An empirical study of automatically-generated tests from the perspective of test smells

Tássio Virgínio
Federal Institute of Tocantins
Paraíso do Tocantins, Brazil
tassio.virginio@ifto.edu.br

Luana Almeida Martins
Federal University of Bahia
Salvador, Brazil
martins.luana@ufba.br

Larissa Rocha Soares
State University of Feira de Santana
Feira de Santana, Brazil
lrsoares@uefs.br

Railana Santana
Federal University of Bahia
Salvador, Brazil
railana.santana@ufba.br

Heitor Costa
Federal University of Lavras
Lavras, Brazil
heitor@ufla.br

Ivan Machado
Federal University of Bahia
Salvador, Brazil
ivan.machado@ufba.br

ABSTRACT

Developing test code can be as or more expensive than developing production code. Commonly, developers use automated unit test generators to speed up software testing. The purpose of such tools is to shorten production time without decreasing code quality. Nonetheless, unit tests usually do not have a quality check layer above testing code, which might be hard to guarantee the quality of the generated tests. A strategy to verify the tests quality is to analyze the presence of test smells in test code. Test smells are characteristics in the test code that possibly indicate weaknesses in test design and implementation. Their presence could be used as a quality indicator. In this paper, we present an empirical study to analyze the quality of unit test code generated by automated test tools. We compare the tests generated by two tools (Randoop and EvoSuite) with the existing unit test suite of twenty-one open-source Java projects. We analyze the unit test code to detect the presence of nineteen types of test smells. The results indicated significant differences in the unit test quality when comparing data from the automated unit test generators and existing unit test suites.

CCS CONCEPTS

• **Software and its engineering** → Software testing and debugging.

KEYWORDS

Unit Testing, Automated Software Testing, Test Smells

ACM Reference Format:
Tássio Virgínio, Luana Almeida Martins, Larissa Rocha Soares, Railana Santana, Heitor Costa, and Ivan Machado. 2020. An empirical study of automatically-generated tests from the perspective of test smells. In *34th Brazilian Symposium on Software Engineering (SBES '20), October 21–23, 2020, Natal, Brazil.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3422392.3422412

SBES '20, October 21–23, 2020, Natal, Brazil

ACM ISBN 978-1-4503-8753-8/20/09…$15.00
https://doi.org/10.1145/3422392.3422412

1 INTRODUCTION

Unit testing is a widely applied strategy for software testing. It aims to test individual units of the program independently. Unit tests are able to anticipate potential problems in the code with a lower cost, compared to tests performed later in the software development process [8]. They can be developed either manually, in conjunction with production code, or automatically, through automated test generators, such as Evosuite[1] and Randoop[2]. Automated unit test generators may test behaviors of production code that humans can rarely exercise [8]. However, providing well-designed tests is still challenging for both manual and automatically generated unit tests [4, 8].

Poorly-written tests can be difficult to comprehend and, therefore, make it difficult for testers to maintain the code and detect failures in the software [2, 8]. Additionally, when a test returns a failure, the developer still needs to understand what caused the failure, by manually analyzing the test code [8]. Recent studies also show that poorly designed tests limit fault detection capabilities and can induce non-deterministic behaviors [13]. Even if they are generated continuously and may not need to be maintained, the code design is relevant for automated test generation techniques [8].

In this scenario, a key strategy to check test code quality is to detect test smells [1, 2, 6–8, 15]. Test smells are characteristics in the test code that possibly indicate weaknesses in test code or design [2, 3, 7]. They may impair comprehensibility, readability, and maintainability of the test code [7, 8]. To support the development of high quality test code, there is a set of guidelines to handle test smells, such as smells prevention, detection and correction [7]. Nonetheless, there is still few evidence about the quality of test code generated by automated tools regarding the presence of test smells.

In this paper, we investigate the quality of test code from the perspective of test smells. We present an empirical study to analyze the quality of tool-generated unit tests and the existing unit tests from software projects. We select twenty-one Java projects and automatically generated test suites with the support of two widely referenced tools, Evosuite and Randoop [4, 5]. We use `JNose Test` [16] to analyze the quality of the test suites. As a result, we could verify significant differences between the existing tests in the selected projects and the test suites generated with the tools.

[1]Evosuite is available at: http://www.evosuite.org/
[2]Randoop is available at: https://randoop.github.io/randoop/

2 BACKGROUND

Test smells are commonly described as bad practices for test code design and implementation [9]. The presence of test smells may serve as an indicator of low-quality software, since they may reduce comprehension, readability and maintainability of the test code [8].

Dozens types of test smells have been defined in the literature [3, 10, 14]. In this study we analyze 19 test smells, which were validated into open source systems [14]. They are: Assertion Roulette (AR), Conditional Test Logic (CTL), Constructor Initialization (CI), Default Test (DT), Dependent Test (DpT), Duplicate Assert (DA), Eager Test (ET), Empty Test (EmT), Exception Catching Throwing (ECT), General Fixture (GF), Ignored Test (IgT), Lazy Test (LT), Magic Number Test (MNT), Mystery Guest, Redundant Print, Redundant Assertion (RA), Resource Optimism (RO), Sensitive Equality (SE), Sleepy Test (ST), verbose Test (VT), and Unknown Test (UT).

Next, we briefly present the test smells detection tool, JNose Test, and the two automated test generators tools used in this study.

2.1 JNose Test

The `JNose Test` detects test smells and calculates the code coverage, and was used in this empirical evaluation to analyze the quality of test sets [16]. It builds an AST (Abstract Syntax Tree) from the test that is analyzed by each detection component, which applies the test smells detection rules. The detection rules are reused from tsDetect [8, 15], which has an accuracy score ranging from 85% to 100% and a recall score from 90% to 100% [14]. `JNose Test` added a set of original features, such as: (i) graphical interface; (ii) automation of data input and output processes; (iii) rules to quantify the quantity of each type of test smells; (iv) parallel processing for the detection of test smells in each test class; and (v) component to calculate code coverage.

2.2 Automated Test Generators

Automated test generation tools allow test code scripts execution with minimal or no human intervention. Consequently, such tools could yield the repeatability, predictability, and efficiency of the test code [7]. In this study, we selected two automated test generators, EvoSuite and Randoop.

EvoSuite (Automatic Test Suite Generation for Java) implements a hybrid approach by combining search-based testing and dynamic symbolic execution to generate test suites towards satisfying a coverage criterion [4, 5]. **Randoop** (RANDom tester for Object-Oriented Programs) uses JUnit to generate random unit tests in the Java language. It works by using the feedback of the executions as inputs to avoid data redundancy [11].

3 EXPERIMENT PLANNING

This section describes the planning used to conduct this research, research questions, hypothesis, and study design.

3.1 Research Questions

A main research question (RQ) drives this investigation: *How do test generators influence the quality of unit tests regarding the presence of test smells?* This question was refined into the sub-questions:

- RQ_1 *Does the quality of test code generated by Randoop differ from those generated by Evosuite?*
- RQ_2 *Does the quality of test code generated by Randoop differ from the existing unit tests of the project?*
- RQ_3 *Does the quality of test codes generated by Evosuite differ from the existing unit tests of the project?*

3.2 Hypotheses, parameters, and variables

To answer the RQ_1, we define the following hypotheses:

Null Hypothesis (H_01): there is no statistically significant difference between the amount of test smells detected in the code generated by Evosuite and Randoop.
Alternative Hypothesis (H_11): there is a statistically significant difference between the amount of test smells detected in the code generated by Evosuite and Randoop.

To answer the RQ_2, we define the following hypotheses:

Null Hypothesis (H_02): there is no statistically significant difference between the amount of test smells detected in the code generated by Randoop and the existing test code in the project.
Alternative Hypothesis (H_12): there is a statistically significant difference between the amount of test smells detected in the code generated by Randoop and the existing test code in the project.

To answer the RQ_3, we define the following hypotheses:

Null Hypothesis (H_03): there is no statistically significant difference between the amount of test smells detected in the code generated by Evosuite and the existing test code in the project.
Alternative Hypothesis (H_13): there is a statistically significant difference between the amount of test smells detected in the code generated by Evosuite and the existing test code in the project.

3.3 Design

Figure 1 shows an overview of the experimental study procedures. Initially, we select 21 open source projects from the GitHub repository following the criteria: (i) project under any open source license, (ii) developed in Java, (iii) projects containing unit tests written with the JUnit library; and (iv) projects configured to use Maven (Figure 1-1). Moreover, we do not set any constraints regarding the projects' size or domain because these characteristics could influence the test suite generation [8].

After selecting the projects (Figure 1-1), we specify all the classes of production that we want to analyze. At the end of the process, each tool generates a corresponding test class, linked to the given production class (Figure 1-2). Then, we obtain three sets of unit tests (Figure 1-3) and we run the `JNose Test` to detect test smells in those test suites (Figure 1-4). As a result, we obtain three datasets regarding test smells for each selected project, generating a single dataset at the end (Figure 1-5).

From the dataset, we perform statistical tests. Since we have three treatments - unit tests from *(i)* Evosuite, *(ii)* Randoop and *(iii)* Existing tests -, we combine them in pairs, using the production classes as a traceability unit (Figure 1-6). Next, we execute the statistical tests (Figure 1-7). We verify the data distribution through the Shapiro-Wilk test and we use the Wilcoxon test to verify the hypotheses, with a significance level of 5%. If the p-value returned by these tests is less than the level of significance, then the samples come from populations with different distributions.

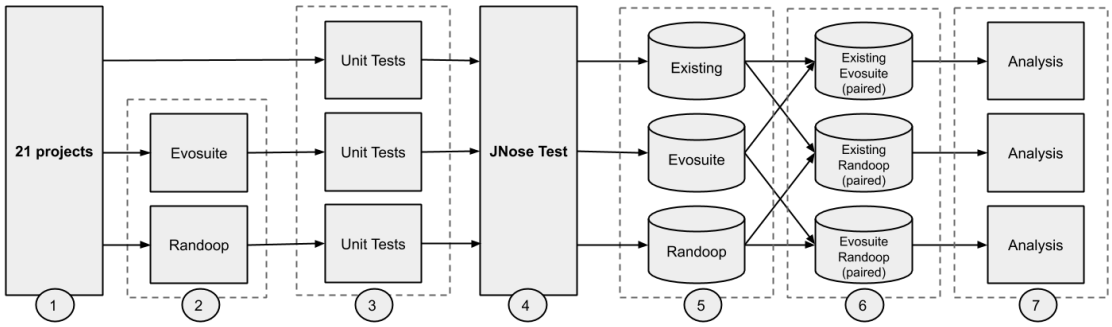

Figure 1: Overview of the experimental study procedures

Table 1: Analyzed test suites - raw data

Suite	LOC	#Methods	#Type	#Test Smells	Occurrence
Existing	168,839	10,141	18	28,825	97.03%
Evosuite	639,979	31,017	14	228,233	99.95%
Randoop	3,733,485	148,284	8	858,887	99.53%

4 RESULTS

The dataset has 5,445 rows, which correspond to the test classes of the projects. The data is organized in 34 columns: project name, test class and production class location, nineteen types of test smells, the number of test class lines and the number of test methods, and ten columns with code coverage data.

Table 1 characterizes the test suites. The *Existing tests* in the projects presented fewer LOC (168,839 lines), number of methods (10,141), and test smells detected (28,825) compared to the other test suites. Conversely, the tests generated by *Randoop* had the highest amount of LOC (3,733,485 lines), number of methods (148,284) and test smells detected (858,887). Regarding the occurrence of test smells, the *Existing* tests suite presented the highest occurrence of distinct types of test smells (18 types), although it generated fewer test smells (28,825) and less occurrence of test smells than the other test suites (97.03%). In turn, Randoop presented the smallest amount of distinct types of test smell (8 types), although it generated more test smells than the others (858,887).

Table 2 shows the amount of each type of test smell per test suite. From our analysis, the test smell type that stood out most in the three test suites was *Lazy Test*, representing 75.61% of test smells in Evosuite, 47.39% in Randoop, and 47.86% in Existing Tests. In addition, for the Existing Tests, other frequent test smells were *Exception Catching Throwing* (14.76%) and *Eager Test*; we also found for the Randoop, the test smells *Exception Catching Throwing* (17.26%) and *Conditional Test Logic* (17.26%); and for the Evosuite, we found *Exception Catching Throwing* (13.48%) and *Eager Test* (4.69%).

In order to leverage the characteristics of the test smells in each test suite, we performed an analysis of the co-occurrences of the test smells using the Spearman correlation test. Figure 2 shows the co-occurrences of test smells in Existing Tests. There is one "strong" correlation between *Resource Optimism* and *Mystery Guest*. Figure 3 shows the co-occurrences for Randoop, which has two "very strong" correlations, *Conditional Test Logic* with *Exception Catching Throwing*, and *Eager Test* with *Lazy Test*. Figure 4 shows the co-occurrences for the Evosuite, which has three "very strong" correlations *Lazy Test* with *Exception Catching Throwing*, *Mystey Guest* with *Resource optimism*, and *Empty Test* with *Unknown Test*.

Table 2: Number of types of test smells per test suite

Test Smell	Existing (%)	Randoop (%)	Evosuite (%)
AR	1,852 (6.42)	86,233 (10.04)	4,953 (2.17)
CI	140 (0.49)	0 (0.00)	0 (0.00)
CTL	485 (1.68)	148,284 (17.26)	28 (0.01)
DA	747 (2.59)	0 (0.00)	1,236 (0.54)
DT	0 (0.00)	0 (0.00)	0 (0.00)
ECT	4,256 (14.76)	148,284 (17.26)	30,767 (13.48)
EmT	20 (0.08)	0 (0.00)	102 (0.04)
ET	2,637 (9.15)	61,101 (7.11)	10,713 (4.69)
GF	660 (2.29)	0 (0.00)	26 (0.01)
IgT	41 (0.14)	0 (0.00)	0 (0.00)
LT	13,797(47.86)	407,019 (47.39)	172,563 (75.61)
MG	128 (0.44)	0 (0.00)	176 (0.08)
MNT	1137(3.94)	297 (0.04)	6,850 (3.00)
PS	108 (0.37)	0 (0.00)	0 (0.00)
RA	32 (0.11)	0 (0.00)	1 (0.00)
RO	282 (0.98)	0 (0.00)	150 (0.07)
SE	518 (1.80)	1,682 (0.20)	533 (0.23)
ST	8 (0.04)	0 (0.00)	0 (0.00)
UT	1,977 (6.86)	5,987 (0.70)	135 (0.07)
Total	28,825 (100)	858,887 (100)	228,233 (100)

We analyzed the hypotheses with the Wilcoxon test. It returned a p-value of $< 2.2^{-16}$ to the paired datasets: (i) Randoop-generated tests *vs* Evosuite-generated tests; (ii) Existing Tests *vs* Randoop-generated tests; and (iii) Existing tests *vs* Evosuite-generated tests.

5 ANALYSIS AND DISCUSSION

In this section, we discuss the main findings of the empirical study.

5.1 Evaluation of results and implications

When comparing the automated tools with existing test suites, the automated test suites presented a higher number of test smells, but a lower number of types of test smells, especially for the Randoop test suite. It may indicate that these tools probably create specific types of smells, which could be removed by refactorings approaches and tools.

Among all types of test smells that we detected, *Lazy Test* was the most frequent one in both automated and existing test suites. *Lazy Test* might be an effect of the polymorphism. Each overloaded production method needs a test method to reproduce its behavior. Therefore, several test methods appear to test the same production method. It might indicate that the concept of *Lazy Test* should be improved to deal with overloaded methods. Listing 1 shows the

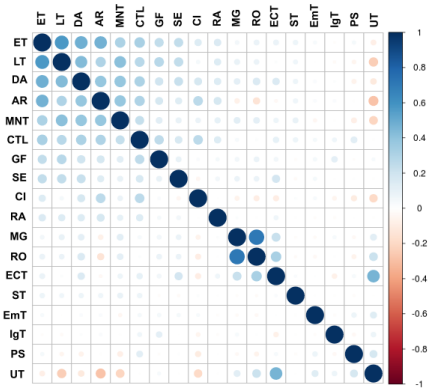

Figure 2: Co-occurrence of test smells in Existing Test Suite

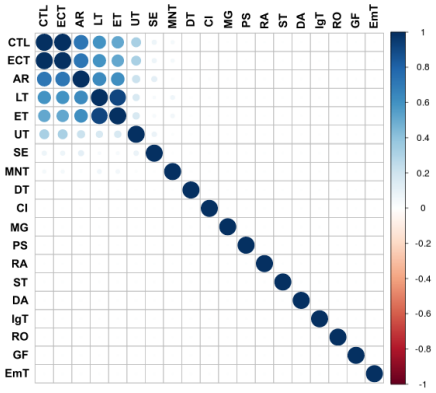

Figure 3: Co-occurrence of test smells in Randoop Test Suite

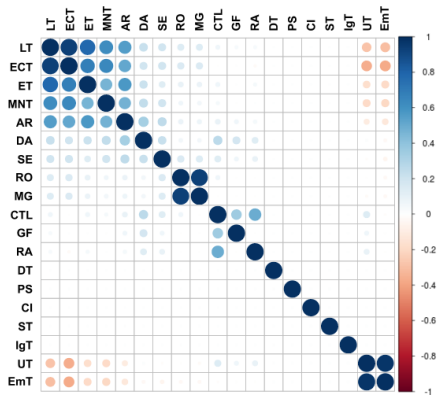

Figure 4: Co-occurrence of test smells in Evosuite Test Suite

existing test code testing the `parse` overloaded methods (lines 989 and 995), from the `CSVParser` production class of the `commons-csv` project. For the existing test suite, *Lazy Test* co-occurred with *Eager Test*. Listing 1 shows *Eager Test* occurs when a dependency is established between the classes `CSVFormat` and `CSVParser` (lines 989, 995).

```
986 @Test
987 public void testParseNullFileFormat() {
988     assertThrows(NullPointerException.class,
989         () -> CSVParser.parse((File) null, Charset.
                defaultCharset(), CSVFormat.DEFAULT));
990 }
991
992 @Test
993 public void testParseNullPathFormat() {
994     assertThrows(NullPointerException.class,
995         () -> CSVParser.parse((Path) null, Charset.
                defaultCharset(), CSVFormat.DEFAULT));
996 }
```

Listing 1: Lazy Test and Eager Tests from `CSVParserTest` test class (`commons-csv` project) - Existing Test Suite

Also, we found other strong correlations between test smells to each test suite. For Randoop, the test smells *Exception, Catching and Throwing* co-occurred with *Conditional Test Logic*. All tests generated with Randoop contained an optional logging statement and presented a *throwable* structure. Listing 2 tests the same production method as Listing 1 (lines 992-996). Randoop generated a Conditional Test Logic (lines 3,4) and an Exception, Catching, and Throwing (9-15).

In Evosuite, *Exception, Catching, and Throwing* co-occur with *Lazy Test*. *Lazy Test* occurs when 2 or more test methods are testing the method `parse` from the `CSVParser` production class (Listing 3, line 588). While the Existing Test suite deals with a method that might throw an exception without generating a test smell, through `assertThrows` (Listing 1, line 994), Evosuite generates a throwing structure (Listing 3, lines 587-596). Evosuite generates this structure only if there is a possibility of throwing an exception, and Randoop generates this structure to all tests.

Once we have correlations between different types of test smells for each test suite, there is an indication that these test suites may have been influenced by the technique used to developed the tests.

```
1  @Test
2  public void test003() throws Throwable {
3      if (debug)
4         System.out.format("%n%s%n","CSVParser0.test003");
5      java.nio.file.Path path0 = null;
6      java.nio.charset.Charset charset1 = null;
7      org.apache.commons.csv.CSVFormat cSVFormat2 = null;
8      // The following exception was thrown during
           execution in test generation
9      try {
10         org.apache.commons.csv.CSVParser cSVParser3 =
               org.apache.commons.csv.CSVParser.parse(path0
               , charset1, cSVFormat2);
11         org.junit.Assert.fail("Expected exception of
               type java.lang.NullPointerException; message
               : path");
12     } catch (java.lang.NullPointerException e) {
13     // Expected exception.
14     }
15 }
```

Listing 2: Exception, Catching Throwing and Conditional Test from `CSVParserTest` test class (`commons-csv`) - Randoop

```
582 @Test(timeout = 4000)
583 public void test37()  throws Throwable  {
584    Charset charset0 = Charset.defaultCharset();
585    CSVFormat cSVFormat0 = CSVFormat.newFormat('6');
586    // Undeclared exception!
587    try {
588      CSVParser.parse((Path) null, charset0, cSVFormat0);
589      fail("Expecting exception:NullPointerException");
590    } catch(NullPointerException e) {
591      // path
592      verifyException("java.util.Objects", e);
593    }
594 }
```

Listing 3: Exception, Catching Throwing and Lazy Test from `CSVParserTest` test class (`commons-csv`) - Evosuite

It shows opportunities to perform improvements in the techniques used by automated testing tools to produce a test code with improved quality. Evosuite and Randoop could be improved by replacing the throwing structures by `assertThrows`. Besides, Randoop could be improved by creating an aspect to encapsulate the optional logging inserted in all test methods.

5.2 Answering the Research Questions

We summarized the answers for each RQ, as follows:

- RQ_1 We found a significant difference in quality, from the perspective of test smells, between Randoop and Evosuite tools. Evosuite presented a higher quality than Randoop regarding the presence of test smells;
- RQ_2 We found a significant difference in quality, from the point of view of test smells, between the Randoop tests and the Existing Tests. Existing Tests showed a higher quality than Randoop regarding the presence of test smells;
- RQ_3 We found a significant difference in quality, from the point of view of the test smells, between the Evosuite tests and Existing Tests. Existing Tests presented a higher quality than Evosuite regarding the presence of test smells.

6 THREATS TO VALIDITY

Internal Validity. Although `JNose Test` might generate false-positive results, we minimize this threat by reusing the detection rules implemented in tsDetect, which has a high precision for test smells detection.

External Validity. In the experiment, we run a script to randomly select 21 open source projects in the Github repository. Our study is limited to Java projects due to limitations regarding the selected automated testing tools and test smell detection tool. Moreover, we did not set any constraints regarding the projects size or domain as a strategy to reduce bias related to the project characteristics.

Conclusion Validity. To analyze the experiment hypotheses, we paired the datasets two by two and used the Wilcoxon test. However, other insights could be drawn whether we had decided to pair the three datasets together and used other statistical tests. In future work, such a strategy could be explored.

Construction Validity. Our experiment relies on automated tools to generate the test suite and detect the test smells inserted in its code. Due to the number of projects and the test classes generated, we could not perform a manual inspection to verify whether the test code was well-designed.

7 RELATED WORK

Bavota et al. [2] investigates the prevalence and impact of test smells on software projects. They found that: i) test smells are commonly spread across free and industrial software projects; and ii) test smells make it difficult to understand the existing code. Later, they analyzed a wider set of systems [1]. The results indicated that test smells can affect about 86% of the test classes and the understanding of the test code can be reduced to 30% with the presence of test smells. Both studies used tsDetect.

Additionally, Palomba et al. [12] conducted an analysis on the diffusion of test smells in automatically generated unit tests. The results indicated a high diffusion of test smells and their strong positive correlation with characteristics structural elements of the production code. Later, they investigated whether the test smells generation is influenced by code characteristics [8]. The results indicated that test smells generated by random algorithms are not influenced by code characteristics.

In this empirical study, we are concerned with comparing the unit test quality generated with automated testing tools and the existing test of the projects regarding the presence of test smells.

8 CONCLUSION

We carried out a controlled experiment to analyze the quality of tool-generated test code and existing tests in software projects regarding the occurrence of test smells. Our results indicated a significant difference in the quality of the test suites related to the occurrence of test smells. We found that the existing tests had a smaller distribution of test smells compared to the tests generated by tools. Also, the test suite generated by Evosuite had a smaller distribution of test smells compared to the test suite generated by Randoop. It indicates that the automated test suites might be influenced by the techniques used by the automated test tools to develop the tests.

As future work, we aim to replicate this study with larger samples, including data from larger projects and different domains. Thus, we would increase the amount of evidence available on the subject. Furthermore, we could analyze data from other automated test generation tools to investigate whether they present a similar behavior related to the presence of test smells.

REFERENCES

[1] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. Are test smells really harmful? An empirical study. *Empirical Software Engineering* 20, 4 (01 Aug 2015), 1052–1094.

[2] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and David W. Binkley. 2012. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *28th IEEE Int. Conf. on Software Maintenance (ICSM)*. Trento, Italy.

[3] Arie Deursen, Leon M.F. Moonen, A. Bergh, and Gerard Kok. 2001. Refactoring Test Code. In *Refactoring Test Code*. CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands.

[4] Gordon Fraser and Andrea Arcuri. 2013. Whole Test Suite Generation. *IEEE Transactions on Software Engineering* 39, 2 (2013), 276–291.

[5] Gordon Fraser, José Miguel Rojas, and Andrea Arcuri. 2018. Evosuite at the SBST tool competition. In *Evosuite at the SBST tool competition*. Gothenburg, Sweden.

[6] Vahid Garousi, Yasaman Amannejad, and Aysu Betin Can. 2015. Software test-code engineering: A systematic mapping. *Information and Software Technology* 58 (2015), 123 – 147.

[7] Vahid Garousi and Barış Küçük. 2018. Smells in software test code: A survey of knowledge in industry and academia. *Journal of Systems and Software* (2018).

[8] Giovanni Grano, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Harald C. Gall. 2019. Scented since the beginning: On the diffuseness of test smells in automatically generated test code. *Journal of Systems and Software* 156 (2019), 312 – 327.

[9] Michaela Greiler, Arie van Deursen, and Margaret-Anne D. Storey. 2013. Automated Detection of Test Fixture Strategies and Smells. In *IEEE 6th Int. Conf. on Software Testing, Verification and Validation*. 322–331.

[10] Gerard Meszaros, Shaun M. Smith, and Jennitta Andrea. 2003. The Test Automation Manifesto. In *Extreme Programming and Agile Methods - XP/Agile Universe*, Frank Maurer and Don Wells (Eds.). Springer, Berlin, Heidelberg.

[11] Carlos Pacheco and Michael Ernst. 2007. Randoop: Feedback-directed random testing for Java. *Conf. on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, 815–816.

[12] Fabio Palomba, Dario Di Nucci, Annibale Panichella, Rocco Oliveto, and Andrea De Lucia. 2016. On the Diffusion of Test Smells in Automatically Generated Test Code: An Empirical Study. In *IEEE/ACM 9th Inter. Workshop on Search-Based Software Testing (SBST)*. IEEE, Austin, TX, US.

[13] Fabio Palomba and Andy Zaidman. 2019. The smell of fear: on the relation between test smells and flaky tests. *Empirical Software Engineering* 24 (02 2019).

[14] Anthony Peruma, Khalid Almalki, Christian D. Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2019. On the Distribution of Test Smells in Open Source Android Applications: An Exploratory Study. In *Proceedings of the 29th Annual Int. Conf. on Computer Science and Software Engineering (CASCON)*. IBM Corp., Riverton, NJ, USA.

[15] Davide Spadini, Martin Schvarcbacher, Ana-Maria Oprescu, Magiel Bruntink, and Alberto Bacchelli. 2020. Investigating Severity Thresholds for Test Smells. In *In Proceedings of the 17th Int. Conf. on Mining Software Repositories*. Seoul, Republic of Korea.

[16] Tássio Virgínio, Santana Railana, Luana Almeida Martins, Larissa Rocha Soares, Heitor Costa, and Ivan Machado. 2019. On the Influence of Test Smells on Test Coverage. (2019), 467–471.