Quem testou o meu teste?

Um breve ensaio sobre qualidade do código de teste

Ivan Machado, D.Sc. - ivan.machado@ufba.br

Softex MPSBR

Software de Qualidade

é o que todos querem (desenvolver, manter, utilizar)

e essa é uma conversa antiga...

http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

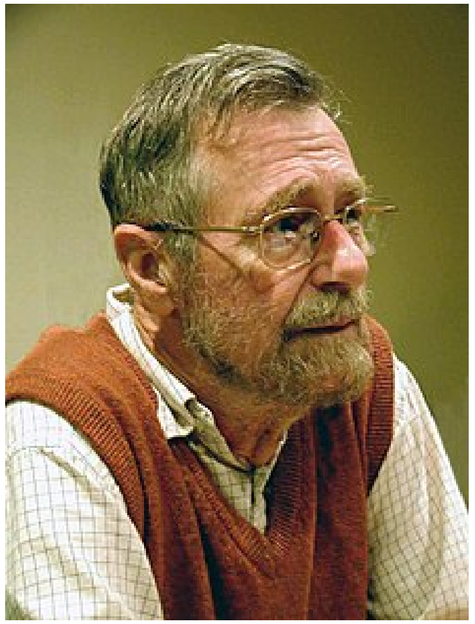
https://www.scrummanager.net/files/nato1968e.pdf

http://homepages.cs.ncl.ac.uk/brian.randell/NATO/NATOReports/

A CRISE DO SOFWARE

A PRINCIPAL MOTIVAÇÃO PARA QUE UM GRUPO DE NOTÁVEIS DECIDISSE INVESTIGAR MAIS A FUNDO COMO DESENVOLVER "SOFTWARE" DE MODO "ENGENHOSO" (A.K.A. SISTEMÁTICO)

"A principal causa da crise do software é que as máquinas se tornaram várias ordens de magnitude mais poderosas! (...) Quando não havia máquinas, programação não era problema algum; quando tínhamos alguns computadores fracos, a programação se tornou um problema brando, e agora temos computadores gigantescos, a programação se tornou um problema igualmente gigantesco."



https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

Nota: Tradução livre

A CRISE DO SOFWARE

levou a uma série de problemas, incluindo:

Projetos que ultrapassaram o orçamento ou o tempo

Software que era muito ineficiente ou de baixa qualidade

Software que não atendiam aos requisitos

Projetos que eram incontroláveis ou difíceis de manter

Projetos de software nunca entregues

essa é uma conversa antiga...

que ainda não saiu de moda!

Como garantir a qualidade do software desenvolvido

CHAPTER

An introduction to modern software quality assurance

2

Bruce R. Maxim and Marouane Kessentini

Department of Computer and Information Science, University of Michigan-Dearborn, Dearborn, MI, USA

2.1 INTRODUCTION

Software quality assurance (SQA) is something everyone talks about, but few seem to want to any spend time on. Many developers have the perception that it is more important to deliver software on time than to try to fix problems before deployment. This perception continues despite the fact that most software failures or disasters could have been avoided using software engineering techniques already known. In many cases, these quality problems could have been predicted because no independent audits of the product were allowed and company management ignored problems widely reported by software users (Hatton, 2007).

There are several major challenges to quality management in the new genera-

Bruce R. Maxim, Marouane Kessentini, Chapter 2 - An introduction to modern software quality assurance, Editor(s): Ivan Mistrik, Richard Soley, Nour Ali, John Grundy, Bedir Tekinerdogan, Software Quality Assurance, Morgan Kaufmann, 2016,

"O teste é uma parte importante do processo de SQA.
O teste deve ser usado em todo o processo de desenvolvimento de software, não apenas no final.
O teste pode ser usado para descobrir defeitos do sistema antes da entrega.
Os testes podem ser usados para melhorar a qualidade do sistema, buscando minimizar os defeitos no produto entregue".

Bruce R. Maxim, Marouane Kessentini, Chapter 2 - An introduction to modern software quality assurance, Editor(s): Ivan Mistrik, Richard Soley, Nour Ali, John Grundy, Bedir Tekinerdogan, Software Quality Assurance, Morgan Kaufmann, 2016,

http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

189

THE TESTING OF COMPUTER SOFTWARE

by

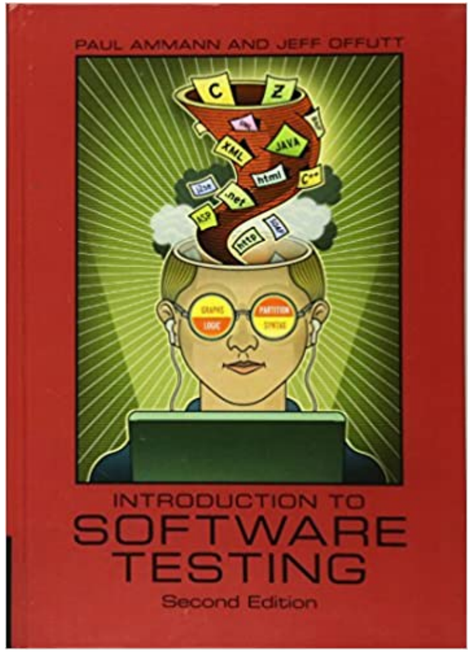
A.I. Llewelyn and R.F. Wickens

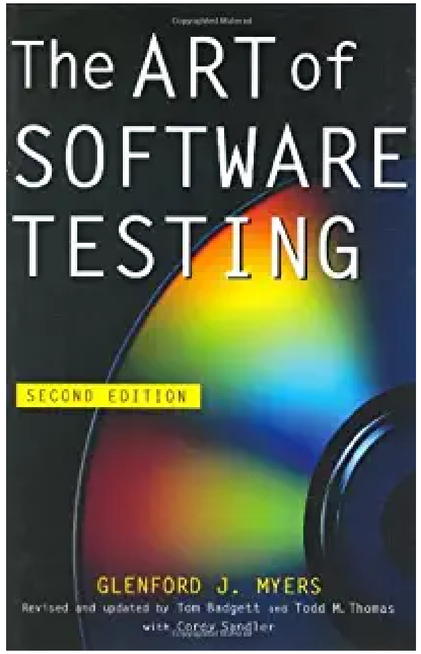
Contents

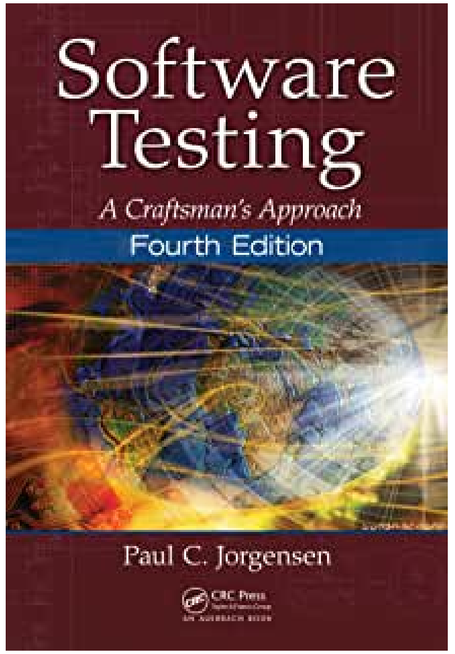
1. Introduction

Computer software has changed rapidly from being merely an adjunct to the hardware to becoming an extremely important part of an installation and one that profoundly affects the overall performance. It is now widely accepted that computer hardware should be given some form of acceptance test by the customer and such tests have formed part of the United Kingdom government's computer contracts for about eight years. However, the weak area of most installations is now that of software.

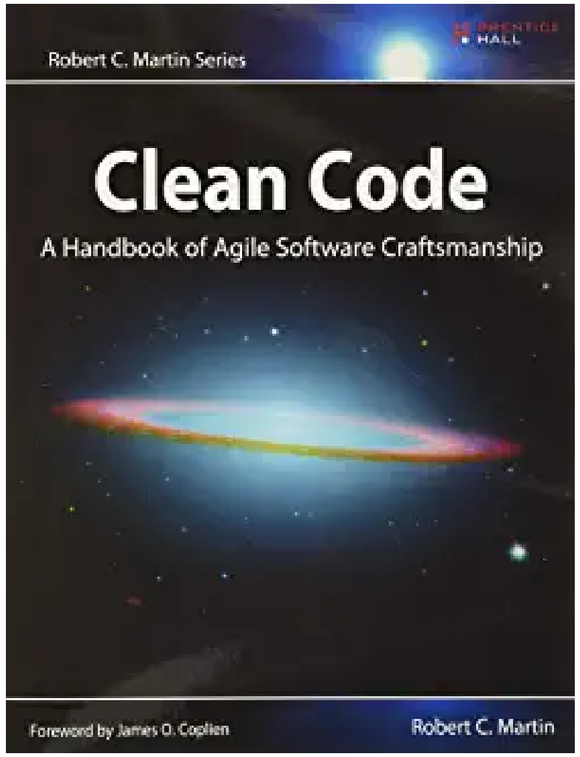
Users complain that the software they received was badly designed and produced, was not delivered on time and when finally delivered, contained serious errors and restrictions and was also inefficient. Unfortunately, 190 these criticisms are often only too true although, happily, it is unusual for all the criticisms to apply simultaneously. The

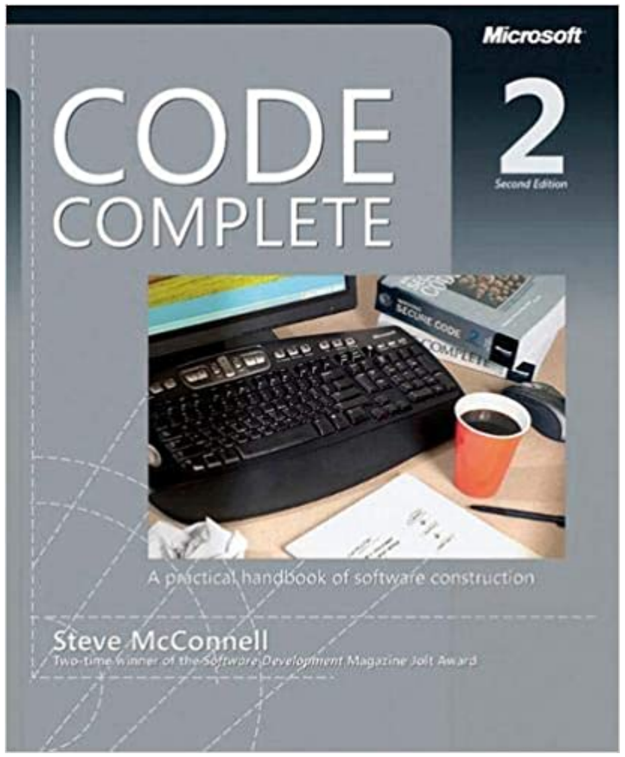

PAUL AMMANN AND JEFF OFFUTT
INTRODUCTION TO
SOFTWARE
TESTING
Second Edition

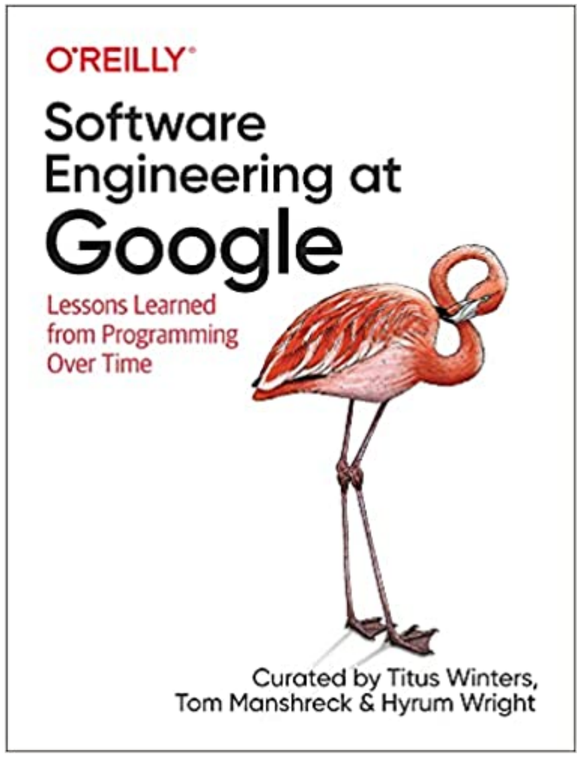

The ART of
SOFTWARE
TESTING
SECOND EDITION
GLENFORD J. MYERS


Software
Testing
A Craftsman's Approach
Fourth Edition
Paul C. Jorgensen
CRC Press


2ª
EDIÇÃO
MÁRCIO EDUARDO DELAMARO
JOSÉ CARLOS MALDONADO
MARIO JINO
Introdução ao
Teste de
Software
ELSEVIER





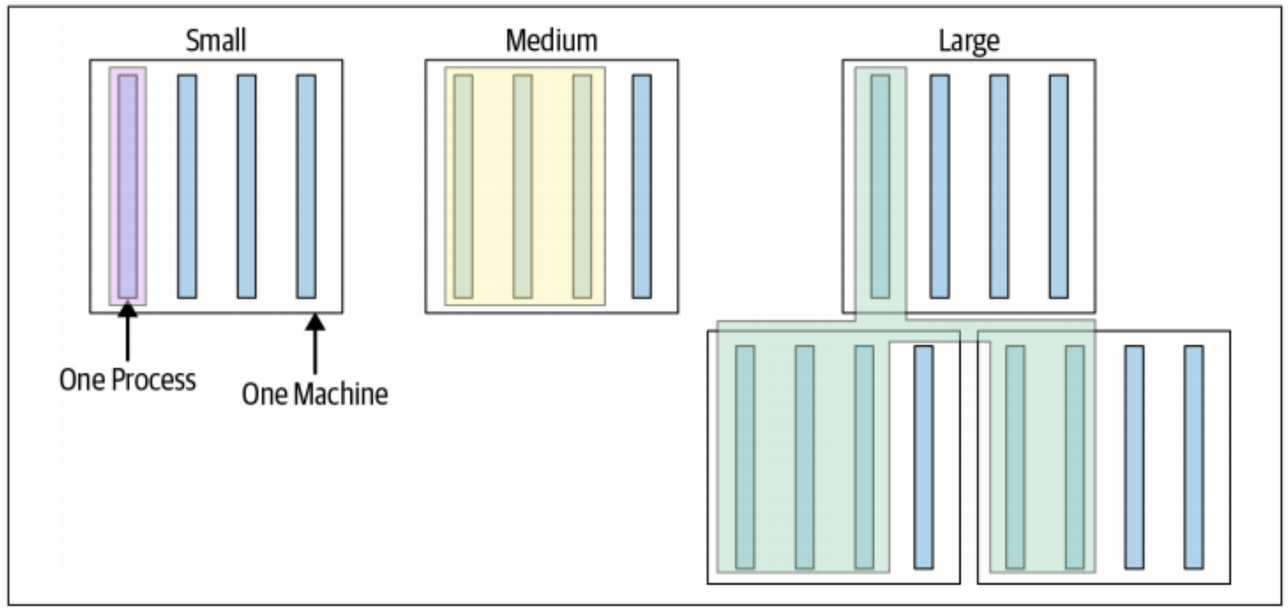


https://abseil.io/resources/swe-book

Observar as práticas de desenvolvimento de empresas que lidam com sistemas de larga escala pode ser uma boa *régua* para avaliar outros cenários

Como as empresas que lidam com sistemas de larga escala testam seus softwares?

Lidar com testes de tamanhos distintos



Titus Winters, Tom Manshreck & Hyrum Wright. Software Engineering at Google - Lessons Learned from Programming Over Time. O'Reilly. 2020

Lidar com escala

Execução de dezenas ou centenas de milhares de casos de teste diariamente

Titus Winters, Tom Manshreck & Hyrum Wright. Software Engineering at Google - Lessons Learned from Programming Over Time. O'Reilly. 2020

Lidar com não-determinismo

Case Study: Flaky Tests Are Expensive

If you have a few thousand tests, each with a very tiny bit of nondeterminism, running all day, occasionally one will probably fail (flake). As the number of tests grows, statistically so will the number of flakes. If each test has even a 0.1% of failing when it should not, and you run 10,000 tests per day, you will be investigating 10 flakes per day. Each investigation takes time away from something more productive that your team could be doing.

In some cases, you can limit the impact of flaky tests by automatically rerunning them when they fail. This is effectively trading CPU cycles for engineering time. At low levels of flakiness, this trade-off makes sense. Just keep in mind that rerunning a test is only delaying the need to address the root cause of flakiness.

If test flakiness continues to grow, you will experience something much worse than lost productivity: a loss of confidence in the tests. It doesn't take needing to investigate many flakes before a team loses trust in the test suite. After that happens, engineers will stop reacting to test failures, eliminating any value the test suite provided. Our experience suggests that as you approach 1% flakiness, the tests begin to lose value. At Google, our flaky rate hovers around 0.15%, which implies thousands of flakes every day. We fight hard to keep flakes in check, including actively investing engineering hours to fix them.

Titus Winters, Tom Manshreck & Hyrum Wright. Software Engineering at Google - Lessons Learned from Programming Over Time. O'Reilly. 2020

Testes Automatizados

Benefícios como alcance de larga escala, reprodutibilidade, e redução nos custos de teste

Testes Automatizados

Benefícios como alcance de larga escala, reprodutibilidade, e redução nos custos de teste

Em contraponto, se não for implementado corretamente, o teste automatizado levará a custos e esforços adicionais e pode até ser menos eficaz do que o teste manual na detecção de falhas

E quais são os potenciais problemas dos testes automatizados?

Assim como o código-fonte de produção, os conjuntos de testes automatizados também são código-fonte e, portanto, devem ser da mais alta qualidade.

Ninguém quer testar seu software em desenvolvimento usando um conjunto de testes de baixa qualidade e com defeitos próprios.

Software test-code Engineering

Conjunto de práticas e métodos usados para desenvolver, verificar e manter sistematicamente códigos de teste de alta qualidade.

Infelizmente, tais práticas e diretrizes nem sempre são seguidas corretamente na prática, resultando em sintomas chamados de *bad smells* (antipadrões) no código de teste (ou simplesmente *test smells*).

V. Garousi and M. Felderer, “Developing, verifying and maintaining high-quality automated test scripts,” IEEE Softw., vol. 33, no. 3, pp. 68–75, 2016.

Test Smells

A preocupação com o design dos testes deve ser premente no processo de automação de testes

Como garantir que os testes sejam compreensíveis, manuteníveis, e eficazes (sem tradeoffs)?

Vamos a um breve exemplo

V. Garousi, B. Kucuk and M. Felderer, "What We Know About Smells in Software Test Code," in IEEE Software, vol. 36, no. 3, pp. 61-73, May-June 2019

```
public class XYTextAnnotationTest {

    /**
     * Confirm that the equals method can distinguish all the
       required fields.
     */
    @Test
    public void testEquals () {
        XYTextAnnotation a1 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        XYTextAnnotation a2 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        assertTrue(a1.equals(a2));

        // text
        a1 = new XYTextAnnotation("ABC", 10.0, 20.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 10.0, 20.0);
        assertTrue(a1.equals(a2));

        // x
        a1 = new XYTextAnnotation("ABC", 11.0, 20.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 11.0, 20.0);
        assertTrue(a1.equals(a2));

        // y
        a1 = new XYTextAnnotation("ABC", 11.0, 22.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 11.0, 22.0);
        assertTrue(a1.equals(a2));

        // font
        a1.setFont(new Font("Serif", Font.PLAIN, 23));
        assertFalse(a1.equals(a2));
        a2.setFont(new Font("Serif", Font.PLAIN, 23));
        assertTrue(a1.equals(a2));

        ...

    }
    ...
}
```

```java
public class XYTextAnnotationTest {

    /**
     * Confirm that the equals method can distinguish all the
       required fields.
     */
    @Test
    public void testEquals () {
        XYTextAnnotation a1 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        XYTextAnnotation a2 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        assertTrue(a1.equals(a2));

        // text
        a1 = new XYTextAnnotation("ABC", 10.0, 20.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 10.0, 20.0);
        assertTrue(a1.equals(a2));

        // x
        a1 = new XYTextAnnotation("ABC", 11.0, 20.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 11.0, 20.0);
        assertTrue(a1.equals(a2));

        // y
        a1 = new XYTextAnnotation("ABC", 11.0, 22.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 11.0, 22.0);
        assertTrue(a1.equals(a2));

        // font
        a1.setFont(new Font("Serif", Font.PLAIN, 23));
        assertFalse(a1.equals(a2));
        a2.setFont(new Font("Serif", Font.PLAIN, 23));
        assertTrue(a1.equals(a2));

        ...

    }
    ...
}
```

```java
/**
     * Confirm that cloning works.
     */
    @Test
    public void testCloning() throws CloneNotSupportedException {
        XYTextAnnotation a1 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        XYTextAnnotation a2 = (XYTextAnnotation) a1.clone();
        assertTrue(a1 != a2);
        assertTrue(a1.getClass() == a2.getClass());
        assertTrue(a1.equals(a2));
    }
```

```java
public class XYTextAnnotationTest {

    /**
     * Confirm that the equals method can distinguish all the
       required fields.
     */
    @Test
    public void testEquals () {
        XYTextAnnotation a1 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        XYTextAnnotation a2 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        assertTrue(a1.equals(a2));

        // text
        a1 = new XYTextAnnotation("ABC", 10.0, 20.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 10.0, 20.0);
        assertTrue(a1.equals(a2));

        // x
        a1 = new XYTextAnnotation("ABC", 11.0, 20.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 11.0, 20.0);
        assertTrue(a1.equals(a2));

        // y
        a1 = new XYTextAnnotation("ABC", 11.0, 22.0);
        assertFalse(a1.equals(a2));
        a2 = new XYTextAnnotation("ABC", 11.0, 22.0);
        assertTrue(a1.equals(a2));

        // font
        a1.setFont(new Font("Serif", Font.PLAIN, 23));
        assertFalse(a1.equals(a2));
        a2.setFont(new Font("Serif", Font.PLAIN, 23));
        assertTrue(a1.equals(a2));

        ...

    }
    ...
}
```

```java
/**
     * Confirm that cloning works.
     */
    @Test
    public void testCloning() throws CloneNotSupportedException {
        XYTextAnnotation a1 = new XYTextAnnotation("Text", 10.0,
                  20.0);
        XYTextAnnotation a2 = (XYTextAnnotation) a1.clone();
        assertTrue(a1 != a2);
        assertTrue(a1.getClass() == a2.getClass());
        assertTrue(a1.equals(a2));
    }
```

eager test

Um outro exemplo

https://testsmells.org/pages/testsmellexamples.html#ConditionalTestLogic

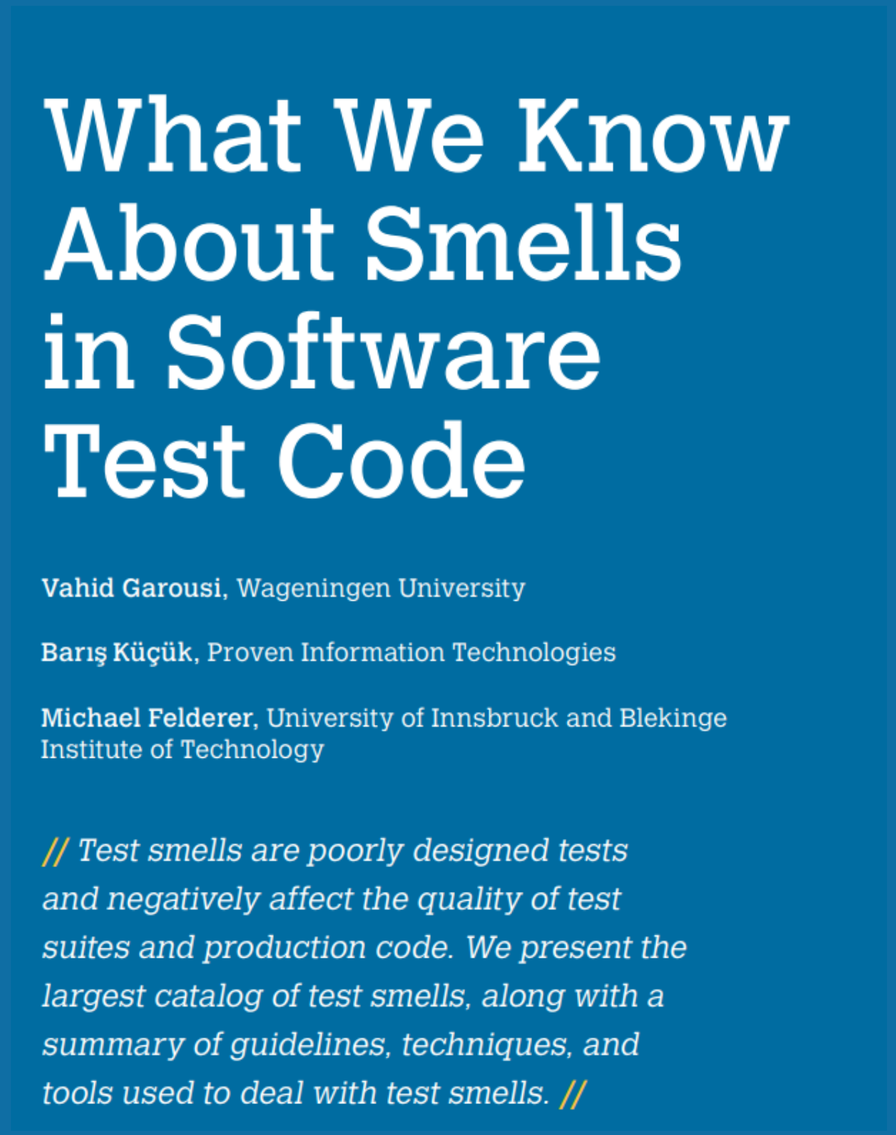
```
@Test
public void testSpinner() {
    for (Map.Entry entry : sourcesMap.entrySet()) {

        String id = entry.getKey();
        Object resultObject = resultsMap.get(id);
        if (resultObject instanceof EventsModel) {
            EventsModel result = (EventsModel) resultObject;
            if (result.testSpinner.runTest) {
                System.out.println("Testing " + id + " (testSpinner)");
                //System.out.println(result);
                AnswerObject answer = new AnswerObject(entry.getValue(), "", new CookieManager(), "");
                EventsScraper scraper = new EventsScraper(RuntimeEnvironment.application, answer);
                SpinnerAdapter spinnerAdapter = scraper.spinnerAdapter();
                assertEquals(spinnerAdapter.getCount(), result.testSpinner.data.size());
                for (int i = 0; i < spinnerAdapter.getCount(); i++) {
                    assertEquals(spinnerAdapter.getItem(i), result.testSpinner.data.get(i));
                }
            }
        }
    }
}
```

```
@Test
public void testSpinner() {
    for (Map.Entry entry : sourcesMap.entrySet()) {

        String id = entry.getKey();
        Object resultObject = resultsMap.get(id);
        if (resultObject instanceof EventsModel) {
            EventsModel result = (EventsModel) resultObject;
            if (result.testSpinner.runTest) {
                System.out.println("Testing " + id + " (testSpinner)");
                //System.out.println(result);
                AnswerObject answer = new AnswerObject(entry.getValue(), "", new CookieManager(), "");
                EventsScraper scraper = new EventsScraper(RuntimeEnvironment.application, answer);
                SpinnerAdapter spinnerAdapter = scraper.spinnerAdapter();
                assertEquals(spinnerAdapter.getCount(), result.testSpinner.data.size());
                for (int i = 0; i < spinnerAdapter.getCount(); i++) {
                    assertEquals(spinnerAdapter.getItem(i), result.testSpinner.data.get(i));
                }
```

O método de teste, testSpinner (), contém várias instruções de controle (ou seja, instruções de fluxo de controle). O sucesso ou a falha do teste é baseado no resultado do método de asserção que está dentro dos blocos de fluxo de controle e, portanto, não é previsível. Isso também aumenta a complexidade do método de teste e, portanto, tem um impacto negativo na manutenção do teste.

```
@Test
public void testSpinner() {
    for (Map.Entry entry : sourcesMap.entrySet()) {

        String id = entry.getKey();
        Object resultObject = resultsMap.get(id);
        if (resultObject instanceof EventsModel) {
            EventsModel result = (EventsModel) resultObject;
            if (result.testSpinner.runTest) {
                System.out.println("Testing " + id + " (testSpinner)");
                //System.out.println(result);
                AnswerObject answer = new AnswerObject(entry.getValue(), "", new CookieManager(), "");
                EventsScraper scraper = new EventsScraper(RuntimeEnvironment.application, answer);
                SpinnerAdapter spinnerAdapter = scraper.spinnerAdapter();
                assertEquals(spinnerAdapter.getCount(), result.testSpinner.data.size());
                for (int i = 0; i < spinnerAdapter.getCount(); i++) {
                    assertEquals(spinnerAdapter.getItem(i), result.testSpinner.data.get(i));
                }
```

O método de teste, testSpinner (), contém várias instruções de controle (ou seja, instruções de fluxo de controle). O sucesso ou a falha do teste é baseado no resultado do método de asserção que está dentro dos blocos de fluxo de controle e, portanto, não é previsível. Isso também aumenta a complexidade do método de teste e, portanto, tem um impacto negativo na manutenção do teste.



What We Know About Smells in Software Test Code

Vahid Garousi, Wageningen University

Barış Küçük, Proven Information Technologies

Michael Felderer, University of Innsbruck and Blekinge Institute of Technology

// *Test smells are poorly designed tests and negatively affect the quality of test suites and production code. We present the largest catalog of test smells, along with a summary of guidelines, techniques, and tools used to deal with test smells.* //

> Test smells são testes mal projetados e afetam negativamente a qualidade das suítes de teste e, consequentemente, do código de produção.

V. Garousi, B. Kucuk and M. Felderer, "What We Know About Smells in Software Test Code," in IEEE Software, vol. 36, no. 3, pp. 61-73, May-June 2019

Consequências negativas da presença de smells no código de teste

Eager Test

“Se você introduziu um bug que quebra o primeiro comportamento, o segundo comportamento pode estar ok, mas você não saberá porque o *test runner* não executará as instruções restantes deste método de teste. E, ao contrário, se o segundo comportamento for interrompido, todo o teste falha, ainda que o primeiro comportamento permaneça inalterado ”.

M. Noback, "A better PHP testing experience Part I: Moving away from assertion-centric unit testing," http://php-and-symfony.matthiasnoback.nl/2014/07/descriptive-unit-tests/, 2014.

General Fixture

Os testes serão frágeis, ou seja, a relação causa-efeito entre o test fixture e os resultados esperados é menos visível. Isso resulta em baixa legibilidade e os testes serão difíceis de entender. Uma alteração feita para um teste afeta os outros testes, pois muitas funcionalidades são cobertas no test fixture.

Como os testes realizam muito trabalho desnecessário, eles serão executados mais lentamente. Isso fará com que os testes demorem muito para serem concluídos.

D. Mathew and K. Foegen, "An analysis of information needs to detect test smells," Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, Singapore, pp. 4–15, 2016.

"Dead Fields"

Esse *test smell* ocorre quando uma classe ou suas superclasses têm campos que nunca são usados por nenhum método de teste. Assim, é um indicativo de que existe uma estrutura de herança inadequada ou que a superclasse entra em conflito com o princípio de responsabilidade única. Nos leva a crer que há atividades de desenvolvimento incompletas ou obsoletas.

Além disso, classes de teste com alta coesão facilitam a compreensão e manutenção do código. Os métodos de teste de baixa coesão seriam *smelly* porque agravam a reutilização, a facilidade de manutenção e a compreensão.

M. Greiler, A. van Deursen, and M.-A. Storey, "Automated detection of test fixture strategies and smells," IEEE International Conference on Software Testing, Verification and Validation, Luxembourg, Luxembourg, pp. 322–331, 2013.

M. Kennedy, "Avoiding 5 common pitfalls in unit testing," https://blog.michaelckennedy.net/2009/08/07/article-avoiding-5-common-pitfalls-in-unit-testing/ , 2009, Last accessed: April 2017.

Test Code Duplication

Afetam a manutenibilidade e a estabilidade do código

Obscure Tests

São mais difíceis de manter e não servem como documentação

Conditional Test Logic

Reduz a capacidade de análise e mudança (*changeability*) do código de teste

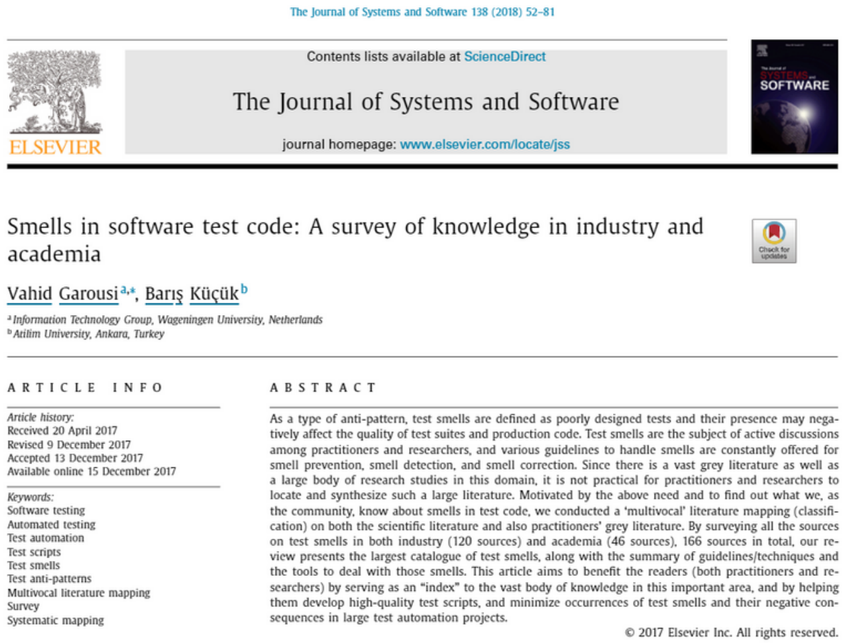
L. Koskela, "Developer test anti-patterns," https://www.youtube.com/watch?v=3Fa69eQ6XgM, 2013, Last accessed: June 2021.

Consequências negativas da presença de smells no código de teste

Os *test smells* por si só não são necessariamente prejudiciais, mas são suas consequências negativas que os tornam indesejáveis.

TEST SMELLS

E o que a indústria SABE a esse respeito?



The Journal of Systems and Software 138 (2018) 52–81

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Smells in software test code: A survey of knowledge in industry and academia

Vahid Garousi[a,*], Barış Küçük[b]

[a] Information Technology Group, Wageningen University, Netherlands
[b] Atilim University, Ankara, Turkey

ARTICLE INFO

Article history:
Received 20 April 2017
Revised 9 December 2017
Accepted 13 December 2017
Available online 15 December 2017

Keywords:
Software testing
Automated testing
Test automation
Test scripts
Test smells
Test anti-patterns
Multivocal literature mapping
Survey
Systematic mapping

ABSTRACT

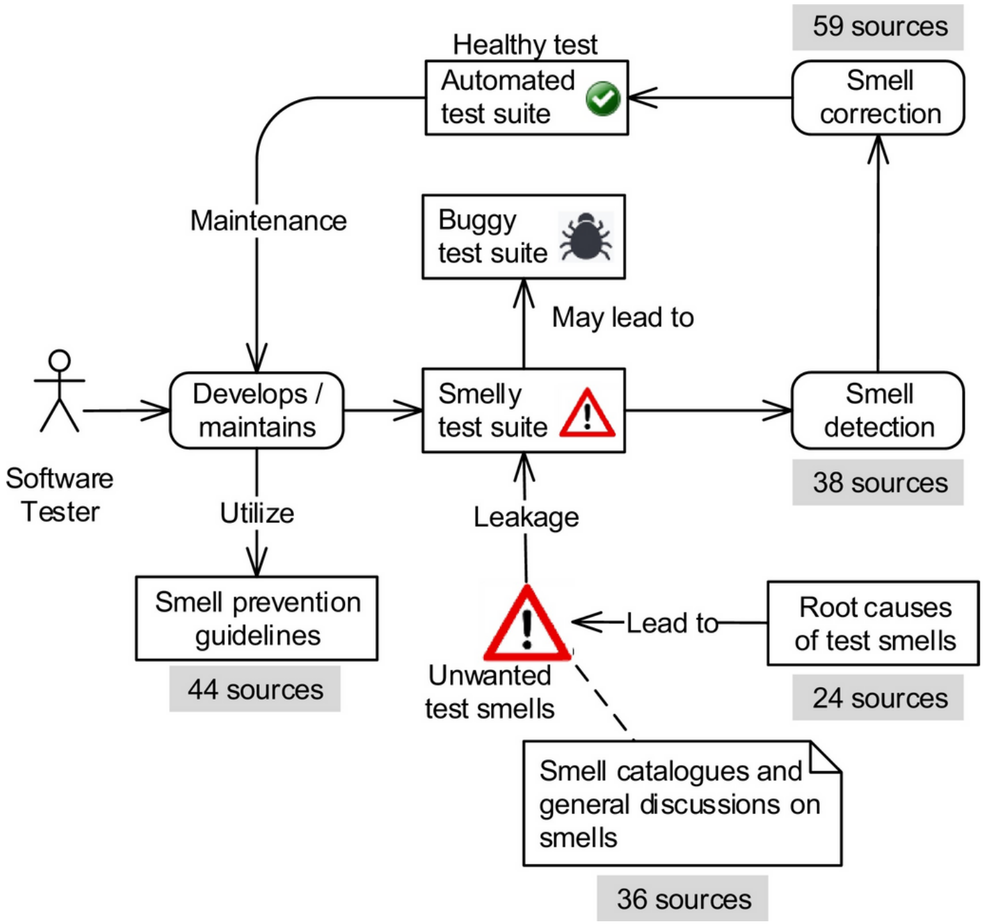
As a type of anti-pattern, test smells are defined as poorly designed tests and their presence may negatively affect the quality of test suites and production code. Test smells are the subject of active discussions among practitioners and researchers, and various guidelines to handle smells are constantly offered for smell prevention, smell detection, and smell correction. Since there is a vast grey literature as well as a large body of research studies in this domain, it is not practical for practitioners and researchers to locate and synthesize such a large literature. Motivated by the above need and to find out what we, as the community, know about smells in test code, we conducted a 'multivocal' literature mapping (classification) on both the scientific literature and also practitioners' grey literature. By surveying all the sources on test smells in both industry (120 sources) and academia (46 sources), 166 sources in total, our review presents the largest catalogue of test smells, along with the summary of guidelines/techniques and the tools to deal with those smells. This article aims to benefit the readers (both practitioners and researchers) by serving as an "index" to the vast body of knowledge in this important area, and by helping them develop high-quality test scripts, and minimize occurrences of test smells and their negative consequences in large test automation projects.

Estudo compreensivo sobre o conhecimento da indústria (*grey literature*) e academia (*mapping study*) a respeito dessa "nova" temática

Estudo de 2017/2018 apresenta 166 fontes sobre *test smells* em código de teste (o maior catálogo até então)

O estudo classifica o conhecimento da indústria e da academia, e busca sintetizar o que se sabe sobre *guidelines* e ferramentas

Vahid Garousi and Barış Küçük. Smells in software test code: A survey of knowledge in industry and academia. Journal of Systems and Software -Volume 138, April 2018, Pages 52-81. Elsevier.



Os autores propuseram um "ciclo de vida" que representa como os test smells se manifestam, e como os profissionais lidam com esses problemas

Os números exibidos na figura indicam a quantidade de fontes analisadas que apontam para cada categoria, com destaque para as fontes que lidam com "prevenção" (44) "detecção" (38) e "correção" (59).

Vahid Garousi and Barış Küçük. Smells in software test code: A survey of knowledge in industry and academia. Journal of Systems and Software -Volume 138, April 2018, Pages 52-81. Elsevier.

01

Prevenção

8 principles of better unit testing

JUnit - Do's and Dont's

Avoiding 5 common pitfalls in unit testing

02

Detecção

Técnicas como test-code review (análogo ao code review tradicional)

Dependente de ferramentas de suporte (para alcançar escala)

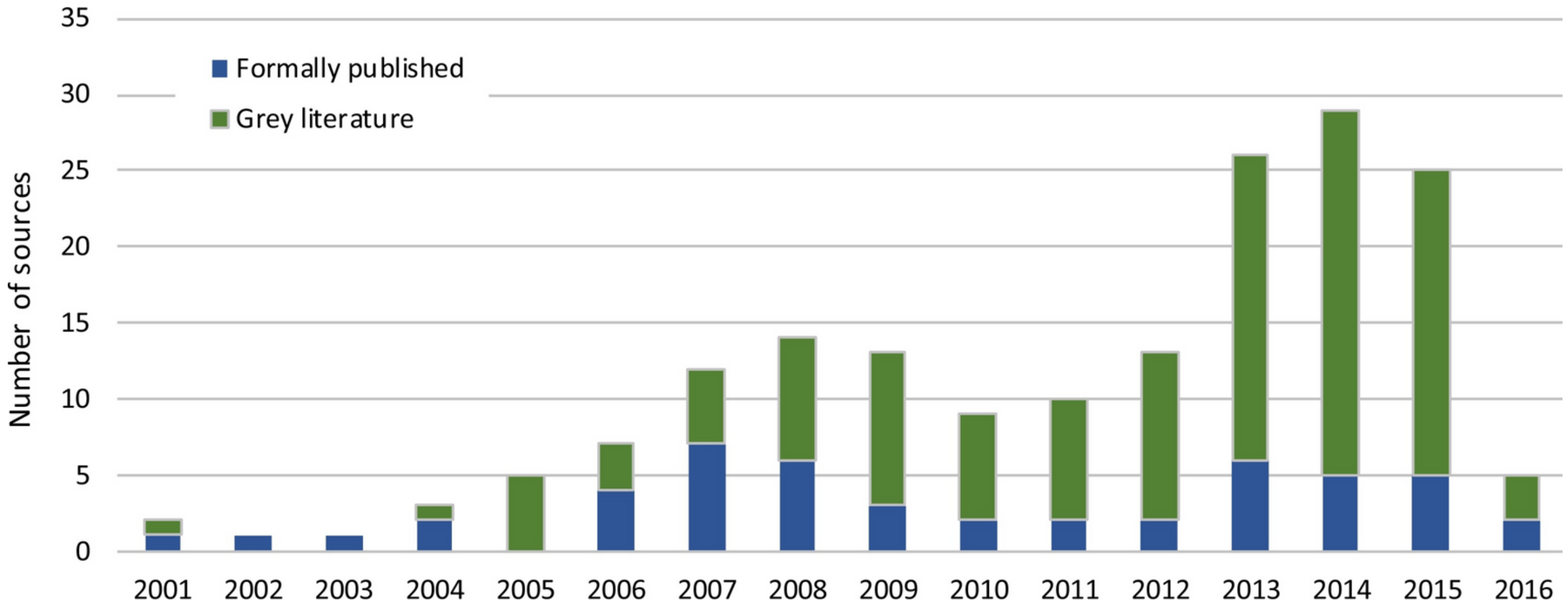
03

Correção

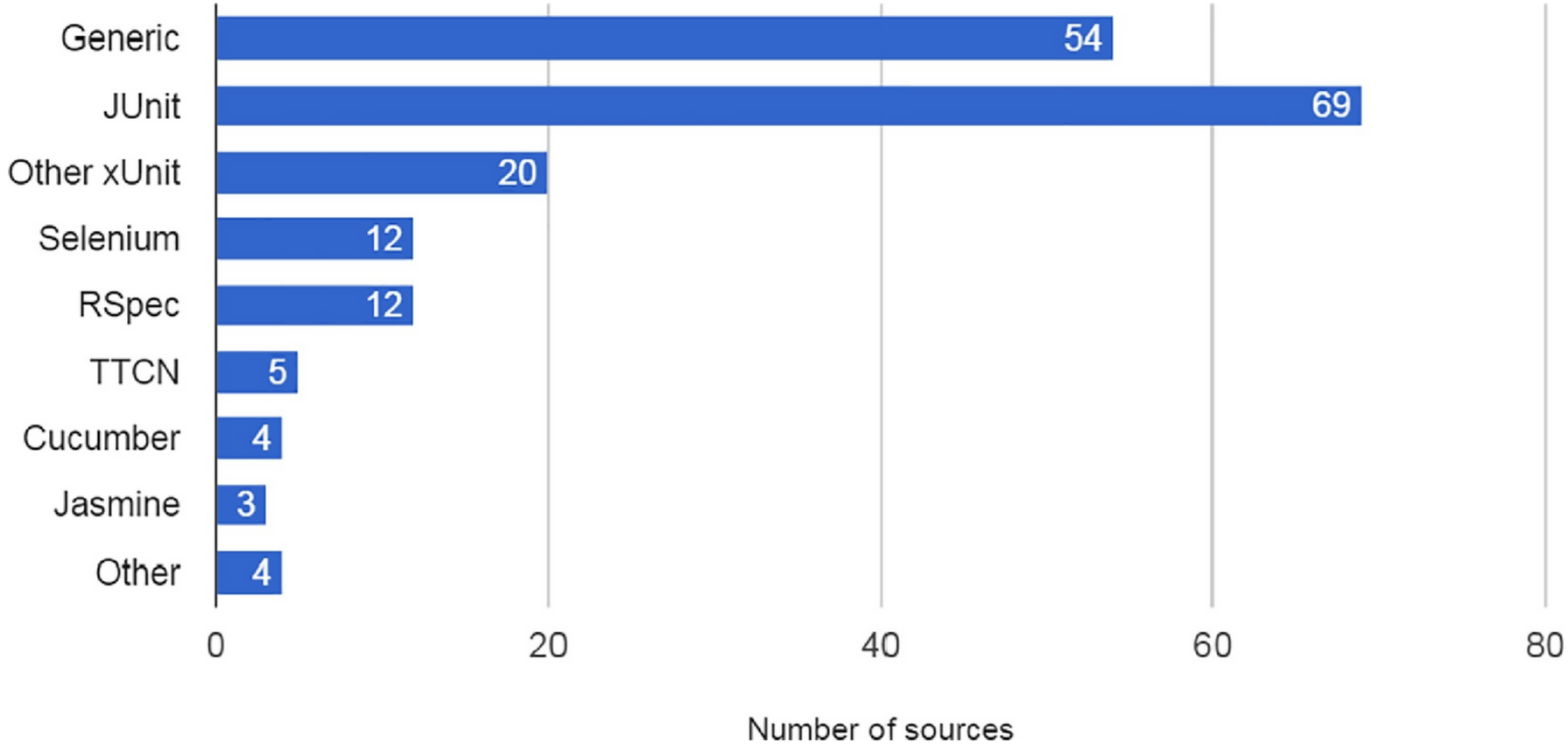
Aplicação de Técnicas de Refactoring

Existe um catálogo (não-exaustivo) de estratégias de correção

Vahid Garousi and Barış Küçük. Smells in software test code: A survey of knowledge in industry and academia. Journal of Systems and Software -Volume 138, April 2018, Pages 52-81. Elsevier.



Vahid Garousi and Barış Küçük. Smells in software test code: A survey of knowledge in industry and academia. Journal of Systems and Software -Volume 138, April 2018, Pages 52-81. Elsevier.



Vahid Garousi and Barış Küçük. Smells in software test code: A survey of knowledge in industry and academia. Journal of Systems and Software -Volume 138, April 2018, Pages 52-81. Elsevier.

Será que os profissionais brasileiros estão cientes do que vem a ser test smells?

E os eventuais problemas decorrentes da sua presença

01

Os profissionais usam práticas de design de casos de teste que podem levar à inserção de *test smells*?

Investigamos se as más escolhas de design podem estar relacionadas com a ocorrência de *test smells*.

02

A experiência do profissional interfere na inserção de *test smells*?

Investigamos se, com o tempo, os profissionais aprimoram o processo de criação de testes.

03

Quais são as práticas presentes no dia a dia dos profissionais que levam à inserção de *test smells*?

Investigamos quais *test smells* estão associados às práticas profissionais mais frequentes.

Nildo Silva Junior, Larissa Rocha, Luana Almeida Martins, Ivan Machado: A survey on test practitioners' awareness of test smells. CIbSE 2020: 462-475

01

60 profissionais de teste foram avaliados

Maioria BA e SP, mas com pessoas de outros estados (2/3 do gênero masculino)

Experientes, em sua grande maioria

45% cria e executa testes; 20% apenas executa

02

Sobre práticas profissionais x inserção de test smells

Os profissionais adotam práticas para o design de casos de teste que levam a introdução de test smells

Essas práticas vêm de padrões (inadequados) pessoais ou da empresa

03

Sobre experiência profissional

Curiosamente, a falta de consciência sobre o assunto faz com que, tanto profissionais inexperientes quanto experientes tratem o problema de modo similar (levando a introdução de smells)

04

Sobre as práticas de codificação

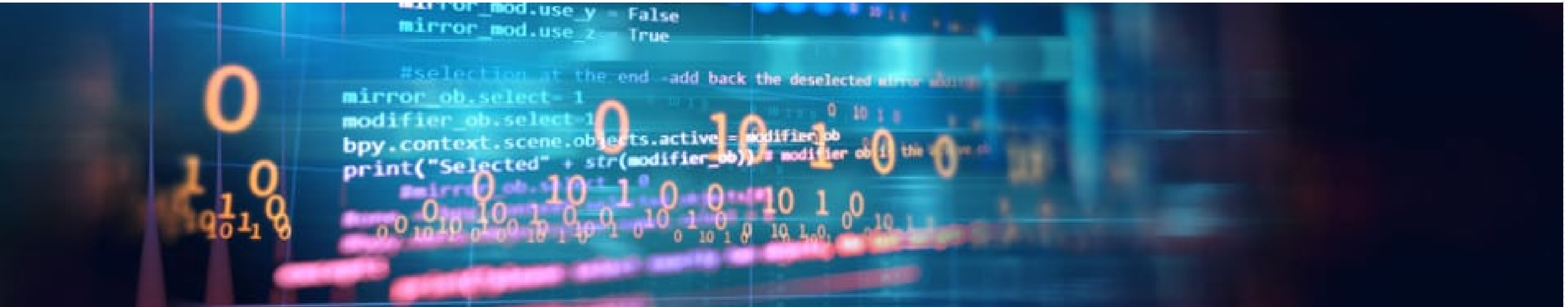
As práticas mais presentes no cotidiano dos profissionais que levam à inserção de test smells são o uso de estruturas condicionais ou de repetição (nos casos de teste) e o uso de dados de configuração genéricos.

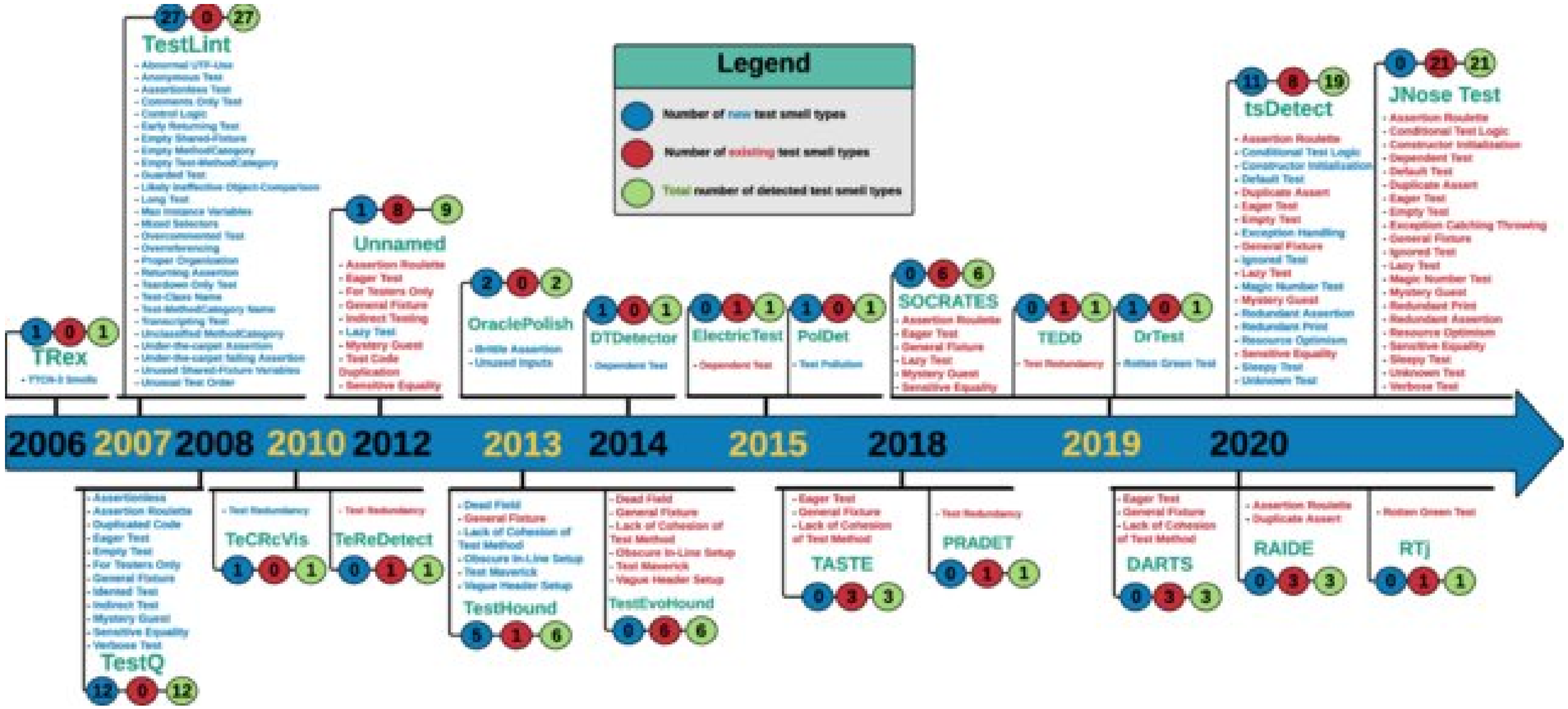
05

Dados preliminares

Mas que podem refletir uma realidade bastante preocupante

E como automatizar o processo de identificação de *test smells*?





Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mohamed Wiem Mkaouer, Ali Ouni, Christian D. Newman, Abdullatif Ghallab, Stephanie Ludi. **Test Smell Detection Tools: A Systematic Mapping Study**. EASE 2021. Trondheim, Norway.

https://conf.researchr.org/details/ease-2021/ease-2021-papers/21/Test-Smell-Detection-Tools-A-Systematic-Mapping-Study

01

Catálogo compreensivo de ferramentas que lidam com test smells

—

Primeiro estudo dessa natureza

02

22 ferramentas disponíveis

—

Alguns projetos abrem os códigos

03

Java, Scala, Smalltalk, and C++ test suites

—

JAVA ainda prevalece. Há uma lacuna explícita para outras linguagens populares, como JavaScript e Python

04

Interesse crescente da comunidade científica e prática

—

2 ferramentas em 2016 para 11 em 2019 e 13 somente em 2020

05

66 tipos de test smells são tratados

—

Há diversas técnicas (Information Retrieval, Regras/heurísticas, Métricas, etc.)

Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mohamed Wiem Mkaouer, Ali Ouni, Christian D. Newman, Abdullatif Ghallab, Stephanie Ludi. **Test Smell Detection Tools: A Systematic Mapping Study**. EASE 2021. Trondheim, Norway.

JNose Test

https://github.com/arieslab/jnose

Detecção automatizada de Test Smells em código Java

User-friendly GUI que automatiza a detecção de 21 test smells em Java

Projeto Open-Source / Identificação baseada em regras

Tássio Virgínio, Luana Almeida Martins, Larissa Rocha Soares, Railana Santana, Adriana Cruz, Heitor Costa, Ivan Machado: JNose: Java Test Smell Detector. SBES 2020: 564-569

Take-home message

A sua empresa está pronta para entregar valor também em testes?

Código de teste também deve ser limpo, e seguir bons padrões

Você costuma validar seu código de teste?

Que tal conhecer as estratégias e as ferramentas que fazem isso "por você"? =)

Adote essas práticas em seu dia-a-dia e compartilhe conosco a sua experiência

Deu certo de verdade? Melhorou algo ou ficou tudo na mesma?

ARIESLAB
Applied Research in Software Engineering

Ivan Machado, D.Sc.

Professor Adjunto no Dep. Ciência da Computação da UFBA
(Salvador - BA - Brasil)

ivan.machado@ufba.br

Nossos interesses

Testes de Software | Startups de Software | Engenharia de Software Sustentável | Mineração de Repositórios de Software

https://arieslab.github.io/

Muito obrigado!

Prof. Ivan Machado, D.Sc.
Depto. Ciência da Computação @UFBA
ivan.machado@ufba.br