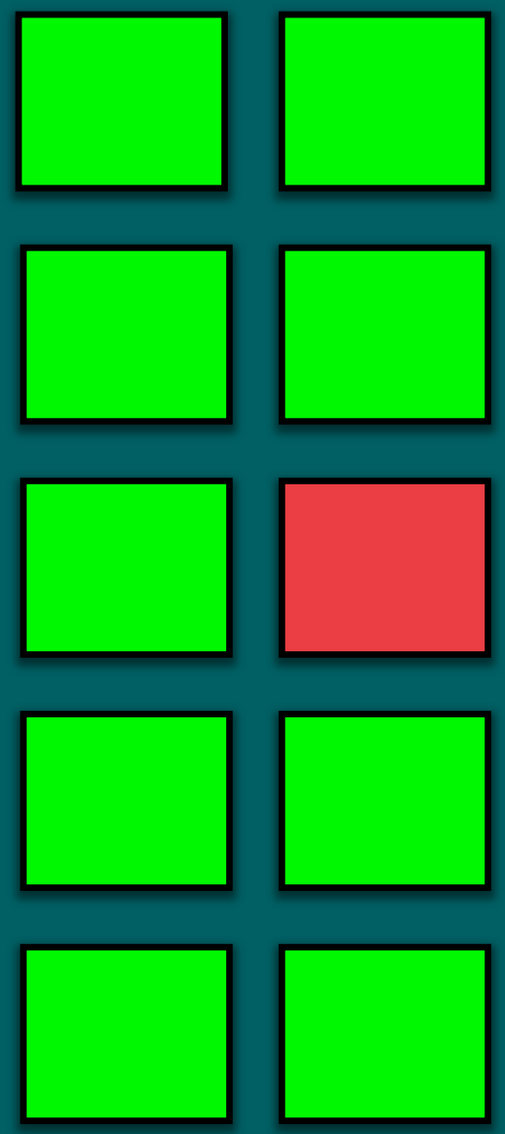
We manually refactored according to the guidelines defined by Van Deursen et al.

Van Deursen et al.
“Refactoring Test Code”
XP 2001

The Role of Refactoring

We re-ran the test smell detection

We re-ran the flaky tests identification



The Role of Refactoring

100%

of the test code refactored did not present a test smell anymore

The Role of Refactoring

100%

of the test code refactored did not present a **flaky test** anymore

The Role of Refactoring

54%

of the total flaky tests were removed by means of refactoring

Morale of the story

1. There is a causal relationship betw
   test smelliness and test flakiness

2. Detecting smells can be done in a
   lightweight manner using tools

3. Refactoring the test smells is curre
   still a manual process

Flaky Tests & Test Smells

azaidman

a.e.zaidman@tudelft.nl