Analysis of Test Smell Impact on Test Code Quality

Ismail Cebeci and Tugkan Tuglular
Department of Computer Engineering
Izmir Institute of Technology
Izmir, Turkiye
e-mail: {ismailcebeci | tugkantuglular}@iyte.edu.tr

***Abstract*—Software testing is a crucial component of the software development life-cycle, playing a key role in ensuring the quality and robustness of software products. However, test code, like production code, is susceptible to poor design choices or "test smells", which can compromise its effectiveness and maintainability. This article investigates the prevalence and impact of various test smells across open-source software projects, using advanced detection tools such as JNose and TestSmellDetector. The research highlights that certain test smells, such as "Assertion Roulette," "Magic Number Test," and "Lazy Test," are notably prevalent. The study also examines the co-occurrence of different test smells, providing understanding of how these issues interrelate. Additionally, the article compares the effectiveness of JNose and TestSmellDetector in detecting test smells, providing insights into their strengths and limitations.**

Keywords-Test Smells; Software Testing; Empirical Software Engineering.

I. INTRODUCTION

Software testing is a fundamental part of the software development process and has significant importance in ensuring the quality of software [1]. Test cases exhibit a crucial role in the early detection of software bugs during the software development process.

The quality of the test suite is measured with test coverage analysis where the number of different structural components (functions, instructions, branches, and lines of code) included in the test suite is considered [2]. Nevertheless, despite having a large amount of code coverage, the test code may still contain design choices that are not well-executed, known as test smells. The inclusion of smells in test code has the potential to affect the overall quality of test suites, hence impacting the quality of the production code.

The motivation behind this research stems from the observation that despite the critical role of testing in software development, test smells are often overlooked. Developers and testers may inadvertently introduce these smells into the test code, not through a lack of skill, but due to pressures of deadlines, lack of awareness, or inadequate tool support.

This study contributes to the field by providing empirical data on the detection and impact of test smells across a broad spectrum of open-source software projects. It leverages modern test smell detection tools-JNose [3] and TestSmellDetector [4] tools-to gather insights into the prevalence and co-occurrence of different smells, thereby offering a granular understanding of how these smells interrelate and the potential for cascading effects within the test code. Moreover, for these two tools, a comparison was made on issues such as the differences between them, which test smells are detected better, which device detects more test smells.

The structure of this thesis is organized as follows: Following this introduction, Section II reviews STATE OF THE ART in the field, laying a theoretical foundation for understanding test smells. Section III describes the TOOL INFRASTRUCTURE used in the study, including a detailed examination of the JNose and TestSmellDetector tools. Section IV presents a CASE STUDY analysis, where these tools are applied to a dataset of software projects to identify and analyze test smells. Section V shows the observed RESULTS and Section VI concludes findings and directions for future research.

II. STATE OF THE ART

Modern studies are going in the direction of discovering, defining, and eliminating various categories of code smells, and explaining their origins and influence on the overall program quality. Such studies utilize several approaches, including empirical analysis of open-source software projects and constructing and testing elaborate security tools.

A study by Silva Junior et al. [5], the researchers examined the awareness of test practitioners and the unknowingly incorporation of smells to test code development. A survey is conducted with 60 chosen professionals from different organizations to investigate the frequency and situations in which they encounter smells, particularly 14 types of test smells, which are frequently used in cutting-edge test smell detection tools.

In another study [6] related to the severity of test smells by Campos et al., a set of tests that cause problematic consequences are targeted and the developers' point of view on the issue of test smells is mentioned. By working with its developer participants from six open-source software projects on GitHub, the study aims at characterizing to which extent developers perceive test smells to affect the test code they implement.

In a similar study by Davide Spadini et al. [7], severity thresholds for test smells are investigated. Using 1489 java projects from Apache and Eclipse ecosystems and TestSmellDetector tool, they considered 4 test smells-Assertion Roulette, Eager Test, Verbose Test, and Conditional Test Logic-are higher thresholds than others.

In our study, with extending the total number of test smell types, 21 types of test smells are used, and with using 500 open-source GitHub projects (more than 5000 Java test files), "Magic Number Test" and "Assertion Roulette" are detected as

most frequent test smells. “Empty Test”, “Sleepy Test”, and “Mystery Guest” are 3 of the 5 lowest test smells detected using JNose tool [3] and TestSmellDetector tool [4].

Another study [8] by Michele Tufano et al. presented (i) a survey among 19 developers is carried out to find out how they rated test smells as design issues, and (ii) a huge empirical study based on commit history of 152 open source projects and focused on identifying aspects of both software systems such as when test smells are introduced, how long they last and their relationship with code smells affecting the classes tested.

In our study, to detect test smells, we used two different automated test smell detection tool "JNose Tool” and TestSmellDetector Tool" and the results show that all test files have at least one type of test smell, and to have better test code quality, all test smells should be resolved by developers.

In another study [9] by Soares et al., an innovative way to raise the quality of test code using the JUnit 5 features is described. As part of this research, a mixed-method survey is executed, covering 485 of the most widely used Java open-source projects, finding out that JUnit 5 is used by only a tiny share (15,9%).

In the paper [10] by Annibale Panichella et al., authors scrutinize test smells in the context of automatic test generation. They critically examine whether such smell detection tools work well on sets of tests generated by tool EVOSUITE that test 100 classes of Java programs, in which there are 2340 test cases. Two tools are used in the study. Static detection rules are the first one among the tools suggested by Bavota et al. [11], Grano et al. [12] also use this same tool to detect test smells in test codes. The next tool is TestSmellDetector tool, which is available on GitHub and can be used publicly. The frequency of detection of test smells in Static Detection rules is significantly lower if we compare the findings between Static Detection rules and TestSmellDetector tool. The TestSmellDetector tool demonstrates slightly superior outcomes. Martins et al. [13] also use TestSmellDetector tool to detect test smells and investigate co-occurrence values between different test smells.

Benefiting from previous articles, in addition to similarities, in this article, a research was conducted for the first time using the two mentioned tools and 21 types of test smells with using huge number of projects "around 500", and the results obtained for both tools were compared. Additionally, the co-occurance of the test smells for both tools were compared.

III. TOOL INFRASTRUCTURE

This section mainly explains the tool infrastructure used to detect test smells, in which a detailed analysis about JNose and TestSmellDetector tools are presented. It introduces the working principles of these tools by detailing how they analyze and recognize test smells in test code.

A. JNose Tool

The JNose Test tool enables testers to review the past versions of the software projects and find the test coverage and the test smells that often bother the code quality. This fact enables us to compare various quality metrics of the project over the course of its development process. There are three crucial procedures in the JNose Test operation as shown in Figure 1.

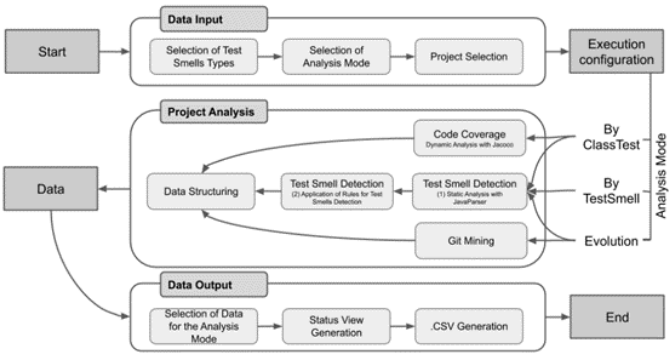

Figure 1. High-level architecture of Jnose tool

- Data Input: This part receives the input set of command parameters for the tool execution, such as test smell types of lists, analysis mode (code coverage, test smells detection and evolution), and the project for analysis.
- Project Analysis: This component presents the analysis of the program by choosing the analysis mode.
- Data Output: By this component, the status of the execution is being rendered and the comma-separated value (CSV) file containing the results of the analysis is generated.

The JNose Tool offers the capability to detect and analyze smells in various ways. Firstly, it can detect smells in a specific test class using the TestClass method, which provides information about the quantity of each type of smell detected in the test class. Secondly, it can detect smells across multiple project versions using the Evolution method, which provides information about the authors and timestamps of the test smell’s insertion in the test code. Lastly, the detection can be used to identify the precise location of a test smell using the TestSmell method, which returns the method location of the smell for the purpose of analyzing the quality of the test code.

In accordance with the GNU General Public License, the JNose Test tool is licensed. The software tool is developed as a Java project and consists of four packages: (i) core, which is responsible for detecting test smells and coverage metrics; (ii) page, which is responsible for displaying web pages and their content; (iii) dto, which includes the classes used in data transfer (Data Transfer Object); (iV) util, which is responsible for identifying tests and production classes and saving results into CSV files.

B. TestSmellDetector Tool

The objective of including TestSmellDetector tool is to offer developers an automated methodology for enhancing the quality of their test suites. The TestSmellDetector tool can

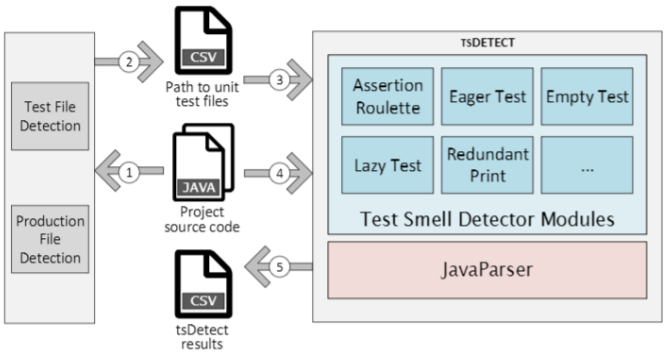

Figure 2. High-level architecture of TestSmellDetector tool

identify 19 smells present in Junit-based unit test files. The TestSmellDetector tool software provides a comprehensive list of detected smells, accompanied by their respective definitions and detection algorithms. The algorithm receives software project source code as input and initially distinguishes between unit test files and production source files.

TestSmellDetector tool is a Java jar file that is open-source and may be used as a command line program. The implementation of TestSmellDetector tool as a self-contained executable file, as opposed to a plugin, eliminates the need for users to own a dedicated Integrated Development Environment (IDE) on their system for the purpose of identifying smells in their test code.

Figure 2 illustrates a comprehensive overview of the architectural design of the TestSmellDetector tool. The project structure is used in 1 and 2 to identify the test and production files. TestSmellDetector tool determines whether test smells are present in the test files in 3 and 4. The test smell detection process findings are saved in 5.

IV. CASE STUDY

To understand test smell impaction of test code quality, we used two different tools which are JNose Tool and TestSmellDetector Tool then we analyzed the result of output files of both tools using projects that they used from Test Smells and Structural Metrics (TSSM) dataset [13].

Figure 3 shows an overview of our study. Mainly in this study, there are four parts to get results to compare and to answer our research questions.

- Project Selection and Preparations: to select projects and preparations to use JNose and TestSmellDetector tools.
- Using TestSmellDetector tool: to follow a way to get results after using TestSmellDetector tool.
- Using JNose tool: to follow a way to get results after using JNose tool.
- Analyzing results: to obtain results to answer research questions.

A. Project Selection

These procedures led to the collection of data from 13,703 open-source Java projects that make up the TSSM dataset.

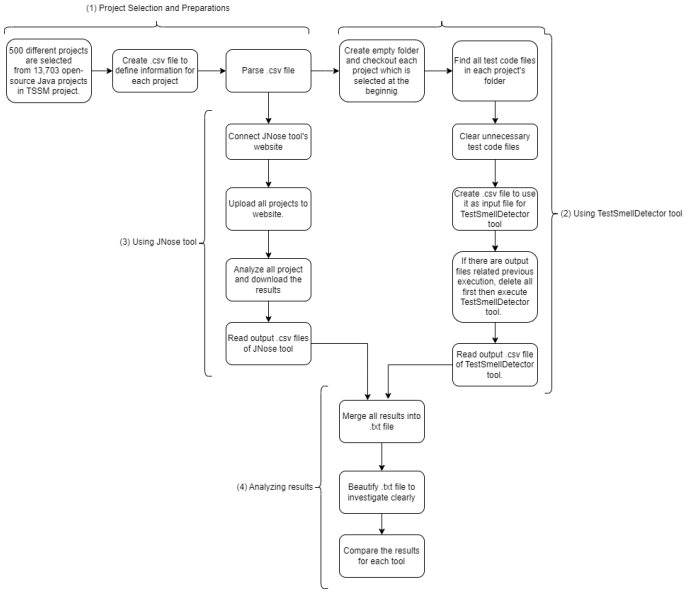

Figure 3. High-level architecture of our study

500 distinct projects are randomly chosen from this collection of open-source Java projects. These projects work with the TestSmellDetector Tool as well as the JNose Tool. Java is among the most common languages today [14] and contains a large number of test codes. This gives us a lot of test code to examine. Additionally, since the two tools used work on Java codes, we decided to work with Java projects. Every project is tested separately at first, and if it works successfully with both tools, it is included in the list.

B. Implementation of Automated Scripts

In this study, four fundamental Python files were implemented. We will do the explanation of these files' roles and functions in detail. Each file has the sole aim of automating and facilitating a different aspect of testing smell analysis process, which in turn makes the identification, comparison, and understanding of test smells in many projects more efficient and accurate.

1) Python File for Preparation of Using Tools: In this file, six functions are created for preparation of using tools. These functions simply do these steps:

- Picking out necessary column names from input CSV file.
- Creating empty folder with using GitHub projects' names.
- Cloning GitHub projects into created empty folders one by one.
- Testing files and their associated source files within GitHub project folders.
- Removing the files, where the lines' sole content are comments.
- Creating a structured CSV file, which is originally named with output.csv and it is specifically designed to meet the given inputs of the TestSmellDetector application.

2) Python File for Using Tools: In this file, six functions are created for using tools. These functions simply do these steps:

- Executing TestSmellDetector Tool with 'output.csv' as a file input.
- Deleting files left over from past executions.
- Reading results clearly going through the created CSV file after executing TestSmellDetector tool. Then, creating txt file after reading CSV file.
- Reading results clearly going through the created CSV files after executing JNose tool. Then, creating txt file after reading all files.
- merging results by two different tools, into one conclusive file titled. After merging, findings might not be next to each other. Therefore, reorganizing findings after merging.

3) Python File for Comparing Results of Each Tools: Comparing the results of different testing methods, which are used in the detection of smells. Co-occurrence Analysis, Ratio Calculation and Comparison and Visualization are done in this file.

4) Python File for Connecting JNose Tool's Website: To accesses the webpage, which is related to Jnose Tool. It automatically inputs GitHub project links into the local server address "http://127.0.0.1:8080" and analyze each project. Then, it downloads results in the CSV format.

V. RESULTS

In this analysis, we compare the effectiveness of two software testing tools, JNose Tool and TestSmellDetector Tool, in identifying several types of test smells within software projects. Test smells play a critical role in ensuring the reliability and efficacy of software testing procedures by identifying any flaws in the test code that could undermine their quality or effectiveness.

The JNose Tool detected 81773 test smells in total using all files. The TestSmellDetector tool detected 89497 test smells in total using all files.

Figure 4 shows a comparative analysis of file affectation by test smells, the total number of files examined alongside those unaffected by test smells as identified by two separate tools: JNose and TestSmellDetector. It is evident that a comprehensive set of 5478 files were subjected to the analysis. JNose Tool identified 1550 files that exhibited no test smells, representing a significant portion of the total, yet still suggesting that many files could contain at least one form of test smell. In contrast, the TestSmellDetector Tool demonstrated a higher identification rate, with 1075 files reported as unaffected. Intriguingly, the bar labeled 'No Affected (Both)' is shown at a value of zero, indicating that there were no files, which both tools concurrently identified as free of test smells.

The data serves as a more encompassing and detailed view of the detection capabilities of both tools as they work across a range of test smells. The fact that different detection rates for various test smells are shown by the two tools indicates a noticeable difference as shown in Figure 5. The

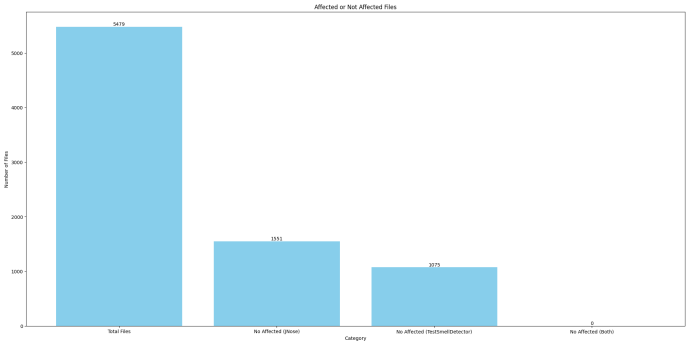

Figure 4. Number of Affected and not Affected Files

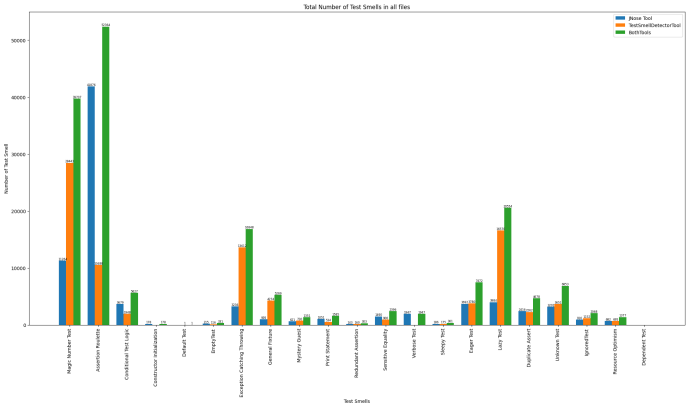

Figure 5. Total Number of Test Smells with using JNose and TestSmellDetector Tools in all files

TestSmellDetector Tool, for instance, is very effective in identifying 'Magic Number Test' smell with 28,443 instances detected entirely outperforming the 11,264 instances detected by the JNose Tool. The pattern of higher detection rates by the TestSmellDetector Tool is also observed in the other types of tests smells like 'Exception Catching Throwing' and 'Lazy Test', which the tool detected 13,612 and 16,570 occurrences, respectively and thus demonstrating its sensitivity towards these particular smells.For 'Assertion Roulette, TestSmellDetector Tool detected 10,488 occurence.

On the other hand, JNose Tool proved to be more effective than TestSmellDetector Tool in discovering the 'Assertion Roulette' instances, which were 41,876 compared to TestSmellDetector Tool, which discovered 10,488 instances as shown in Figure 5. This revelation of the JNose Tool's effectiveness in this case indicates that it can be particularly useful for scenarios where the tests contain multiple non-documented assertions, resulting in unclear test outcomes. In addition, the JNose Tool exhibits greater detection rates for various sorts of test smells, such as the 'Magic Number Test' and 'Lazy Test', with detection rates of 11,264 and 3984 occurrences, respectively. This demonstrates the tool's sensitivity towards these specific smells. JNose tool also performed high detection rates for 'Eager Test' with detection rate of 3692.

This analysis provides the absolute number of files affected

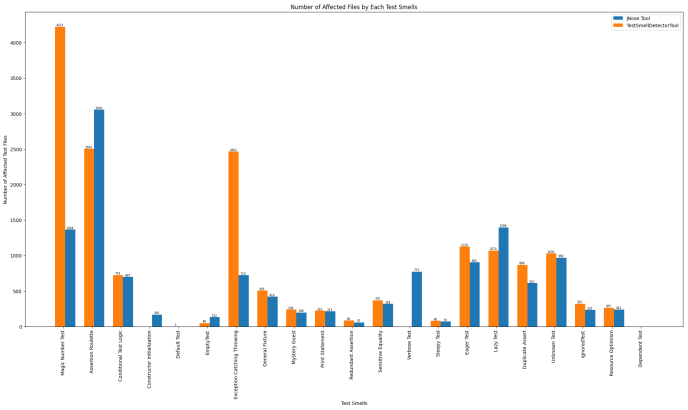

Figure 6. Number of Affected Files by Each Test Smells

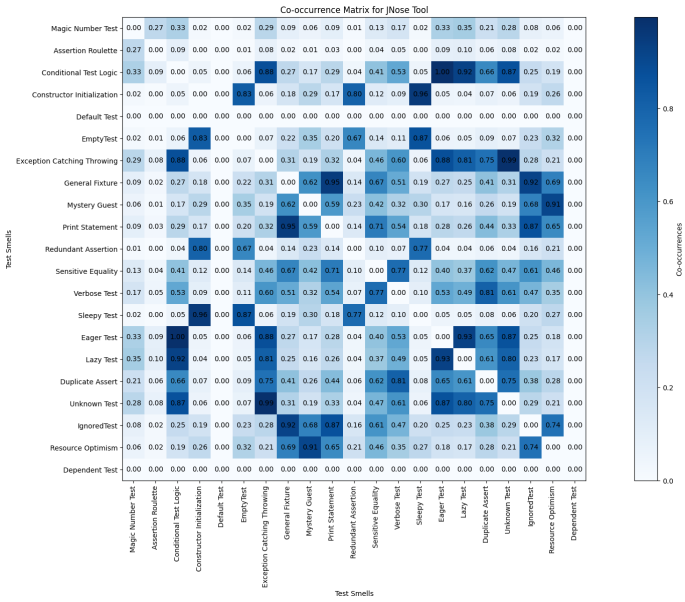

Figure 7. Co-occurrence Matrix for JNose Tool

by each test smell and allows an assessment of the extent of testing and detection of smell testing for both tools across various categories of test smell as shown in Figure 6.

By using the TestSmellDetector tool, highest numbers of affected files by 'Magic Number Test', 'Assertion Roulette', 'Exception Catching Throwing', 'Eager Test', 'Lazy Test', and 'Unknown Test' are detected as 4222, 2503, 2463, 1126, 1070, and 1030. On the other hand, by using the JNose tool, highest numbers of affected files by 'Assertion Roulette', 'Lazy Test', 'Magic Number Test', 'Exception Catching Throwing', 'Unknown Test', and 'Eager Test' are detected as 3056, 1396, 1364, 969, and 905.

The analysis also highlights test smells that are most and least prevalent in the datasets. 'Magic Number Test', 'Assertion Roulette', 'Exception Catching Throwing', 'Eager Test', 'Lazy Test', and 'Unknown Test' are among the most affecting test smells, with both tools identifying a considerable number of affected files. In contrast, 'Constructor Initialization', 'Default Test', and 'Dependent Test' show minimal to no detection across both tools.

The utilization of co-occurrence matrices serves as an analytical cornerstone for uncovering the underlying patterns of test smell interactions within software testing environments. The matrices of The JNose Tool and TestSmellDetector Tool explain these patterns, illustrating both pronounced and negligible relationships among various test smells. In the interest of refining testing strategies, it becomes necessary to research into the specifics of these relationships.

Results for the JNose Tool as shown Figure 7, the one, which stands out the most is a correlation established between 'Conditional Test Logic' and 'Eager Test' with a co-occurrence value of [1.00], indicating a strong likelihood of these issues to arise simultaneously.

Similarly, the pairing of 'Exception Catching Throwing' with 'Unknown Test' and a high co-occurrence rate of [0.99] of using JNose Tool shows a strong correlation.

Next strong correlations are the one observed between 'Sleepy Test' and 'Constructor Initialization', with a co-occurrence value of [0.96] for the JNose Tool.

Conversely for the JNose tool, a pair exposes relationships that are markedly tenuous, as is the case between 'Magic Number Test' and 'Redundant Assertion', with a negligible co-occurrence rate of [0.01]. Another pair exhibiting minimal interdependence comprises 'Mystery Guest' and 'Assertion Roulette' and, 'Empty Test' and 'Assertion Roulette' where the co-occurrence rate stands at [0.01] for both pairs.

Results for the TestSmellDetector Tool as shown in Figure 8, the notable correlation observed in this case is between 'Unknown Test' and 'Eager Test' and their co-occurrence value of [0.97].

The pairing of 'Source Optimism' with 'Mystery Guest' also has a strong co-occurrence rate of [0.95] with using TestSmellDetector Tool.

Conversely, the matrix unveils relationships that are markedly tenuous, as is the case between 'Magic Number Test' and 'Redundant Assertion', 'Magic Number Test' and 'Sleepy Test', 'Assertion Roulette' and 'Empty Test', 'Empty Test' and 'Exception Catching Throwing', 'Empty Test' and 'Lazy Test', so on with a negligible co-occurrence rate of [0.01] with using TestSmellDetector Tool.

VI. CONCLUSION AND FUTURE WORK

Testing is currently considered to be an essential process for improving the quality of software. Unfortunately, past literature has shown that test code can often be of low quality and may contain design flaws, also known as test smells. This paper presented a comparison of the results of the most well-known test smell detector tools (JNose and TestSmellDetector) using 500 distinct open-source GitHub projects. These results give us (i) the number of detection of test smells by each tool, (ii) the number of affected test code files by test smells, and (iii) the co-occurrence rate of detected test smells with the mentioned tools.

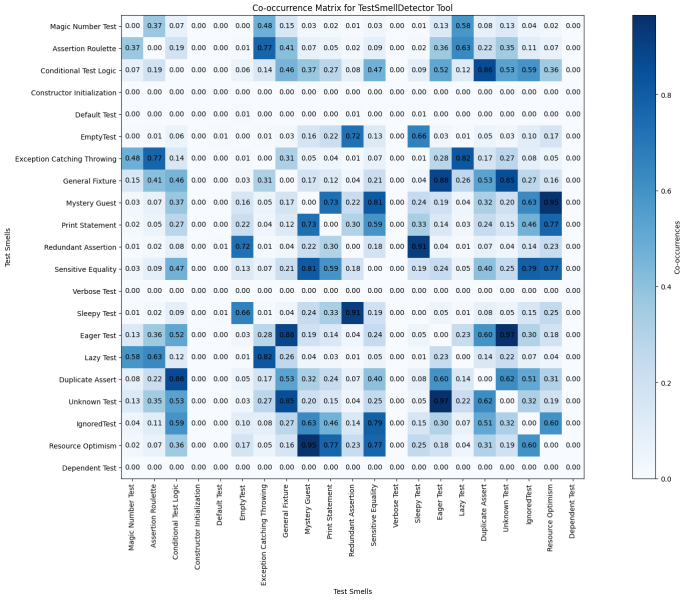

Figure 8. Co-occurrence Matrix for TestSmellDetector Tool

- (i) The 'Assertion Roulette' is the most prevalent smell in the JNose Tool with 41,876 detections. Like 'Assertion Roulette', other common the test smells 'Magic Number Test' with 11264 detections, 'Lazy Test' with 3984 detections, 'Eager Test' with 3692 detections, 'Conditional Test Logic' with 3679 detections, 'Exception Catching Throwing' with 3236 detections, and 'Unknown Test' with 3202 detections. On the other hand, the TestSmellDetector tool has found that the test smells 'Magic Number Test' with 28443 detections and 'Lazy Test' with 16570 detections are the most frequently observed. Furthermore, the test smells 'Exception Catching Throwing' with 13612 detections, 'Assertion Roulette' with 10488 detections, 'General Fixture' with 4274 detections, and 'Eager Test' with 3780 detections are observed in all files.
- (ii) The TestSmellDetector tool detected several files affected by the test smells ('Magic Number Test', 'Assertion Roulette', 'Exception Catching Throwing', 'Eager Test', 'Lazy Test', and 'Unknown Test'), with respective counts of 4222, 2503, 2463, 1126, 1070, and 1030. On the other hand, the JNose tool detected several affected files by 'Assertion Roulette', 'Lazy Test', 'Magic Number Test', 'Exception Catching Throwing', 'Unknown Test', and 'Eager Test' are detected as 3056, 1396, 1364, 969, and 905.
- (iii) The JNose tool showed that there is a strong correlation between the test smells 'Conditional Test Logic' and 'Eager Test', as indicated by a co-occurrence value of [1.00]. Furthermore, the JNose tool reveals a strong relationship between the pairs 'Exception Catching Throwing' and 'Unknown Test', as evidenced by a high co-occurrence rate of [0.99]. In contrast, a high-rated correlation was noticed in this significant relationship between the test smells 'Unknown Test' and 'Eager Test', with a co-occurrence value of [0.97] when using the TestSmellDetector tool. Furthermore, the TestSmellDetector Tool exhibited a combination of 'Source Optimism' and 'Mystery Guest', with a significant co-occurrence rate of [0.95].

As future work, we plan to replicate this study with larger projects, including a more extensive set of test smells. We also plan to implement a new tool to detect test smells and refactor them further. Then, we plan to compare these three tools with larger projects and to show decreased number of detected test smells after refactoring.

References

[1] M. Aberdour, "Achieving quality in open-source software," *IEEE Software*, vol. 24, no. 1, pp. 58–64, 2007. DOI: 10.1109/MS.2007.2.

[2] R. Gopinath, C. Jensen, and A. Groce, "Code coverage for suite evaluation by developers," ICSE 2014, pp. 72–82, 2014. DOI: 10.1145/2568225.2568278.

[3] T. Virgínio *et al.*, "Jnose: Java test smell detector," SBES '20, pp. 564–569, 2020. DOI: 10.1145/3422392.3422499.

[4] A. Peruma *et al.*, "Tsdetect: An open source test smells detection tool," ESEC/FSE 2020, pp. 1650–1654, 2020. DOI: 10.1145/3368089.3417921.

[5] N. S. Junior, L. Rocha, L. A. Martins, and I. Machado, "A survey on test practitioners' awareness of test smells," 2020. arXiv: 2003.05613.

[6] D. Campos, L. Rocha, and I. Machado, "Developers perception on the severity of test smells: An empirical study," 2021. arXiv: 2107.13902.

[7] D. Spadini, M. Schvarcbacher, A.-M. Oprescu, M. Bruntink, and A. Bacchelli, "Investigating severity thresholds for test smells," MSR '20, pp. 311–321, 2020. DOI: 10.1145/3379597.3387453.

[8] M. Tufano *et al.*, "An empirical investigation into the nature of test smells," pp. 4–15, 2016.

[9] E. Soares *et al.*, "Refactoring test smells: A perspective from open-source developers," SAST '20, pp. 50–59, 2020. DOI: 10.1145/3425174.3425212.

[10] A. Panichella, S. Panichella, G. Fraser, A. A. Sawant, and V. J. Hellendoorn, "Revisiting test smells in automatically generated tests: Limitations, pitfalls, and opportunities," pp. 523–533, 2020. DOI: 10.1109/ICSME46990.2020.00056.

[11] G. Bavota, A. Qusef, R. Oliveto, A. D. Lucia, and D. Binkley, "Are test smells really harmful?" *Empirical Software Engineering*, vol. 20, pp. 1052–1094, 2015. DOI: 10.1007/s10664-014-9313-0.

[12] G. Grano, F. Palomba, D. Di Nucci, A. De Lucia, and H. C. Gall, "Scented since the beginning: On the diffuseness of test smells in automatically generated test code," *Journal of Systems and Software*, vol. 156, pp. 312–327, 2019, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.07.016.

[13] L. Martins, H. Costa, and I. Machado, "On the diffusion of test smells and their relationship with test code quality of java projects," *Journal of Software: Evolution and Process*, vol. 36, no. 4, e2532, 2024. DOI: https://doi.org/10.1002/smr.2532. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/smr.2532.

[14] IEEE Spectrum, "Top programming languages 2024," 2024, [Online]. Available: https://spectrum.ieee.org/top-programming-languages-2024 (visited on 08/21/2024).