Test Smells Learning by a Gamification Approach

Anna Rita Fasolino
University of Naples Federico II
Naples, Italy
fasolino@unina.it

Porfirio Tramontana
University of Naples Federico II
Naples, Italy
ptramont@unina.it

Abstract

The presence of test smells related to low-quality test cases is a known factor contributing to problems in maintaining both test suites and production code. The need to avoid and fix test smells is becoming more and more popular in the scientific community, as well as the importance of knowing how to detect and refactor existing test cases. However, these practices are very little considered in academic testing courses, due also to the difficulty of making them attractive to students. This position paper presents an approach for teaching test smells based on gamification. It exploits a tool, TSGame, that provides a serious game where students can familiarize with test smells by practicing with their detection and removal from JUnit test code. TSGame has been implemented as a Web-based application that allows a teacher to assign students test smell detection and refactoring tasks that they have to accomplish in game sessions. Upon completion of the tasks they have the possibility to gain rewards. A preliminary version of the tool has been validated in the context of a Software Testing course at Master degree level, with 37 students that showed the tool effectiveness and usefulness for test smell learning.

CCS Concepts

• **Applied computing → Interactive learning environments**.

Keywords

Software Testing Education, Test Smells, Refactoring, Gamification of Learning Activities

ACM Reference Format:
Anna Rita Fasolino and Porfirio Tramontana. 2024. Test Smells Learning by a Gamification Approach. In *Proceedings of the 3rd ACM International Workshop on Gamification in Software Development, Verification, and Validation (Gamify '24), September 17, 2024, Vienna, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3678869.3685687

1 Introduction

Test smells are poorly designed tests and negatively affect the quality of test suites and production code [5]. Already in 2006 Van Rompaey et al. observed that their presence may worse the maintainability of test suites and production code, or even affect their functionality [5]. This result was recently confirmed by Panichella et al. [11] which found that test smells are capable to capture design issues in test code that reduces its maintainability.

The problem of test smells was formalized for Java test cases implemented in JUnit in 2001 by van Deursen et al. [15], who proposed a canonical catalog consisting of 11 smell types with the associated recommended refactoring. This catalogue was frequently discussed, both in literature and by practitioners taking into account the evolution of testing practices and technologies: hundreds of other possible smells were proposed over time [5, 6].

Most of the work in literature are related to the automatic detection of test smells. Aljedaani et al. [1] have recently catalogued 22 detection tools developed since 2006 (but most of them were realized since 2019). These tools are able to find 85 different types of test smells in 4 different programming languages, but most of the tools are related to Java and JUnit, e.g. TeCReVis [9], PraDeT [4], DARTS [10], JNose Test [16], and tsDetect [12].

Jones in 2001 pointed out that, "there is a basic set of testing skills that every undergraduate should acquire" [8]. The practice of software testing, such as unit testing, is suggested to be integrated into the Computer Science and Software Engineering curricula as part of the educational experience. On the other hand, test smells do not represent a frequent topic in software testing courses. In a recent analysis of the syllabi of 22 software testing courses and other 95 courses including testing topics taught in universities in Belgium, Italy, Spain and Portugal, no references to test smells teaching were found [14]. The unique experience of teaching test smells found in literature appears to be the one of Aniche et al. [2], for which a Learning Goal of their course is to write maintainable test code by avoiding well-known test code smells.

To fill in this gap, we decided to develop a tool that supports Test Smells learning by exploiting a gamification approach. Gamification is a well-know and effective approach for aiding educational processes and it has been used with success in software testing education [7]. Our tool, named TSGame (Test Smells learning Game), has been designed to provide a serious game that allows students to learn different types of test smells, to recognize them in existing JUnit test code, and to remove them by refactoring techniques.

A preliminary case study has been carried out to validate the learning approach supported by the tool. The study has been performed in the context of a Software Testing course offered in a Master Degree in Computer Engineering at the University Federico II of Naples in the autumn of 2023. Its results were encouraging and showed the effectiveness of TSGame in improving the student performance in test smells learning.

The remainder of this position paper is organized as follows. Section 2 presents the TSGame features and Section 3 illustrates how we implemented it. Section 4 describes the case study we executed, while Section 5 reports final conclusions and future works.

Gamify '24, September 17, 2024, Vienna, Austria
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1113-8/24/09
https://doi.org/10.1145/3678869.3685687

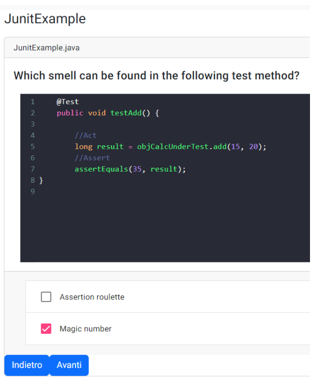
2 TSGame Features

The TSGame tool has been designed to provide a serious game that allows students to learn different types of test smells, to recognize them in existing test code, and to remove them by refactoring techniques. Using the game, the students can be engaged in game sessions where they can:

- practice test smells detection;
- practice test smell refactoring;
- share with other students the experiences made in refactoring existing test smells and receive their comments on the proposed solutions.

The game sessions may have growing levels of difficulty depending on the type of assigned task, on the complexity of the test classes and on the amount of test smells they include.

In a first type of task the app provides the student with several exercises where test cases containing test smells are proposed. The player is invited to detect and classify the observed smell types, by answering a set of questions. After answering the questions, the student is provided a feedback showing how many test smells were correctly detected and classified and possibly receives a reward, when all the test smells have been detected. In addition, a ranking of the better achieved scores (in terms of correctly detected smells) is maintained by the system. Figure 1 shows the Web page offering the Detection game session.


JunitExample

JunitExample.java

Which smell can be found in the following test method?

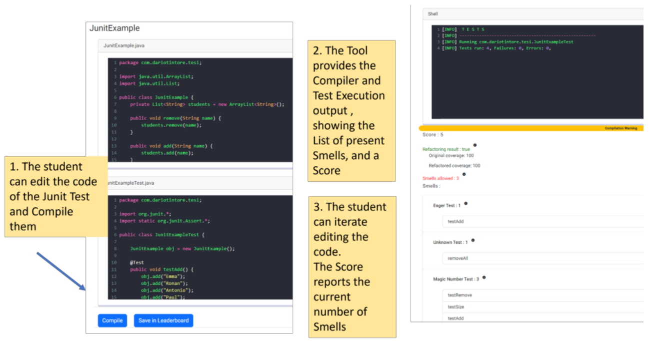
```
    @Test
    public void testAdd() {

        //Act
        long result = objCalcUnderTest.add(15, 20);
        //Assert
        assertEquals(35, result);
}

```

[ ] Assertion roulette

[✓] Magic number

Figure 1: Detection Game Web page

The second type of task to be performed during a Game Session is an advanced one, where students are encouraged to do practice of test smells refactoring. The game provides the student with a set of JUnit test cases including test smells. The player is informed about the number and type of test smells which are present in each test method and is invited to detect all of them and to refactor the test code in order to remove the smells. When he finishes, the application automatically analyses the refactored code and provides the player with a feedback consisting in the number of test smells that have been successfully removed, along with the number of the ones which were not detected and removed. The app provides a reward when all the smells have been correctly removed from the code. In order to discourage the students to remove the test smell just by removing the smelly test case or by oversimplifying it, the system evaluates code coverage and possibly signals the loss of code coverage of refactored test cases as a warning on the user interface. Figure 2 reports the Web page shown to the student during the execution of this learning task.


1. The student can edit the code of the Junit Test and Compile them

2. The Tool provides the Compiler and Test Execution output , showing the List of present Smells, and a Score

3. The student can iterate editing the code. The Score reports the current number of Smells

JunitExample

JunitExample.java

```
package com.dariotintore.tesi;

import java.util.ArrayList;
import java.util.List;

public class JunitExample {
    private List<String> students = new ArrayList<String>();

    public void remove(String name) {
        students.remove(name);
    }

    public void add(String name) {
        students.add(name);
    }
```

JunitExampleTest.java

```
package com.dariotintore.tesi;

import org.junit.*;
import static org.junit.Assert.*;

public class JunitExampleTest {

    JunitExample obj = new JunitExample();

    @Test
    public void testAdd() {
        obj.add("Emma");
        obj.add("Ronan");
        obj.add("Antonio");
        obj.add("Paul");
```

Shell

```
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running com.dariotintore.tesi.JunitExampleTest
[INFO] Tests run: 4, Failures: 0, Errors: 0,
```

Compilation Warning

Score : 5

Refactoring result : true
Original coverage: 100
Refactored coverage: 100

Smells allowed : 3
Smells :

Eager Test : 1
testAdd

Unknown Test : 1
removeAll

Magic Number Test : 3
testRemove
testSize
testAdd

Figure 2: Refactoring Game Web page

As the students complete the assigned game sessions, they accumulates rewards and a leaderboard of players is compiled based on the achieved results.

The game has been also designed to allow the students to share their refactoring solutions and learn from their peer work, by analysing and comparing concrete different solutions. In particular, the player can browse a catalog containing existing solutions of test smells refactoring made by other students and can rate/ comment/ try to improve them.

3 TSGame Implementation

The TSGame learning environment has been developed as a Web application usable from a Web browser. Its architecture is modular, designed to be easily deployed, and includes the following main components:

- The Front-End, providing the presentation logic of the game. It exploits an open source component (Code Mirror[1]) offering many code editing features;
- The Compiler Service, that is responsible for the building and the execution of the test class, the evaluation of the achieved code coverage (by exploiting the JaCoCo library) and the automatic detection of the test smells by means of the tsDetect tool[2];
- The User Service, providing authentication functionality and including a H2 database of registered users;
- The Exercise Service, providing access to a list of exercises, including smelly test classes and quizzes;
- The Leaderboard Service, that maintains a list of all the submitted solutions and of the comments related to each of them.

The teacher can configure the application by creating quizzes and smelly test classes to be refactored by means of json files. The test classes can be uploaded directly on the Exercise Service container or by linking it to github resources.

[1]Code Mirror, available at https://codemirror.net/
[2]tsDetect, available at https://testsmells.org

Table 1: List of Test Smells Automatically Detected by TS-Game exploiting tsDetect

Assertion Roulette	Empty Test	Print Statement
Conditional Test Logic	Exception Catching Throwing	Redundant Assertion
Constructor Initialization	General Fixture	Resource Optimism
Default Test	Ignored Test	Sensitive Equality
Dependent Test	Lazy Test	Sleepy Test
Duplicate Assert	Magic Number Test	Unknown Test
Eager Test	Mystery Guest	Verbose Test

The application can be deployed in two different configurations. In the first one, a Thin-Client solution, all the components are deployed on the Server Side in the context of Docker containers. The Front-End component is an Angular application deployed on a Web server, thus the student can access it via a Web browser in a straightforward way. This configuration is illustrated in Figure 3. In the latter solution, the Front-End and the components of the Compiler Service are deployed in an Electron executable that can be installed and executed locally by the students, whereas the User Service, the Exercise Service and the LeaderBoard Service remain deployed in Docker Containers available via http. This solution implements more features on the client side with respect to the former one, thus it can run faster and is more scalable due to the decentralization of the building operations that may be computationally expensive.

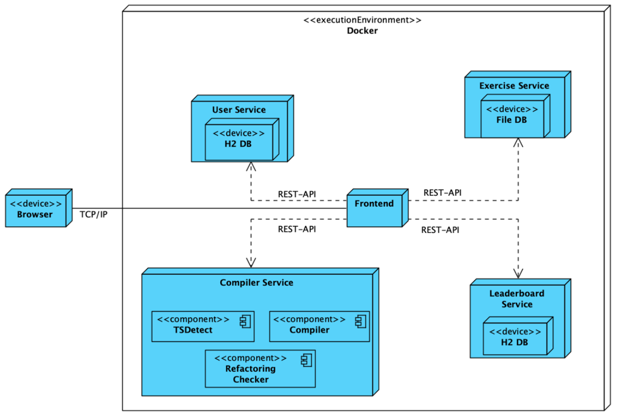


Figure 3: Architecture of the TSGame Learning Environment

Table 1 shows the list of test smells that the environment is able to detect, exploiting the tsDetect tool capabilities.

4 Validation

In this section we present a case study we performed to evaluate the effectiveness of TSGame in aiding the students in test smells learning and to gather the students' perception about usefulness and usability of the tool in the learning experience.

In particular, we wanted to answer the following questions:

RQ1 What is the effectiveness of test smells learning with the support of TSGame?

RQ2 What is the student perception about usability, usefulness, and satisfaction about TSGame, in test smells learning?

4.1 Experimental Procedure

The experimental procedure included the following steps:

- the students were provided a 2-hours traditional lecture presenting test smells basics, a taxonomy of test smells[3], and examples of how to remove them by refactoring;
- the students were presented the TSGame app in a practical 1-hour lecture, where they learned how to use the TSGame features;
- the students were assigned a Game Session homework requiring to solve a realistic test smell refactoring problem on a Java class using TSGame. There was no specific time constraints except that the exercise had to be completed in two weeks;
- after playing the game, the students answered a post- questionnaire designed to collect their perception about the usefulness and the usability of TSGame for learning test smells and to confirm the refactoring results achieved and the actual effort dedicated to complete the task.

4.2 Subjects And Objects

The subjects involved in the study were 37 students of a Software Testing course offered in a Master Degree in Computer Engineering at the University Federico II of Naples. During the course the students had the opportunity to learn several test case design techniques and how to implement test cases with JUnit. The 20 Java classes under test used in the study belong to the SF110 repository originally proposed in [3], whereas the corresponding test classes were produced as part of an experiment carried out to validate the student capability of writing effective test case with Code Defenders [13]. The test classes have a size between 345 and 750 LOCs, with an average of 520 LOCs. The test classes contain a number of test methods ranging between 11 and 83 (38 in average) and a number of smells between 34 and 225 (119 in average). The number of different types of smells in each test class vary between 4 and 8 (6 in average).

4.3 Results

To answer our research questions, we used a questionnaire structured in five sections. In the former one, the students were asked to confirm the values measured by the tool regarding the number of detected and removed test smells, the number of remaining smells, the actual time spent to accomplish the task, and the LOC coverage of the refactored test suite. Three sections of the questionnaire addressed three different attributes, including perceived Usability, Usefulness, and Satisfaction of the students in the learning experience they made by TSGame. For each question, the students provided a level of agreement according to a five-value Likert scale (Strongly agree = 5. Agree =4, Neutral =3, Disagree=2, Strongly Disagree = 1). In the latter section the students could provide comments, suggestions or encountered issues.

As regards RQ1, we observed that 28 out of 37 students (76%) were able to fix all the test smells, while 4 students fixed all the smells except one or two of them, and the 4 remaining students missed to refactor more than two smells. On the other hand, the

[3]https://testsmells.org/pages/testsmellexamples.html

refactoring activities carried out by 28 out of 37 students did not cause reductions in the achieved coverage, whereas in the other 8 cases the achieved code coverage reduced of amounts between 2% and 26%. In general 22 out of 37 students (59%) were able to fix all the test smells without losses in coverage. To accomplish the assigned tasks, the students spent a time ranging between 30 minutes and 12 hours, with an average of about 3.4 hours, depending on the size of the assigned class, the amount of smells to be refactored and the student's motivation. On the basis of the reported data about the refactoring activity, we could answer RQ1 by concluding that *more than half of the students (about 60%) effectively accomplished the test smell refactoring activity, showing the validity of the learning experience supported by the TSGame tool*.

To answer RQ2, we analysed the questionnaire answers. The results to the 31 posed questions and the number of answers for each of the Likert scale values between 1 (strongly disagree) and 5 (strongly agree) are online available[4]. As to the Usability of the tool, most students agreed that the tool was easy to use, understandable, and with a clear UI and interaction mechanisms. They appreciated the user interface and the experience of interacting with it. As to the Usefulness, most of the students agreed in considering the tool useful for the learning experience. As regards the User Satisfaction, the students were mostly satisfied about the tool, in particular with the feedback and the challenging and stimulating experience it provides. The main issues declared by the students were related to the need for more background knowledge on test smells and the teacher support for starting learning with the tool. Another issue regarded the tediousness of the learning experience requiring repetitive refactoring of the same type of test smells, which emerged by the student comments. Some students also declared some inconsistencies in the tsDetect smell detection techniques (particularly for the detection of Magic Number, Exception Catching Throwing and Unknown Test smells). These issues were generally due to the heuristics used by tsDetect which uses tolerance thresholds to reduce false positives. We concluded these thresholds should be better tuned. On the other hand, several students declared difficulties in refactoring the Eager Test smells (8 out of 37 students), since their removal often required a complete restructuring of the test cases.

5 Conclusions And Future Works

In this paper we presented an approach for supporting test smells learning by gamification. We proposed a tool that provides a serious game allowing students to learn different types of test smells, to recognize them in existing test code, and to remove them by refactoring techniques. At the moment, the gamification features implemented in the tool are preliminary, but the case study we performed showed that the idea to use a serious game for engaging students doing practice of test smell detection and refactoring can be useful in a learning process. The case study results also showed us insights about how to improve the tool and the teaching supporting materials.

In future work, we want to add further gamification features to the tool, such as badges, reward mechanisms, challenges with growing levels of complexity, etc., in order to further engage and stimulate the learners in doing practice of test smells. In addition, we intend to design and carry out further evaluations of the benefits of using TSGame in learning contexts, and want to make available our tool and the supporting teaching materials to foster its adoption by other teachers in other Software Testing courses. Finally, we aim to assess whether professionals will value our approach.

[4]Frequencies of answers to the 31 questions, https://zenodo.org/records/13125996

Acknowledgments

This work has been partially funded by ENACTEST (European innovation alliance for testing education), ERASMUS+ Project number 101055874, 2022-2025 and by GATT (GAmification in Testing Teaching), funded by the University of Naples Federico II Research Funding Program (FRA).

References

[1] Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mohamed Wiem Mkaouer, Ali Ouni, Christian D. Newman, Abdullatif Ghallab, and Stephanie Ludi. 2021. Test smell detection tools: A systematic mapping study. In *ACM International Conference Proceeding Series*. 170 – 180. https://doi.org/10.1145/3463274.3463335

[2] Maurício Aniche, Felienne Hermans, and Arie van Deursen. 2019. Pragmatic Software Testing Education. In *Proc. of the ACM Technical Symposium on Computer Science Education*. ACM, 414–420. https://doi.org/10.1145/3287324.3287461

[3] Gordon Fraser and Andrea Arcuri. 2014. A Large-Scale Evaluation of Automated Unit Test Generation Using EvoSuite. *ACM Trans. Softw. Eng. Methodol.* 24, 2, Article 8 (dec 2014), 42 pages. https://doi.org/10.1145/2685612

[4] Alessio Gambi, Jonathan Bell, and Andreas Zeller. 2018. Practical Test Dependency Detection. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. 1–11. https://doi.org/10.1109/ICST.2018.00011

[5] Vahid Garousi, Baris Kucuk, and Michael Felderer. 2019. What we know about smells in software test code. *IEEE Software* 36, 3 (2019), 61 – 73. https://doi.org/10.1109/MS.2018.2875843

[6] Vahid Garousi and Barış Küçük. 2018. Smells in software test code: A survey of knowledge in industry and academia. *Journal of Systems and Software* 138 (2018), 52 – 81. https://doi.org/10.1016/j.jss.2017.12.013

[7] Vahid Garousi, Austen Rainer, Per Lauvås, and Andrea Arcuri. 2020. Software-testing education: A systematic literature mapping. *Journal of Systems and Software* 165 (2020), 110570. https://doi.org/10.1016/j.jss.2020.110570

[8] E.L. Jones. 2001. An experiential approach to incorporating software testing into the computer science curriculum. In *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education*, Vol. 2. F3D–7. https://doi.org/10.1109/FIE.2001.963741

[9] Negar Koochakzadeh and Vahid Garousi. 2010. TeCReVis: A Tool for Test Coverage and Test Redundancy Visualization. In *Testing – Practice and Research Techniques*. Springer. https://doi.org/10.1007/978-3-642-15585-7_12

[10] Stefano Lambiase, Andrea Cupito, Fabiano Pecorelli, Andrea De Lucia, and Fabio Palomba. 2020. Just-in-time test smell detection and refactoring: The DARTS project. In *IEEE International Conference on Program Comprehension*. 441 – 445. https://doi.org/10.1145/3387904.3389296

[11] A. Panichella, S. Panichella, G. Fraser, A. A. Sawant, and V. J. Hellendoorn. 2022. Test smells 20 years later: detectability, validity, and reliability. *Empirical Software Engineering* 27, 7 (2022). https://doi.org/10.1007/s10664-022-10207-5

[12] Anthony Peruma, Khalid Almalki, Christian D. Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2020. TsDetect: An open source test smells detection tool. In - *Proceedings of ESEC/FSE 2020*. 1650 – 1654. https://doi.org/10.1145/3368089.3417921

[13] Jose Miguel Rojas, Thomas D. White, Benjamin S. Clegg, and Gordon Fraser. 2017. Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game. In *Proceedings of ICSE 2017*. 677 – 688. https://doi.org/10.1109/ICSE.2017.68

[14] Porfirio Tramontana, Beatriz Marín, Ana C. R. Paiva, Alexandra Mendes, Tanja E. J. Vos, Domenico Amalfitano, Felix Cammaerts, Monique Snoeck, and Anna Rita Fasolino. 2024. State of the Practice in Software Testing Teaching in Four European Countries. In *17th IEEE International Conference on Software Testing, Verification and Validation (ICST) 2024*. https://doi.org/10.1109/ICST60714.2024.00015

[15] Arie van Deursen, Leon Moonen, Alex van den Bergh, and Gerard Kok. 2001. Refactoring test code. In *Proc. Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering (XP)*.

[16] Tássio Virgínio, Luana Martins, Larissa Rocha, Railana Santana, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. JNose: Java Test Smell Detector *(SBES '20)*. ACM, 564–569. https://doi.org/10.1145/3422392.3422499

Received 2024-07-03; accepted 2024-07-24