

# Insights from the Field: Exploring Students' Perspectives on Bad Unit Testing Practices

Anthony Peruma  
peruma@hawaii.edu  
University of Hawai'i at Mānoa  
Hawai'i, USA

Eman Abdullah AlOmar  
ealomar@stevens.edu  
Stevens Institute of Technology  
New Jersey, USA

Wajdi Aljedaani  
wajdi.aljedaani@unt.edu  
University of North Texas  
Texas, USA

Christian D. Newman  
cnewman@se.rit.edu  
Rochester Institute of Technology  
New York, USA

Mohamed Wiem Mkaouer  
mmkaouer@umich.edu  
University of Michigan-Flint  
Michigan, USA

## Abstract

Educating students about software testing practices is integral to the curricula of many computer science-related courses and typically involves students writing unit tests. Similar to production/source code, students might inadvertently deviate from established unit testing best practices, and introduce problematic code, referred to as test smells, into their test suites. Given the extensive catalog of test smells, it becomes challenging for students to identify test smells in their code, especially for those who lack experience with testing practices. In this experience report, we aim to increase students' awareness of bad unit testing practices, and detail the outcomes of having 184 students from three higher educational institutes utilize an IDE plugin to automatically detect test smells in their code. Our findings show that while students report on the plugin's usefulness in learning about and detecting test smells, they also identify specific test smells that they consider harmless. We anticipate that our findings will support academia in refining course curricula on unit testing and enabling educators to support students with code review strategies of test code.

## CCS Concepts

• **Software and its engineering** → **Software maintenance tools; Maintaining software.**

## Keywords

software engineering, code quality, test smells, unit testing, education, open-source tool, tsdetect

## 1 Introduction

Unit testing is an industry-standard software engineering technique that involves writing small, self-contained tests to verify the correctness of individual code units. Unit testing is the basic building block of software testing, and therefore the Association for Computing Machinery (ACM) recommends the integration of unit testing in Computer Science and Software Engineering curricula [2]. Consequently, seeking educational practices for teaching software testing is becoming the focus of various researchers, resulting in a Software Testing Education Workshop [3].

These education-related studies focus on designing appropriate pedagogical methods of teaching unit test programming and identifying appropriate testing frameworks that can be adopted for

educational purposes. Yet, little is known about how to assess the non-functional quality of students' tests, in order to avoid bad programming practices in their test code. Bad programming practices in the test code, also known as *test smells*, are indicators of potential design problems in the test suite. Test smells are similar to code smells (i.e., bad programming practices in production code). The existence of test smells negatively impacts the efficiency and effectiveness system's testing by increasing the flakiness of test cases [15], and hindering test code readability and understandability [26].

Prior research has correlated the prevalence of test smells with developers' deviation from adopting best practices of writing test code, such as X-Unit [21]. Therefore, it is critical to raise awareness of test smells among students (i.e., early career developers) and to provide them with the tools and training needed to write high-quality test code. In this study, we aim to determine *what type of test smells are perceived by students to be non-harmful to their test code*. To do so, we designed a two-step experiment in which students initially write unit test cases for an input project. Then, we provide students with tsDetect, a test smell detection tool, to identify smelly units in their test suites [24]. Originally, tsDetect is a command line tool. However, for this study, we obtained the source code for the tool from its authors and converted the command line tool to an IntelliJ IDE plugin [22].

Students are instructed to address the smelly test only when they think it is *worth it*, i.e., it is harmful to their test code. Finally, we follow up with a survey to identify the main reasons behind their decisions, particularly when it contradicts the state-of-the-art research findings in this domain.

The results of our study show the existence of a variation between researchers and students on what is considered to be harmful smell types. In particular, the Lazy Test smell was on the top of the list of least harmful smells, followed by Magic Number Test, Eager Test, Empty Test, and Duplicate Assert.

To avoid the propagation of these misperceptions, the findings of our study motivate educators to illustrate the negative aspects of all types of test smells, with a particular focus on how their existence is harmful to test functionality and code comprehension. Also, we show how tsDetect, as an IDE plugin, was found useful by students to improve the overall quality of their test code.

## 2 Related Work

The research literature presents a rich body of work on automatic approaches and tools for identifying test smells in software testing [6]. Investigators have invested substantial effort in identifying and classifying a wide range of test smells [18]. In contrast, some researchers have directed their efforts toward understanding the implications of test smells and formulating effective strategies for their elimination [20]. For instance, Van Bladel and Demeyer [25] introduced a technique to eliminate test smells within the domain of refactoring test code. Similarly, Van Deursen et al. have extensively addressed harmful test smells, providing well-founded techniques for their mitigation [26]. Furthermore, they have introduced innovative conceptual and technical frameworks to evaluate students' coding activities by detecting test smells in their codebases, leading to valuable observations and insights related to test smells. These research pursuits have made notable contributions to the comprehension and management of test smells, significantly impacting software testing practices and overall code quality assessment.

Numerous methods [4, 17] have been devised and explored in evaluating test suites generated by students. In a recent study, Bai et al. [10] examined the student's effectiveness in utilizing a testing checklist on writing the test case, focusing on aspects related to completeness, effectiveness, and maintainability. Buffardi and Aguirre-Ayal [14] conducted an investigation into the origins of unit test smells and their relationship with inaccurate test results, employing a thorough examination of students' testing assignments. The study explored the correlation between the test accuracy of students' assignments and the specific types of test smells present in their codebases. In a complementary study, Bai et al. [11] conducted an experimental investigation to understand students' comprehension of unit testing and the obstacles they encountered. Furthermore, an empirical investigation was conducted by Bavota et al. [13], involving both students and industrial developers, to assess the effect of test smells on software comprehension tasks.

In academic research and educational contexts, scholars and instructors frequently use test suites as an evaluation rate to assess the quality of source code created by the students [8, 19, 27]. However, students' productivity is typically determined by metrics such as lines of code every working hour [7] or any designated coding session [19]. Regrettably, the importance of test smells in the educational setting has been overlooked or underemphasized in the existing literature. Bai et al. [10] conducted a study focusing on the influence of the Assertion Roulette smell on students' productivity and coding behavior, utilizing the Bowling Score Keeper project as an illustrative context. In a similar study, Aljedaani et al. [5] conducted a controlled experiment carried out involving 96 undergraduate computer science students to examine the influence of two prevalent test smells, specifically "Assertion Roulette" and "Eager Test," on the student's proficiency in debugging and troubleshooting test case failures.

## 3 Method

In this section, we provide a detailed explanation of our study's research environment, including information on participants, activities, and data collection. Figure 1 presents a high-level overview of the process, where participants were given a project's source code

and tasked with writing test cases. After completing the task, participants were introduced to the concept of test smells and guided on using an IntelliJ IDEA plugin, `tsDetect`, to detect these smells in their code. Subsequently, they were optionally asked to complete an online questionnaire.

It is important to mention that prior to conducting the study with our student participants, we conducted a pilot run involving a group of three teaching assistants. This pilot study aimed to identify areas for improvement. Valuable feedback from the pilot study allowed us to identify flaws in the assignment source code, and technical challenges related to the plugin, `tsDetect` setup and usage, along with some ambiguities in the lecture notes and questionnaires. Consequently, we fixed source code errors, created video tutorials for the plugin, and addressed ambiguity and duration issues based on the feedback received. The responses to the questionnaire from the pilot run have been excluded from our analysis.

Furthermore, since this study is conducted at multiple locations, lecture content and activity instructions were shared among institutes to minimize the risk of inconsistencies. Finally, to enable replication and extension, lecture artifacts, coding assignments, `tsDetect`, and survey questions are available at [1].

### 3.1 Participants

To ensure a representative sample, our study recruited participants from three higher education institutions located in the United States: Rochester Institute of Technology, Stevens Institute of Technology, and University of North Texas. These participants consisted of both graduate and undergraduate students, the majority of whom were enrolled in a Computing-related major at their respective institute, and were concurrently taking a software engineering-related course taught by one of the authors. Participants were not provided monetary compensation for their involvement in the study. A total of 190 participants were enrolled in our study. However, six participants did not complete the assignment and the questionnaire and were excluded from our analysis. Therefore, in this study, we report on the experience and perception of 184 participants. Table 1 provides a breakdown of participants by institute, degree, and primary major type.

### 3.2 Coding Activity

We constructed three Java programs, and each participant was randomly assigned one program. The programs were intentionally designed to strike a balance between simplicity and functionality. They were made simple enough to ensure participants could comprehend their behavior, yet complex enough to allow for the creation of multiple test cases. A summary description of each program is provided below:

- Automated Teller Machine - Includes functionality to verify a customer, retrieve a customer's balance, deposit money, withdraw money, change a customer's PIN, and transfer money from one account to another.
- Vending Machine - Involves adding and vending items, with necessary checks for item availability and price.
- Calculator - This program contains methods to perform common arithmetic operations on a single or collection of digits

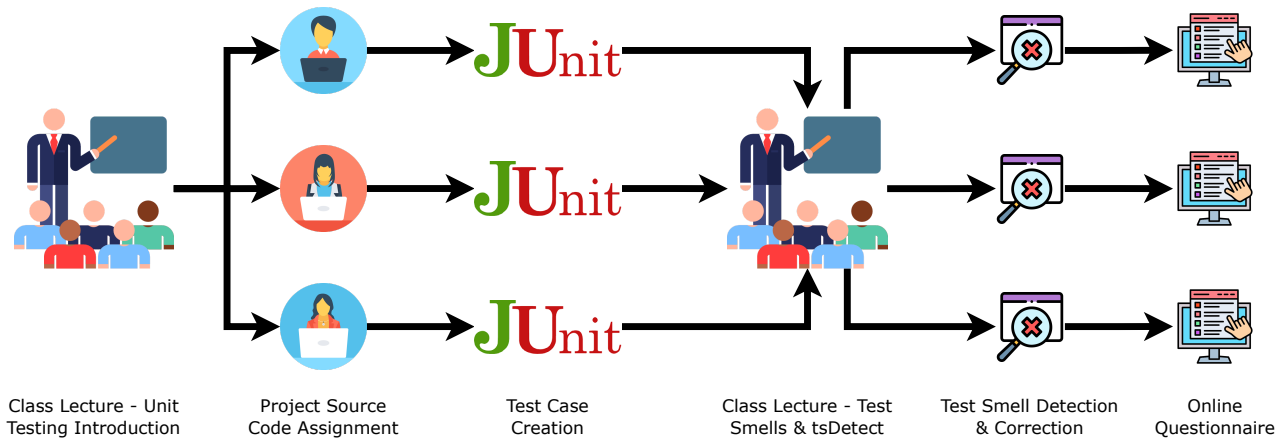


Figure 1: Overview of the setup of our study.

Table 1: A breakdown of the participants in our study by institute and degree type.

Institute Name	Degree Type	Primary Major	Recruited Participants
RIT <sup>1</sup>	Graduate	Data Science	70
		Applied Statistics	1
SIT <sup>2</sup>	Undergraduate	Software Engineering	3
	Graduate	Software Engineering Cybersecurity	34 1
UNT <sup>3</sup>	Undergraduate	Computer Science	70
	Graduate	Computer Science	5
<b>Total Recruited Participants</b>			<b>184</b>

The participants were instructed not to alter the source code. Their task was writing test cases that would achieve a minimum of 85% code coverage. Furthermore, participants were instructed to use IntelliJ IDEA as their IDE and the JUnit 4 testing framework to construct their test suite. Importantly, the concept of test smells was not introduced or discussed with participants during the initial class lecture on unit testing and was not included in the instructions for the coding activity. This ensured that participants approached the task without prior knowledge or bias related to test smells. The participants had to complete the coding activity within five days and submit their work to the instructor.

### 3.3 Test Smell Detection & Correction

After completing the coding activity, the participants had to attend a lecture on test smells. This lecture discussed the different types of test smells and their negative impact on the overall quality of the software system and maintenance activities. Additionally, participants received instructions on how to utilize the tsDetect plugin to identify test smells in their test code.

<sup>1</sup>Rochester Institute of Technology

<sup>2</sup>Stevens Institute of Technology

<sup>3</sup>University of North Texas

In the subsequent activity, participants were tasked with running the tsDetect plugin on their test code, written in the previous activity. The participants were instructed to correct the identified test smell instances, as suggested by the plugin, but only if they agreed with its assessment that the reported smell instance needs to be fixed. It was perfectly acceptable if the participants disagreed with specific instances flagged by tsDetect. As before, participants were prohibited from modifying the source code; only modifications to the test code were permitted. This activity had to be completed within five days and submitted to the instructor.

### 3.4 Online Questionnaire

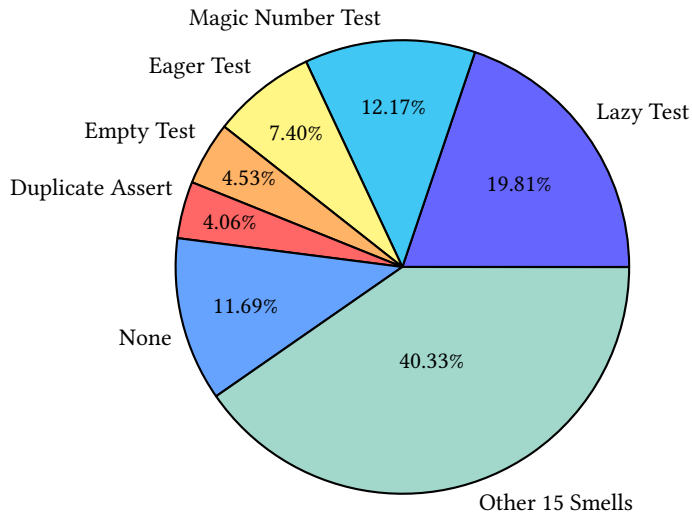
The final activity in the study involved participants completing an online questionnaire after the test smell detection and correction activity. The questionnaire contained 25 questions, incorporating single-choice, multiple-choice, and open-ended questions. Furthermore, for some single- and multi-choice questions, we asked the participants to explain their choice selection using free text. As part of the analysis, the authors performed a thematic analysis [16] of the free text responses to identify recurring themes or patterns within the responses, thereby providing context to better understand their responses. These questions aimed to gather insights on various aspects, including the participants' experience and skill level, perceptions of test smells, and feedback regarding tsDetect.

## 4 Results

In this section, we present the findings of the questionnaire responses. We first provide an overview of the participants' general experience, followed by a detailed analysis of their feedback and experiences with the coding activity in our survey. When reporting our findings for RQ1 and RQ2, we incorporate qualitative data from participants' free-text responses to specific questions. This feedback text is presented as written by the participants.

### 4.1 General Experience

While our survey did not collect personally identifiable information, we did gather information on the participants' programming and industry experience. Concerning software engineering industry



**Figure 2: Distribution of (top 5) test smells considered not harmful by student participants.**

experience, 89 participants (48.37%) reported having no industry experience. Moving forward, approximately 92.39% of the 184 participants had over a year of programming experience, with 50% having between 3 to 5 years of programming background. However, 48.91% of all participants had less than one year of Java experience.

As for testing, only 19 participants (10.33%) had no prior unit testing experience. Additionally, we found that only 25 participants (13.59%) were unfamiliar with JUnit, and 14 participants (7.61%) were not acquainted with the IntelliJ IDEA IDE.

In summary, the above demographic data demonstrates the diversity and representativeness of our sample population; our study’s findings are not biased toward a specific group of students, as we have students with varying degrees of experience.

## 4.2 RQ1: How do students perceive the harmful nature of test smells?

Research on test smells shows that they negatively impact the internal quality of the test suite and hinder the maintainability of the test suite [13]. Nevertheless, it is important to note that its negative impact can vary subjectively, depending on the developer and the coding context [23]. To this extent, this RQ aims at understanding, from a student’s point of view, the types of test smells they consider not harmful. This RQ helps us to better comprehend the subjective nature of test smells and their evaluation, and it helps educators refine curriculum and train future developers effectively.

As part of the questionnaire, we asked participants to indicate and justify what, if any, test smells they consider to be not harmful. Subsequently, the authors examined the participants’ responses and identified the prevalent reasons for participants classifying a specific smell type as non-harmful by adopting a thematic analysis approach [16]. We performed this analysis to determine the main reasons students thought these smells were not harmful.

As shown in Figure 2, Lazy Test smell is the one most participants (around 19.81%) regard as not harmful, followed by Magic Number Test, Eager Test, Empty Test, and Duplicate Assert. Conversely, only

2.15% of participants consider Sensitive Equality, Sleepy Test, Exception Handling, Conditional Test Logic, and Resource Optimism smells as not harmful. Further, approximately 11.69% of participants considered all the smell types they encountered as harmful.

Next, we analyze participants’ reasons for deeming certain smells harmless. Through thematic analysis for each smell type, we learn the rationale behind their views. Below, for each of the top five smells deemed harmless, we elaborate on the common rationales.

**Lazy Test.** This smell occurs when multiple test methods test the same source method. The problem with this practice is that it can cause difficulty maintaining consistency and the possibility of duplicating test cases [12]. However, in our study, participants believe that such smells do not impact maintenance activities as long as test cases pass and they achieve the desired code coverage. Participants created multiple test methods to test a single source method, where each test method verifies a different part/behavior of the source method, leading to the emergence of this smell. Such practices lead to more focused test cases. As one participant commented, “The reason for using multiple test cases is to produce high line coverage. The detection of lazy test makes no sense in that case.”

**Magic Number Test.** This smell occurs when numeric literals are utilized within assert methods, negatively impacting the test’s understandability [23]. Participants consider this smell to be un-harmful since it does not cause test case failures. Participants feel that this smell is not practical as it requires the addition of more lines of code (i.e., constant declarations), which they consider redundant. As stated by a participant, “The magic number smell does not necessarily cause any errors in a program. It is just that we need an extra step to elaborate it better.” Listing 1 shows an example.

```

1 @Test // Magic Number Test smell is present
2 public void NegSqrt() {
3     double result = calculator.squareRoot(-4);
4     assertEquals(Double.MIN_VALUE, result, 0);
5 }
6
7 @Test // Magic Number Test smell is not present
8 public void NegSqrt() {
9     double i = -4;
10    double result = calculator.squareRoot(i);
11    double delta = 0;
12    assertEquals(Double.MIN_VALUE, result, delta);
13 }

```

**Listing 1: An example of a test case with and without a Magic Number Test smell. Participants regarded this as a non-harmful smell and did not find adding more lines of code to support documented numeric literals practicable.**

**Eager Test.** This smell occurs when a single test method checks multiple source methods, negatively impacting comprehension and maintenance activities. Similar to other non-harmful smells, participants’ feedback shows that this smell does not cause failures. The participants did not see the advantage of having separate test methods for the different source method calls. As one participant stated, “I don’t think the Eager Test is harmful because having multiple test cases within a single test method would be the same as having the test cases in separate test methods.” However, while this might help cut down the lines of code, it makes maintenance of the test method more challenging, which is a fact participants did not realize due to their lack of experience.

**Empty Test.** Having a test method without an assertion method gives rise to this smell, as such test methods will always be reported as passing [23]. As any project test codebase grows, empty tests become less visible, misleading testers that a given method in the code is always correct. In this small-scale class assignment, empty tests do not present a major issue resulting in participants not understanding the severity of this smell, as stated by one participant, “I feel like since it is empty, its like having a variable you dont use.”

**Duplicate Assert.** This smell occurs when the same condition is tested multiple times within the same test method, leading to an increase in the length of the test method [23]. Participants did not feel that having duplicates of an assertion method was problematic, as it does not affect the test outcomes. One participant stated, “Duplicate assert is not harmful because it just detects the same assert that we use multiple times.”

A common belief shared by participants when marking a smell type as non-harmful is that since they do not experience test case failures or runtime errors, they consider the smells as not harmful. Participants are primarily concerned about passing test cases and code coverage. As far as participants are concerned, the increase in lines of code due to duplication of testing conditions and source method calls does not impact the maintainability or understandability of the test suite. Furthermore, as test cases pass, some participants regard the detected instances of test smells as false positives (e.g., “I feel it is a false positive, we call the same method for the test code coverage which should not be an issue/ error”). There are a few potential reasons for these outcomes. One reason is that participants may not fully understand the potential negative impacts of these smells, possibly due to a lack of experience (for example, a magic number appearing in a small, easy-to-read program might seem relatively benign). Another reason is that these assignments are designed to introduce concepts and not real-world systems. Consequently, participants might find it challenging to grasp the effort required for comprehending and maintaining the code. Finally, there is also the possibility that some instances detected by `tsDetect` are, in fact, false positives.

Our findings represent education opportunities. Future research, based on these participants' feedback, can help us understand where education on code quality might be lacking. We can use this to determine what materials are essential for us to improve upon/add to curriculums. We can also understand more about how students' awareness of code quality evolves and how we can support this evolution through tools and education.

### 4.3 RQ2: To what extent does an IDE plugin assist students in enhancing their understanding of test smells?

As described in Section 3, this study involves participants using the IDE plugin `tsDetect` to detect test smells. Hence, this RQ examines participants' feedback about the plugin by examining the plugin's ease of use, including the ease of interpreting the output, the detection accuracy of the plugin, and the overall learning experience. Through this analysis, we gain an understanding of the plugin's performance and its impact on users. These insights serve as input to guide improvements and enhancements to ensure that the plugin evolves to meet the user's needs and expectations better.

**4.3.1 Usability:** As shown in Table 2, the majority of the participants found the plugin easy to use, with 18.48% providing a neutral response, while only 10.33% rating the ease of use as difficult. Further, participants had to give a free text explanation for their choice selection. An analysis of the responses shows participants highlighting areas such as installing the plugin, the stability and performance of the plugin, and the plugin's user interface.

Notably, some participants encountered challenges during the installation process or faced environmental issues. However, once successfully installed, participants found the plugin's user interface intuitive and straightforward. For instance, one participant remarked, “Setting up the plugin took time and was a little confusing, but once it was set up the rest was easy.” However, at the same time, we did see a few participants having difficulty performing the activity due to trouble understanding the instructions or their lack of experience in unit testing, and test smells (e.g., “Hard to understand the purpose of the tool, and what the goal was.”). Encouragingly, most participants reported no performance degradation while running the plugin, with one participant commenting, “Getting results very faster and easy to analyze.”

Moving on, approximately 66.3% of participants found it easy to interpret the output, while 14.13% faced difficulty. Once more, analyzing the free text responses, we observe feedback about visualization, documentation, and prior knowledge.

One common area of improvement is the incorporation of comprehensive documentation into the tool to provide details about the detected smells, including recommendations for fixing them. Currently, the plugin provides only the name of the detected smell, which is insufficient, especially for less experienced users, like most participants in this study. For example, one participant remarked, “If I had not used the `testsmells.org` website, I wouldn't have known what to fix just by looking at the code.” On a positive note, participants appreciated the visualizations provided by the plugin, such as the pie chart and tables. For instance, a participant commented, “The pie chart was easy to understand. Additionally, the option of viewing the specific infected class and methods was very helpful.”

**4.3.2 Detection Accuracy:** From Table 2, we observe that a majority of participants, approximately 80.43%, expressed satisfaction with the plugin's detection accuracy, while 3.26% were dissatisfied.

Reviewing the textual answers associated with their choices, we notice feedback associated with false positives and inexperience. Interestingly, the feedback associated with the “Unsure” option shares similarities with those who selected “Dissatisfied.” Common feedback includes participants indicating the occurrence of false positive smells, as in this example: “It seemed like a lot of false positives, so I feel mixed.” However, at the same time, we did encounter instances of participants acknowledging their lack of experience, which hindered their ability to confidently assess the accuracy of the plugin's detections. For instance, one participant mentioned, “I am unsure how accurate it is especially i need to be super familiar.”

**4.3.3 Learning Experience:** While the purpose of `tsDetect` is to help developers improve the internal quality and maintainability of their test cases, it also helps educate developers about the concept and types of test smells. To this extent, we asked participants about the learning effectiveness of the plugin. As shown in Table 2, an overwhelming majority of participants, approximately 94.58%,

**Table 2: Answers to questions examining participants’ feedback about the test smell detection IDE plugin, tsDetect.**

How would you rate the plugin’s ease of use?			Interpreting the output generated by the plugin was:			Rate your level of satisfaction with the detection accuracy of the plugin:			To what extent did the plugin help you better understand test smells?		
Answer Options	Count	Percentage	Answer Options	Count	Percentage	Answer Options	Count	Percentage	Answer Options	Count	Percentage
Very easy to use	70	38.04%	Easy	89	48.37%	Satisfied	119	64.67%	The plugin helped me understand test smells to a moderate extent	59	32.07%
Somewhat easy to use	61	33.15%	Neutral	36	19.57%	Unsured	30	16.3%	The plugin helped me understand test smells to a great extent	59	32.07%
Neutral	34	18.48%	Very easy	33	17.93%	Very satisfied	29	15.76%	The plugin somewhat helped me better understand test smells	34	18.48%
Somewhat difficult to use	17	9.24%	Difficult	22	11.96%	Dissatisfied	6	3.26%	The plugin greatly helped me understand test smells	22	11.96%
Very difficult to use	2	1.09%	Very difficult	4	2.17%	Very dissatisfied	0	0%	The plugin did not help me better understand test smells	10	5.43%

stated that the plugin contributed to their improved understanding of test smells to varying degrees. Only a small minority of 10 participants felt that the plugin did not significantly aid them in better grasping the concept of test smells. Finally, when asked if the plugin’s suggestions helped improve their code, 91.3% of participants responded “Yes”, while the remaining 8.7% responded “No”.

## 5 Reflections

In this section, we reflect on our experience of utilizing an IDE plugin to enhance our teaching of test smells for students. While the students’ overall feedback was positive, we identified areas that could benefit from improvement. Our reflections are not limited to educators planning on using this or similar tools, but also to the research community and tool vendors/builders.

### 5.1 Education

Engaging students in hands-on activities enhances their overall learning experience by reinforcing the concepts covered in lectures. Participants’ feedback shows that tsDetect is a valuable tool for educating students about test smells. As an IDE plugin, tsDetect enables students to seamlessly view both the code and the test smell detection results without switching between multiple applications. For instance, one participant remarked, “The tool was very easy to use and was helpful in understanding test smells within the test cases I had written,” while another stated, “Overall, I just wanted to say that it was really great to learn such a different topics related to testing and it would be really useful in the future as well.”

Educators should emphasize to students that while writing tests to achieve high code coverage is essential, this high coverage should not come at the expense of test code quality. For example, one participant remarked, “The plugin shows lazy test error even though the code line coverage is 100%.” Furthermore, educators should encourage the use of test smell detection tools, such as tsDetect, in addition to source/production code quality tools, such as PMD and linters, to improve the maintainability of test suites.

Due to the wide array of laptop/desktop configurations and environments, it’s likely that some students may face difficulties during the installation and setup of these tools. The time and effort invested by students, educators, and teaching assistants to address these challenges can negatively impact the quality of the students’ learning experience. One solution to mitigate such risks is to provide pre-built virtual machines with running instances of these tools.

### 5.2 Academic Research

When compared to prior work that involved student feedback on test smells, our findings show both similar and contrasting findings. More specifically, Bai et al. [9] reports that the smell Assertion Roulette is not considered harmful by students. However, in our

case, approximately 2.86% of participants consider this smell type as not harmful. In contrast, our work shows similarities with Aljedaani et al. [5], where the authors report that students encounter more challenges working with code exhibiting Assertion Roulette than code having the Eager Test smell. These comparisons highlight the varying student views on test smells’ harmfulness and contribute to broadening our understanding of software testing education.

### 5.3 Tooling

Tool vendors/developers should construct tools to accommodate users with diverse levels of experience. For instance, aside from creating tools that efficiently perform their desired functionality, tool developers should also include help guides/documentation to assist users in using the tool. This is particularly crucial in the context of smells (both code and test smells), owing to the multitude of distinct smell types, and challenges that novice users encounter in comprehending the adverse effects of these specific smells and the techniques for correcting them. For instance, as one participant commented, “It was easy to read the output but if one is not familiar with the type of test smells it was not very useful.”

## 6 Conclusion

Unit testing is an essential practice in software development that involves writing code to test individual components or units of source/production code to verify their correctness and functionality. As such, writing high-quality and maintenance-friendly test code is essential and should be emphasized when teaching students about software testing. In this paper, we report on our experience of utilizing a test smell detection IDE plugin, tsDetect, to complement our teaching of test smells to undergraduate and graduate students in three higher education institutes. Our findings show that, while using tsDetect helps students understand the concept of test smells through hands-on activities, there are also challenges with using this tool. Additionally, we report on specific test smells that students report as not harmful, such as the Eager Test smell.

## References

- [1] [n. d.]. Artifacts. <https://doi.org/10.6084/m9.figshare.23993511.v1>.
- [2] [n. d.]. Curricula Recommendations. <https://www.acm.org/education/curricula-recommendations>. (Accessed on 08/18/2023).
- [3] [n. d.]. TestEd 2023 - The 2nd Testing Education Workshop. <https://testedworkshop.github.io/2023/>. (Accessed on 08/18/2023).
- [4] Kalle Aaltonen, Petri Ihanntola, and Otto Seppälä. 2010. Mutation analysis vs. code coverage in automated assessment of students’ testing skills. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*. 153–160.
- [5] Wajdi Aljedaani, Mohamed Wiem Mkaouer, Anthony Peruma, and Stephanie Ludi. 2023. Do the Test Smells Assertion Roulette and Eager Test Impact Students’ Troubleshooting and Debugging Capabilities?. In *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*. 29–39. <https://doi.org/10.1109/ICSE-SEET58685.2023.00009>

- [6] 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. *Evaluation and Assessment in Software Engineering* (2021), 170–180.
- [7] Prashant Baheti, Laurie Williams, Edward Gehringer, and David Stotts. 2002. Exploring pair programming in distributed object-oriented team projects. In *Educator's Workshop, OOPSLA*. Citeseer, 4–8.
- [8] Gina R Bai, Brian Clee, Nischal Shrestha, Carl Chapman, Cimone Wright, and Kathryn T Stolee. 2019. Exploring tools and strategies used during regular expression composition tasks. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE, 197–208.
- [9] Gina R. Bai, Kai Presler-Marshall, Susan R. Fisk, and Kathryn T. Stolee. 2022. Is Assertion Roulette still a test smell? An experiment from the perspective of testing education. In *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. 1–7. <https://doi.org/10.1109/VL/HCC53370.2022.9833107>
- [10] Gina R Bai, Kai Presler-Marshall, Thomas W Price, and Kathryn T Stolee. 2022. Check It Off: Exploring the Impact of a Checklist Intervention on the Quality of Student-authored Unit Tests. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1*. 276–282.
- [11] Gina R Bai, Justin Smith, and Kathryn T Stolee. 2021. How students unit test: Perceptions, practices, and pitfalls. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 248–254.
- [12] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and David Binkley. 2012. An empirical analysis of the distribution of unit test smells and their impact on software maintenance. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. 56–65. <https://doi.org/10.1109/ICSM.2012.6405253>
- [13] 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 (2015), 1052–1094.
- [14] Kevin Buffardi and Juan Aguirre-Ayala. 2021. Unit test smells and accuracy of software engineering student test suites. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 234–240.
- [15] Bruno Camara, Marco Silva, Andre Endo, and Silvia Vergilio. 2021. On the Use of Test Smells for Prediction of Flaky Tests. In *Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing (Joinville, Brazil) (SAST '21)*. Association for Computing Machinery, New York, NY, USA, 46–54. <https://doi.org/10.1145/3482909.3482916>
- [16] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.
- [17] Stephen H Edwards. 2004. Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. 26–30.
- [18] 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.
- [19] Ayaan M Kazerouni, Stephen H Edwards, and Clifford A Shaffer. 2017. Quantifying incremental development practices and their relationship to procrastination. In *Proceedings of the 2017 ACM Conference on International Computing Education Research*. 191–199.
- [20] Dong Jae Kim, Tse-Hsun Chen, and Jinqiu Yang. 2021. The secret life of test smells—an empirical study on test smell evolution and maintenance. *Empirical Software Engineering* 26 (2021), 1–47.
- [21] Gerard Meszaros. 2007. *xUnit test patterns: Refactoring test code*. Pearson Education.
- [22] Anthony Peruma. [n. d.]. *TSDetect - IntelliJ Plugin*. <https://github.com/TestSmells/TSDetect>
- [23] 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 International Conference on Computer Science and Software Engineering (Toronto, Ontario, Canada) (CASCON '19)*. IBM Corp., USA, 193–202.
- [24] 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 the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Virtual Event, USA) (ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1650–1654. <https://doi.org/10.1145/3368089.3417921>
- [25] Brent Van Bladel and Serge Demeyer. 2017. Test refactoring: a research agenda. In *CEUR workshop proceedings*, Vol. 2070. 1–6.
- [26] Arie Van Deursen, Leon Moonen, Alex Van Den Bergh, and Gerard Kok. 2001. Refactoring test code. In *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*. Citeseer, 92–95.
- [27] Laurie Williams and Richard L Upchurch. 2001. In support of student pair-programming. *ACM Sigcse Bulletin* 33, 1 (2001), 327–331.