ANALYSIS OF TEST SMELL IMPACT ON TEST CODE QUALITY

A Thesis Submitted to
the Graduate School of Engineering and Sciences of
İzmir Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE

in Computer Engineering

by
İsmail CEBECİ

June 2024
İZMİR

We approve the thesis of **İsmail CEBECİ**

Examining Committee Members:

_____

Prof. Dr. Tuğkan TUĞLULAR

Department of Computer Engineering, İzmir Institute of Technology

_____

Asst. Prof. Dr. Emrah İNAN

Department of Computer Engineering, İzmir Institute of Technology

_____

Asst. Prof. Dr. Kaan KURTEL

Department of Software Engineering, İzmir University of Economics

12 June 2024

_____

Prof. Dr. Tuğkan TUĞLULAR

Supervisor, Department of Computer Engineering

İzmir Institute of Technology

_____

Prof. Dr. Onur DEMİRÖRS

Head of the Department of

Computer Engineering

_____

Prof. Dr. Mehtap EANES

Dean of the Graduate School

ACKNOWLEDGEMENTS

I would like to thank my thesis advisors Prof. Dr. Tuğkan TUĞLULAR of the Computer Engineering Department at Izmir Institute of Technology. He was always available to help me whenever I needed, and he was always helpful no matter what the issue was.

Also, I would like to thank my wife and my family for supporting me spiritually while researching similar studies, implementing this project, and writing this thesis. Without her support, this study would not have been completed.

ABSTRACT

ANALYSIS OF TEST SMELL IMPACT ON TEST CODE QUALITY

Software testing is a crucial component of the software development lifecycle, 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 thesis investigates the prevalence and impact of various test smells across open-source software projects, using advanced detection tools such as JNose and TestSmellDetector. The study reveals insights into the nature of test smells, their occurrence, and the efficacy of these detection tools.

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. Highest co-occurrence rates are observed between 'Conditional Test Logic' and 'Eager Test' and between ‘Exception Catching Throwing' and 'Unknown Test' using the JNose tool. On the other hand, Highest co-occurrence rates are observed between 'Unknown Test' and 'Eager Test' and 'Source Optimism' and 'Mystery Guest' using TestSmellDetector Tool.

Additionally, the thesis compares the effectiveness of JNose and TestSmellDetector in detecting test smells, providing insights into their strengths and limitations. The analysis of these tools demonstrates their utility in identifying problematic patterns in test code, thereby contributing to better testing practices.

The thesis concludes with mentioning future work, including the development of more advanced detection algorithms and the exploration of refactoring techniques to mitigate the impact of test smells.

ÖZET

TEST KOKUSUNUN TEST KODU KALİTESI ÜZERİNDEKİ ETKİSİNİN ANALİZİ

Test Kokuları, test kodundaki kalıplardır ve mutlaka yanlış olmasa da, test kodunun sürdürülebilirliğini ve etkililiğini engelleyebilecek kötü tasarım seçimlerini önerir. Yazılım geliştirmede, programlamada daha derin sorunlara işaret eden kod kokuları kavramından kaynaklanan test kokuları, benzer şekilde otomatik test komut dosyalarındaki, yazılım test sürecinin güvenilirliğini ve netliğini tehlikeye atabilecek sorunlara işaret eder. Bu tez içinde en çok bilinen 2 araç kullanarak (JNose and TestSmellDetector), GitHub üzerinden erişilen 500 proje incelenmiştir. Belirtilen 500 adet projelerde Java dili kullanılmasına dikkat edildi. İncelenen projelerde bulunan bütün test dosyaları, kullanılan 2 araç için input olarak kullanılmıştır. Araçların çıktıları karşılaştırılarak, toplam kaç adet test kokusu bulunduğu, hangi aracın hangi test kokularını daha iyi tespit ettiğini, en çok hangi test kokularının test dosyalarına etki ettiğini, test kokularının birbiriyle olan ilişkileri ve meydana gelme şıklıkları araştırılmıştır. Sonuç olarak "Assertion Roulette," "Magic Number Test," ve "Lazy Test,” iki araç içinde en yaygın test kokuları olarak elde edilmiştir. Ek olarak, JNose aracı kullanılarak en yüksek birlikte gerçekleşme oranları 'Koşullu Test Mantığı' ile 'Hevesli Test' ve 'İstisna Yakalama Fırlatma' ile 'Bilinmeyen Test' arasında gözlemlenmiştir. Öte yandan, TestSmellDetector Aracı kullanıldığında en yüksek birliktelik oranları 'Bilinmeyen Test' ile 'Hevesli Test' ile 'Kaynak İyimserliği' ve 'Gizemli Misafir' arasında gözlenmiştir. Bu sonuçlar kullanılarak, test dosyaları üzerinde yeniden düzenleme işlemleri için ne tür çalışmalar yapılması gerektiği kolaylıkla belirlenebilir.

TABLE OF CONTENTS

APPENDICES

LIST OF FIGURES

Figure **Page**

Figure **Page**

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Software testing is a fundamental part of the software development process and has significant importance in ensuring the quality of software (Source: Aberdour, 2007) Test cases exhibit a crucial role in the early detection of software bugs during the software development process. They are to consistently test the quality of software and identify any regressions that may occur (Source: Harrold, 2000; Rothermel et al., 2001). Nevertheless, like the production code, the test code may likewise have quality concerns. Previous research has indicated that certain test cases may yield unreliable results, such as flaky tests, because of bugs present in the test code (Source: Vahabzadeh et al., 2015).

Numerous methodologies have been mentioned in the literature (Source: van Deursen et al., 2001) to evaluate the quality of test suites. The code coverage assessment is one of the materials that has been used extensively as the way of assessing the effectiveness of the automated testing. 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 (Source: Gopinath et al., 2014). 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 (Source: Peruma et al., 2020). In addition, tests that are poorly written might be challenging to understand, making it burdensome for testers to maintain the code and identify errors (Source: Bavota et al., 2015; Grano et al., 2019).

In recent years, academics and professionals have started to observe design bugs within the test code (Source: Bavota et al., 2015; Palomba et al., 2017; Spadini et al., 2018; van Deursen et al., 2001). According to Bavota et al. (Source: Bavota et al., 2015), test smells are common in software systems and can impede the understanding and maintenance of programs. According to Palomba et al. (Source: Palomba et al., 2019), certain test smells can lead to the creation of flaky tests, which can have a negative impact

on the quality of the test code. Despite a recent survey indicating that developers possess knowledge about test smells and their potential effects (Source: Garousi et al., 2018), it remains uncertain whether developers actively take care of test smells throughout the development of software and whether fixing these smells would impact the quality of the production code.

When test smells are present in a test suite, empirical research (Source: Bavota et al., 2012) indicates that this might lead to bugs, unreadability, poor maintainability, and poor comprehensibility. Hence, refactoring procedures have been suggested to eliminate these smells (Source: van Deursen et al., 2001). Despite that, a level of imprecision as to developers' understanding of smells and the level of their awareness about them has not already been overcome. In addition, it is not clear where test smells are introduced initially. They might come either during test suite creation or system development and get "smelly" as software evolves. Additionally, it is unknown whether developers undertake any refactoring operations to eliminate test smells. The inclusion of such information is crucial in the development of smell detection rules and the creation of automated detection tools. This is particularly significant in the context of continuous integration processes (Source: Duvall et al., 2007). Automated tools have the capability to identify test smells, resulting in the failure of the build and the subsequent notification of developers regarding the presence of these test smells. The inclusion of test smells in situations where developers have no desire or necessity to maintain them, such as when there is no superior alternative, will enhance the usability of automated smell detection tools. This approach would help prevent recommendation overload (Source: Murphy, 2007) and mitigate the risk of build failures.

1.1. Motivation

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. Therefore, this study not only identifies the most prevalent test smells using

sophisticated detection tools like JNose and TestSmellDetector but also analyzes the impact of these smells on the software testing process and test code quality.

1.2. Major Contributions of the Thesis

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 and TestSmellDetector 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.

1.3. Goal and Research Questions

Purpose of our study is to answer the following research questions (RQs):

- RQ1: What are the most and least frequently detected test smells in test codes? We aim to analyze which test smells are detected mostly and rarely in test code files using JNose and TestSmellDetector tools.
- RQ2: What is the total number of test smells detected by each tool and their distribution in the test code files? By answering this question, we can say which tool works more consistently and effectively.
- RQ3: Is there a considerable co-occurrence between the test smells detected by JNose and TestSmellDetector tools? We aim to identify which test smells co-occurrence in test code files.

1.4. Outline of Thesis

The structure of this thesis is organized as follows: Following this introduction, Chapter 2 reviews related works in the field, laying a theoretical foundation for understanding test smells. Chapter 3 describes the tool infrastructure used in the study, including a detailed examination of the JNose and TestSmellDetector tools. Chapter 4 presents a case study analysis, where these tools are applied to a dataset of software projects to identify and analyze test smells. Finally, Chapter 5 concludes the thesis with a discussion of the findings, implications for software testing practice, and directions for future research.

CHAPTER 2

RELATED WORK

The study of test smells, which are problematic patterns in test code that compromise its maintainability and effectiveness, has attracted significant attention in the software testing community. Here the topic of research becomes a central one because the test smells can lead to added technical debt and less test suite efficiency. 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.

In most of the previous studies, Cutting-edge test smell detection tools (Source: van Deursen et al., 2001; Meszaros et al., 2003; Peruma, 2018) are used to identify issues in test code that can affect its quality and effectiveness. These tools are adaptable to different programming languages and are built on research that confirms their effectiveness. These tools often use complex algorithms to spot problems that simpler tools might miss. They can be integrated directly into software development environments, offering real-time feedback as developers work. Examples include tools like TSDETECT, which are regularly updated to improve their ability to detect problematic patterns in test code.

A study by Silva Junior et al. (Source: Junior et al., 2020), 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. The results indicated that a common pattern is that even though the programmers followed the organizations' standardized practices, it is also very easy for experienced professionals to introduce test smells into their daily programming tasks. In this study, "Conditional Test Logic" and "General Fixture" are detected as the most frequent test smells.

In another study (Source: Campos et al., 2021) 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 most cases, test smells are rated low by developers as they are considered inconsequential. Eight test smells (Assertion Roulette (AR), Empty Test (EpT), Unknown Test (UT), Eager Test (ET), Lazy Test (LT), Constructor Initialization (CI), Sensitive Equality (SE), and Redundant Assertion (RA)) are examined and as a result, LT, SE, EpT, RA are considered as low severity and the AR, ET and UT are considered as high severity.

In a similar study by Davide Spadini et al. (Source: Spadini et al., 2020), severity thresholds for test smells are investigated. Using 1489 java projects from Apache and Eclipse ecosystems and TestSmellDetector tool (Source: Peruma et al., 2020), they considered 4 test smells-Assertion Roulette (AR), Eager Test (ET), Verbose Test (VT), and Conditional Test Logic (CTL)-are higher thresholds than others. Also, they considered developers' points to define a new severity threshold for test smells using their own tool which is provided by TestSmellDetector tool. According to 31 developers' points, “Empty Test (EpT)”, “Sleepy Test (ST)”, and “Mystery Guest (MG)” have the highest priority for code maintainability.

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, "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 (Source: Virgínio et al., 2020) and Test Smell Detector tool (Source: Peruma et al., 2020).

Another study (Source: Tufano et al., 2016) 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. To sum up these lessons show major gaps in present day addressing of test smells in software development. Therefore, better tools, awareness, and practices are needed to be developed to identify and resolve them at every stage of the development

process. In our study, to detect test smells, we used two different automated test smell detection tool "JNose Tool" (Source: Virgínio et al., 2020) and Test Smell Detector Tool" (Source: Peruma et al., 2020) 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 (Source: Soares et al., 2020) 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%). To tackle this, the authors provide new strategies of refactoring featuring JUnit 5 functionalities which facilitate maneuvering against the current test issues namely Assertion Roulette, Test Code Duplication and Conditional Test Logic. The acceptance of developers for the refactored versions of test code could be confirmed by a survey of 212 developers, which shows a strong preference.

In the paper (Source: Panichella et al., 2020) 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. (Source: Bavota et al., 2015). It has been successfully applied in many of the previous work (Source: Bavota et al., 2015; Spadini et al., 2018; Tufano et al., 2016) (Source: Bavota et al., 2015; Spadini et al., 2018; Tufano et al., 2016) to analyze the distribution of test smells by analyzing (manually written tests of) open-source projects. Grano et al. (Source: Grano et al., 2019) also use this same tool to detect test smells in test codes. The next tool is TestSmellDetector tool (Source: Peruma et al., 2020), which is available on GitHub and can be used publicly. Spadini et al. (Source: Spadini et al., 2020) recently calibrated detection rules in TestSmellDetector tool based on developers' perception and classification of test smell level of severity, thus the thresholds that are more harmonious with developers' actual bad test design choices are obtained. The findings indicate that the presence of test smells is frequently observed in a minor yet significant proportion of test suites that are prepared automatically. Nevertheless, 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. (Source: Martins et al., 2024) also use TestSmellDetector tool to detect test smells and investigate co-occurrence values between different test smells.

CHAPTER 3

METHODOLOGY

Chapter 3 of this thesis, titled "Methodology," delves into the technical foundation and procedural steps involved in the study. This chapter 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. Also, methods for project selection, data input, analysis modes and the interpretation of results are explained thus preparing the necessary groundwork for subsequent chapter case study analysis.

3.1. Tool Infrastructure

3.1.1. JNose Tool

The JNose Test tool[1] (Source: Virgínio et al., 2020) 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. Three crucial procedures in the JNose Test operation are given in Figure 3.1.

- (i) 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.
- (ii) Project Analysis: This component presents the analysis of the program by choosing the analysis mode.

- (iii) Data Output: By this component, the status of the execution is being rendered and the .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.

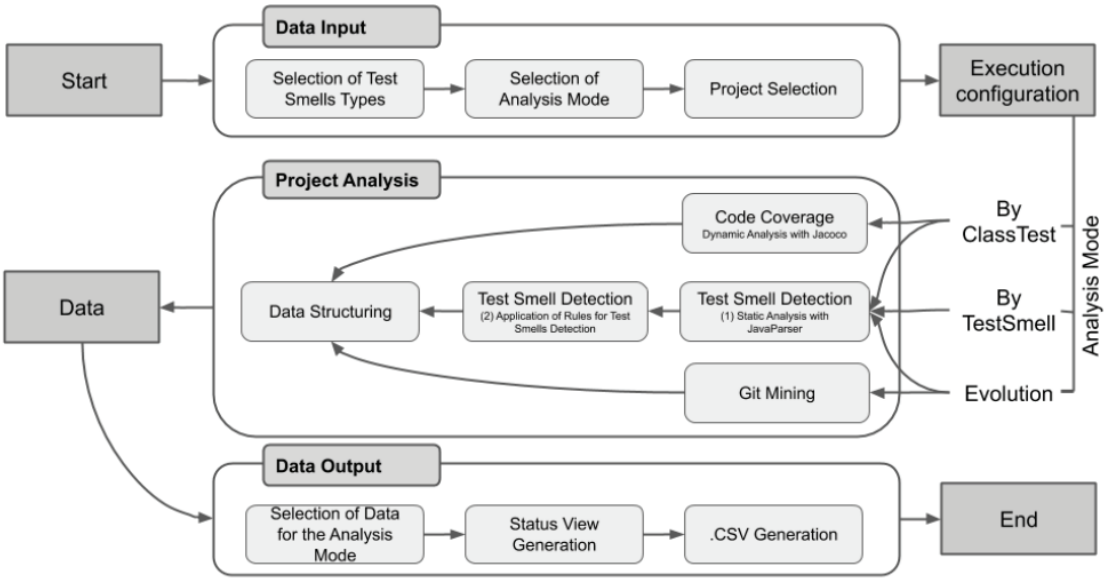


Figure 3.1. Schematic overview of the JNose Test tool and its main features

In accordance with the GNU General Public License, the JNose Test tool (Source: Virgínio et al., 2020) 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); and (iv) util, which is responsible for identifying tests and production classes and saving

results into.csv files. The Project Analysis is implemented by the core package, which is divided into three additional packages, as outlined below:

- Coverage. The use of business rules is essential in the computation of late code coverage. The process involves the identification of test classes associated with a production class, followed by the execution of the JaCoCo library[2]. JaCoCo is a Java-based open-source package designed for the purpose of calculating code coverage. The JaCoCo software conducts a comprehensive analysis of the production code branching (BC), instructions (IC), lines (LC), complexity (CC), and methods (MC) to identify any instances where these components are either overlooked or addressed by the test code (Source: Virgínio et al., 2019).
- TestSmellDetector. The test code undergoes a static analysis using an Abstract Syntax Tree (AST) built by JavaParser[3]. Subsequently, the system pulls pertinent details regarding the code structure to implement the criteria for detecting test smells. Furthermore, it gathers supplementary information pertaining to the position and quantity of test smells. The detection criteria are derived from the TestSmellDetector tool, which encompasses a set of twenty-one rules for the detection of test smells. These rules encompass a range of smells, as documented in references (Source: Virgínio et al., 2019) and (Source: Peruma et al., 2020). Assertion Roulette (AR), Conditional Test Logic (CTL), Constructor Initialization (CI), Default Test (DT), Dependent Test (DpT), Duplicate Assert (DA), Eager Test (ET), Empty Test (EmT), Exception Catching Throwing (ECT), General Fixture (GF), Ignored Test(IgT), Lazy Test(LT),Magic Number Test(MNT), Mystery Guest, Redundant Print, Redundant Assertion (RA), Resource Optimism (RO), Sensitive Equality (SE), Sleepy Test (ST), verbose Test (VT), and Unknown Test (UT).

JNose Test interface is executed within the page package, which is built upon the Apache Wicket[4] framework, which is a Java-based platform designed for the creation of web applications. In addition, HTML5 and CSS3 are employed in the development of the web pages. The Data Input is implemented by this package, as shown in Figure 3.1.

The util package incorporates two utility classes that are responsible for locating the test and production classes, as well as documenting the outcomes in.csv files. A

unique report is provided for each analysis option. The Data Output as shown in Figure 3.1 is implemented by this package.

The dto package encompasses the classes that facilitate the transmission of data between the various layers within the projects. The framework presented in Figure 3.1 encompasses the implementation of Data Input, Project Analysis, and Data Output.
The Apache Maven[5] framework is employed in this project to effectively manage the dependencies, construct, and execute the JNose Test tool. In addition, the JNose Test execution employs parallel processes, wherein a new thread is created for each uploaded project, another thread is created for each test class, and so on. The utilization of parallel processing enables the JNose Test tool to efficiently examine a large volume of projects within a limited time (Source: Virgínio et al., 2019).

3.1.2. TestSmellsDetector Tool

The objective of including TestSmellDetector tool (Source: Peruma et al., 2020) is to offer developers an automated methodology for enhancing the quality of their test suites. The TestSmellDetector tool can identify 19 smells present in Junit-based unit test files, some of which have been identified as troublesome in previous research (Source: Spadini et al., 2018; Tufano et al., 2016; Greiler et al., 2013; Meszaros, 2010). The TestSmellDetector tool software provides a comprehensive list of detected smells, accompanied by their respective definitions and detection algorithms. In brief, the TestSmellDetector tool algorithm examines the test suite to identify specific breaches of the xUnit testing rules (Source: van Deursen et al., 2001; Meszaros, 2010). The algorithm receives software project source code as input and initially distinguishes between unit test files and production source files. It then generates Abstract Syntax Trees (ASTs) for these files, which are then used to syntactically search for preset patterns of inadequate test programming methods using detection criteria. While the tool detects test smells that are applicable to all Java-based systems, there is one smell, known as Default Test, that is unique to Android applications. To adhere to spatial constraints, we present the essential contextual details about the olfactory stimuli substantiated by the TestSmellDetector tool

in previous investigation (Source: Peruma et al., 2020). Furthermore, the website of this project[6] includes real world code snippets that demonstrate each of the possible smell categories. TestSmellDetector tool is designed to be highly extensible, allowing developers to effortlessly adjust the provided criteria and incorporate their own tailored rules as required. Furthermore, while the TestSmellDetector tool presently identifies 19 test smells, it is specifically engineered to seamlessly include novel smell categories.

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 (Source: Peruma et al., 2020) 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. Like PMD and Find Bugs, TestSmellDetector tool is provided as an executable via the command line, enabling its seamless integration with contemporary continuous integration frameworks. This feature also facilitates its utilization in mining software repositories and empirical investigations within the field of software engineering. Furthermore, we incorporate other modules to automate the complete detection operation, in addition to the TestSmellDetector tool detection technique. The detection process is facilitated by these modules, which analyze the input source files to identify unit test files (as well as their corresponding production files) inside the project hierarchy. Figure 3.2 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.

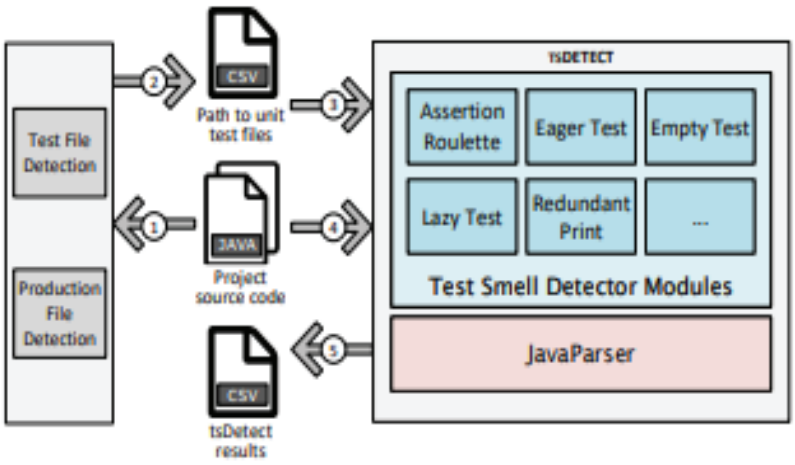


Figure 3.2. High-level architecture of TestSmellDetector tool

3.2. Test Smells

Test smells are indicators of potential problems in test code that might reduce the quality and maintainability of tests. Just like code smells in production code, test smells do not necessarily indicate bugs, but they do suggest that the test code might be poorly structured or difficult to understand and maintain.

Table 3.1 provides a detailed description of various test smells detected by the JNose and TestSmellDetector tools. Each test smell is listed with an acronym, a name, and a brief description. For example, "Assertion Roulette" (AR) refers to tests with multiple assertions where it’s unclear which one caused a failure, while "Constructor Initialization" (CI) refers to test setup logic being placed in the constructor instead of a dedicated setup method. Other smells include "Conditional Test Logic" (CTL), which involves tests containing conditional statements like if-else to handle different scenarios, and "Duplicate Assert" (DA), which denotes multiple assertions in a test checking the same condition redundantly. This table serves as a reference for understanding the specific issues each test smell represents and highlights the breadth of test smells that the tools can detect, which is crucial for maintaining high-quality test code.

Table 3.1. Description of test smells as detected by JNose and TestSmellDetector tools

#	Acronym	Test Smell	Description
1	AR	Assertion Roulette	Tests with multiple assertions where it's unclear which one caused a failure.
2	CI	Constructor Initialization	Test setup logic is placed in the constructor instead of a dedicated setup method.
3	CTL	Conditional Test Logic	Tests containing conditional statements (e.g., if-else) to handle different scenarios.
4	DA	Duplicate Assert	Multiple assertions in a test checking the same condition redundantly.
5	ECT	Exception Catching Throwing	Tests that catch and rethrow exceptions, potentially obscuring the source of failures.
6	EpT	Empty Test	Tests that do not contain any assertions or logic to verify behavior.
7	ET	Eager Test	Tests attempting to verify multiple behaviors or functionalities at once.
8	GF	General Fixture	A shared setup used by many tests, often containing more data or state than necessary for individual tests.
9	IgT	Ignored Test	Tests that are marked to be ignored and do not run during the test suite execution.
10	LT	Lazy Test	Tests that rely heavily on shared fixtures or minimal setup, leading to potential interdependencies.
11	MG	Mystery Guest	Tests that rely on external resources or hidden dependencies not explicitly stated in the test.
12	MNT	Magic Number Test	Tests containing hard-coded values without explanation or context, making them difficult to understand.
13	PS	Print Statement	Tests using print statements for debugging instead of proper assertions.
14	RA	Redundant Assertion	Assertions that are unnecessary because their conditions are already tested elsewhere.
15	RO	Resource Optimism	Tests if external resources (e.g., files, databases) will always be available and in a specific state.
16	SE	Sensitive Equality	Tests that fail due to overly strict equality checks that do not allow for minor variations.
17	ST	Sleepy Test	Tests using sleep statements to wait for conditions instead of proper synchronization methods.
18	UT	Unknown Test	Poorly named or structured tests that do not clearly indicate what functionality they are verifying.

Cont. on next page

Table 3.1 (cont.)

19	VT	Verbose Test	Tests with excessive setup or overly detailed steps, making them hard to read and maintain.
20	DT	Default Test	Auto-generated tests that have not been customized or fully implemented.
21	DepT	Dependent Test	Tests that rely on the results or state of other tests to pass.

In this study, various methods have been used to compare the test smell detection capabilities of the JNose Tool and TestSmellDetector Tool and to uncover the relationships between different test smells. The JNose Tool employs static analysis techniques to examine test code and detect test smells based on specific rules. This tool analyzes the structural characteristics of the code and identifies code segments that match the predefined rules, highlighting potential issues in the tests. Similarly, the TestSmellDetector Tool operates using static analysis methods but offers a more comprehensive analysis by utilizing advanced algorithms and a broader database of test smells. Both tools not only detect test smells but also provide metrics to analyze the impact of these smells on software testing and the tendency of different test smells to occur together. This approach yields detailed information on the prevalence of test smells and their interrelationships, providing valuable insights for improving software testing processes.

Table 3.2 presents a comprehensive comparison of various test smells detected in the study, alongside their occurrence in previous related research. The table includes a list of test smells such as Assertion Roulette, Constructor Initialization, Conditional Test Logic, and others, marking their presence in multiple studies. Each test smell is evaluated across several research works, indicating whether it was identified in those studies with a checkmark (☒). The projects and corresponding test smells are drawn from studies such as "A survey on test practitioners' awareness of test smells," "Developers' perception on the severity of test smells: an empirical study," "Investigating Severity Thresholds for Test Smells," and others. This comparative analysis highlights the consistency and prevalence of various test smells across different research efforts, showing how common these issues are in test code and emphasizing the importance of addressing them to improve software quality.

Table 3.2. Test smells were used and taken from similar studies

Projects / Test Smells	A survey on test practitioners' awareness of test smells[7]	Developers' perception on the severity of test smells: an empirical study[8]	Investigating Severity Thresholds for Test Smells[9]	An empirical investigation into the nature of test smells[10]	Refactoring Test Smells: A Perspective from Open-Source Developers[11]	Revisiting Test Smells in Automatically Generated Tests: Limitations, Pitfalls, and Opportunities[12]	On the diffusion of test smells and their relationship with test code quality of Java projects[13]	In our study
Assertion Roulette	☒	☒	☒	☒	☒	☒	☒	☒
Constructor Initialization	☒	☒	☐	☐	☐	☐	☒	☒
Conditional Test Logic	☒	☐	☒	☐	☒	☐	☒	☒
Duplicate Assert	☐	☐	☐	☐	☐	☐	☒	☒
Exception Catching Throwing	☐	☐	☐	☐	☒	☐	☒	☒
Empty Test	☒	☒	☒	☐	☐	☐	☒	☒
Eager Test	☒	☒	☒	☒	☐	☒	☒	☒
General Fixture	☒	☐	☒	☒	☐	☐	☒	☒
Ignored Test	☐	☐	☒	☐	☐	☐	☒	☒
Lazy Test	☐	☒	☐	☐	☐	☐	☒	☒
Mystery Guest	☒	☐	☒	☒	☐	☒	☒	☒
Magic Number Test	☒	☐	☒	☐	☒	☐	☒	☒
Print Statement	☒	☐	☐	☐	☐	☐	☒	☒
Redundant Assertion	☐	☒	☐	☐	☐	☐	☒	☒
Resource Optimism	☒	☐	☒	☐	☒	☒	☒	☒
Sensitive Equality	☐	☒	☐	☒	☒	☒	☒	☒
Sleepy Test	☐	☐	☒	☐	☐	☐	☒	☒
Unknown Test	☐	☒	☐	☐	☐	☐	☒	☒
Verbose Test	☐	☐	☒	☐	☐	☐	☒	☒
Default Test	☐	☐	☐	☐	☐	☐	☐	☒
Dependent Test	☐	☒	☐	☐	☐	☐	☐	☒

CHAPTER 4

CASE STUDY

To understand test smell impaction of test code quality, we used two different test smell detector tools JNose Tool[14] and Test Smell Detector Tool[15] then we analyzed the result of output files of both tools using projects that they used from TSSM dataset[16].

Figure 4.1 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.

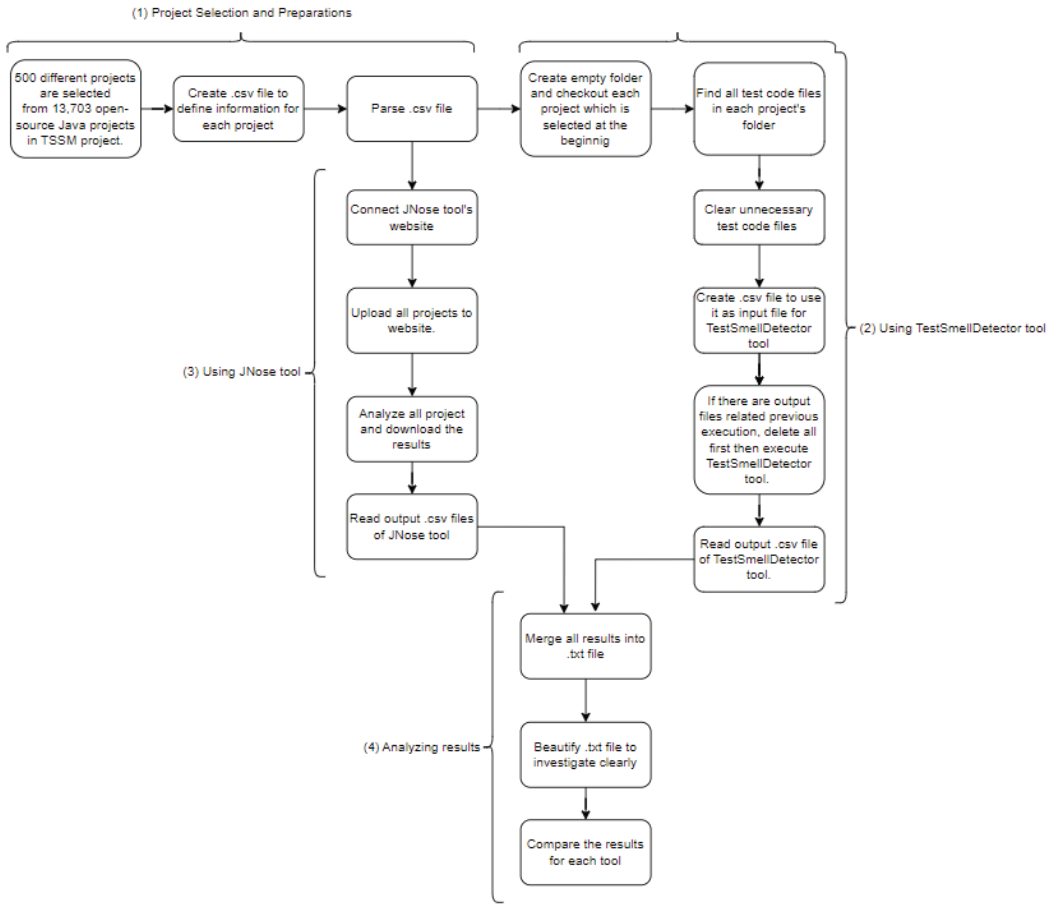


Figure 4.1. High-level architecture of our study

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

4.1. Project Selection

Table A.1 in APPENDIX A displays the projects that we used in our study. The TSSM Dataset utilizes existing data generated by its own process. This process involves the following steps, resulting in the creation of a project.csv file:

- Compilation of Java Projects: Amongst over 8 million available projects, analyzing the huge database GitHub is a long-term job as its size makes the task long. This project focuses on top-5-stars Java projects representing 147,991 items extracted from works of Loriot et al. (Source: Loriot et al., 2022) and Durieux et al. (Source: Durieux et al., 2021) to shorten the process of data gathering. Project_list.txt is the name of the file created and stored in the working environment. The file contains the list of the proposed projects.
- Filtering GitHub Projects: The system utilizes filters to single out the projects from GitHub that are to be used within the project. The process involves sweeping out clones which might result in biased results due to code snippet similarities and settling for open-source projects with licenses that fall under OSI or FSF criteria. Moreover, the representative purpose of the initiative ensures that Java is the first language of choice for the projects that are made. This phase verifies this fact, which in turn provides a confirmation that Java is the prime language, which consequently means the dataset remains intact.

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

selected_project_list.txt, and projects.csv contains the metadata for these projects. Additionally, the TSSM has files that include metrics and test smell data.

500 distinct projects are randomly chosen from this collection of open-source Java projects. These projects work with the Test Smell Detector Tool as well as the JNose Tool. Every project is tested separately at first, and if it works successfully with both tools, it is included in the list.

Following three attributes for each project in APPENDIX A's Table A.1 that was used as an input for both tools in the study:

- Full_name_modified
- clone_url

4.2. 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. All functions' explanations are present on the GitHub project "Master_Thesis_Project" (Source: Cebeci, 2024)

- **preparation_for_using_tools.py**

Regarding improving the usefulness and accessibility of software tools aiming at this project, the preparation_for_using_tools.py script plays a crucial role as well.

def read_csv_and_extract_info(file_path) function: The main purpose of the read_csv_and_extract_info function is to pick out necessary components such as "git_url", "Full_name", and "Full_name_modified" which are needed to conduct the subsequent functional phases of the script. Upon successful parsing, the function meticulously extracts and organizes pertinent data into three distinct lists:

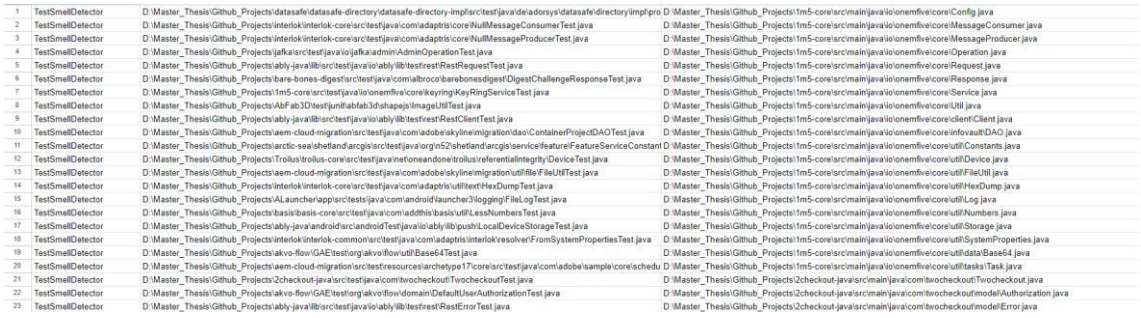
def create_folders(base_path, folder_names) function: The create_folders function is presented with the ability to operate using the "git_project_modified_name" list that is derived from the output of the "read_csv_and_extract_info function." Its purpose is to create smartly utilizing the local filesystem for the GitHub repositories in an organized manner through dedicated folders.

clone_git_projects(base_path, git_clone_url, git_project_modified_name) function: The clone_git_projects function is a cloning process created to manage GITHUB repositories effectively from the project perspectives. Utilizing the "git_clone_url" list that is parsing from read_csv_and_extract_info, it is then initiation of the process of cloning GitHub projects into the directories that was created previously.

find_files_for_test_and_source_codes_by_partial_name(folder_path, partial_name) function: In creating software tools for analyzing projects, we, in particular for this project, find the function, find_files_for_test_and_source_codes_by_partial_name, to be a simple yet important function. This operation is created to simplify discovering test files and their associated source files within several GitHub project folders, which is the main task.

remove_java_test_and_source_files_from_list(test_file_paths,source_file_paths) function: The remove_java_test_and_source_files_from_list routine has an important function related to the data cleansing process in TestSmellDetector tool that is part of the project while focusing on Java test and source files. This is a function that intelligently removes the files, where the lines' sole content are comments.

write_lists_to_csv(constant_name,list1, list2, output_folder, file_name) function: The write_lists_to_csv function represents a part of the data pre-processing with a purpose to run the TestSmellDetector tool application during the project. The main role of this method is the creation of a structured csv file as shown Figure 4.2, which is originally named with output.csv and it is specifically designed to meet the given inputs of the TestSmellDetector application.



1	TestSmellDetector	D:\Master_Thesis\Github_Projects\datasafe\datasafe-directory\datasafe-directory-impl\src\test\java\de\adorsys\datasafe\directory\impl\pro	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\Config.java
2	TestSmellDetector	D:\Master_Thesis\Github_Projects\interlok\interlok-core\src\test\java\com\adaptris\core\NullMessageConsumerTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\MessageConsumer.java
3	TestSmellDetector	D:\Master_Thesis\Github_Projects\interlok\interlok-core\src\test\java\com\adaptris\core\NullMessageProducerTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\MessageProducer.java
4	TestSmellDetector	D:\Master_Thesis\Github_Projects\jafka\src\test\java\io\jafka\admin\AdminOperationTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\Operation.java
5	TestSmellDetector	D:\Master_Thesis\Github_Projects\ably-java\lib\src\test\java\io\ably\lib\test\rest\RestRequestTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\Request.java
6	TestSmellDetector	D:\Master_Thesis\Github_Projects\bare-bones-digest\src\test\java\com\albroco\barebonesdigest\DigestChallengeResponseTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\Response.java
7	TestSmellDetector	D:\Master_Thesis\Github_Projects\1m5-core\src\test\java\io\onemfive\core\keyring\KeyRingServiceTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\Service.java
8	TestSmellDetector	D:\Master_Thesis\Github_Projects\AbFab3D\test\junit\abfab3d\shapejs\ImageUtilTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\Util.java
9	TestSmellDetector	D:\Master_Thesis\Github_Projects\ably-java\lib\src\test\java\io\ably\lib\test\rest\RestClientTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\client\Client.java
10	TestSmellDetector	D:\Master_Thesis\Github_Projects\aem-cloud-migration\src\test\java\com\adobe\skyline\migration\dao\ContainerProjectDAOTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\infovault\DAO.java
11	TestSmellDetector	D:\Master_Thesis\Github_Projects\arctic-sea\shetland\arcgis\src\test\java\org\n52\shetland\arcgis\service\feature\FeatureServiceConstant	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\Constants.java
12	TestSmellDetector	D:\Master_Thesis\Github_Projects\Trollus\trollus-core\src\test\java\net\oneandone\trollus\referentialintegrity\DeviceTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\Device.java
13	TestSmellDetector	D:\Master_Thesis\Github_Projects\aem-cloud-migration\src\test\java\com\adobe\skyline\migration\util\file\FileUtilTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\FileUtil.java
14	TestSmellDetector	D:\Master_Thesis\Github_Projects\interlok\interlok-core\src\test\java\com\adaptris\util\text\HexDumpTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\HexDump.java
15	TestSmellDetector	D:\Master_Thesis\Github_Projects\ALauncher\app\src\tests\java\com\android\launcher3\logging\FileLogTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\Log.java
16	TestSmellDetector	D:\Master_Thesis\Github_Projects\basis\basis-core\src\test\java\com\addthis\basis\util\LessNumbersTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\Numbers.java
17	TestSmellDetector	D:\Master_Thesis\Github_Projects\ably-java\android\src\androidTest\java\io\ably\lib\push\LocalDeviceStorageTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\Storage.java
18	TestSmellDetector	D:\Master_Thesis\Github_Projects\interlok\interlok-common\src\test\java\com\adaptris\interlok\resolver\FromSystemPropertiesTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\SystemProperties.java
19	TestSmellDetector	D:\Master_Thesis\Github_Projects\akvo-flow\GAE\test\org\akvo\flow\util\Base64Test.java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\data\Base64.java
20	TestSmellDetector	D:\Master_Thesis\Github_Projects\aem-cloud-migration\src\test\resources\archetype17\core\src\test\java\com\adobe\sample\core\scheduler	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\io\onemfive\core\util\tasks\Task.java
21	TestSmellDetector	D:\Master_Thesis\Github_Projects\2checkout-java\src\test\java\com\twocheckout\TwocheckoutTest.java	D:\Master_Thesis\Github_Projects\2checkout-java\src\main\java\com\twocheckout\Twocheckout.java
22	TestSmellDetector	D:\Master_Thesis\Github_Projects\akvo-flow\GAE\test\org\akvo\flow\domain\DefaultUserAuthorizationTest.java	D:\Master_Thesis\Github_Projects\2checkout-java\src\main\java\com\twocheckout\model\Authorization.java
23	TestSmellDetector	D:\Master_Thesis\Github_Projects\ably-java\lib\src\test\java\io\ably\lib\test\rest\RestErrorTest.java	D:\Master_Thesis\Github_Projects\2checkout-java\src\main\java\com\twocheckout\model\Error.java

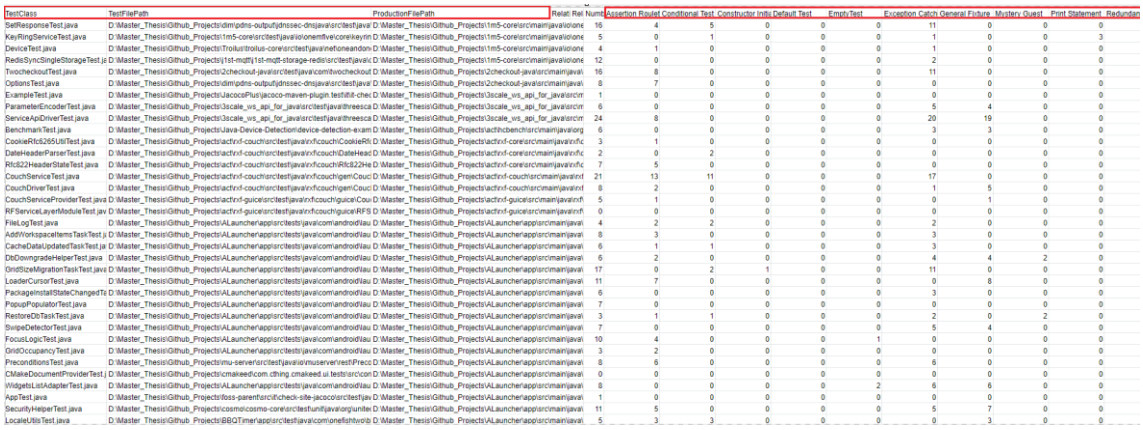
Figure 4.2. Output csv file of write_lists_to_csv function

- **using_test_smell_tools.py**

execute_tool(tool_path, file_name) function: The execute_tool function has been considered an instrument in bolstering software testing quality by utilizing the TestSmellDetector tool within the project, which deals with the issue of test smells. This function can achieve its goal by utilizing a command structure based on the command 'java -jar {tool_path} {file_name}', to perform the TestSmellDetector tool execution with 'output.csv' as a file input. After the tool gets executed, it checks a given set of test codes, among which the smells are searched; it produces a detailed output file, named “output_TestSmellDetection_*.csv”, by default.

delete_files_by_pattern(folder_path, filename_pattern) function: The function delete_files_by_pattern is important for keeping the software's file system as clean and orderly as can be while doing file analysis. It is designed to implement the procedure for deleting files left over from past executions.

read_csv_files_by_pattern(folder_path,filename_pattern) function: The read_csv_files_by_pattern() method is the one that correctly extracting and processes the test smell data from .csv files that have "Output_TestSmellDetection_*.csv" at the end, which are from the TestSmellDetector app. The role of the function has been clearly established for systematically going through the csv file and detecting paths of files under analysis as well as providing the numerical results of selected test smells from a given list of columns as shown in Figure 4.3. After reading, Output_of_TestSmellDetector_Tool.txt file is saved as Figure 4.4.



TestClass	TestFilePath	ProductionFilePath	Relati	Rel Numb	Assertion Roulet	Conditional Test	Constructor Initia	Default Test	EmptyTest	Exception Catch	General Fixture	Mystery Guest	Print Statement	Redundan
SetResponseTest.java	D:\Master_Thesis\Github_Projects\dim\pdns-output\jdnssec-dnsjava\src\test\java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\dione		16	4	5	0	0	0	11	0	0	0	
KeyRingServiceTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\test\java\dione\m5v\core\keyring	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\dione		5	0	1	0	0	0	1	0	0	3	
DeviceTest.java	D:\Master_Thesis\Github_Projects\Trolius\trolius-core\src\test\java\net\oneandon	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\dione		4	1	0	0	0	0	1	0	0	0	
RedisSyncSingleStorageTest.je	D:\Master_Thesis\Github_Projects\jt1st-mqtt\1st-mqtt-storage-redis\src\test\java	D:\Master_Thesis\Github_Projects\1m5-core\src\main\java\dione		12	0	0	0	0	0	2	0	0	0	
TwocheckoutTest.java	D:\Master_Thesis\Github_Projects\2checkout-java\src\test\java\com\twocheckout	D:\Master_Thesis\Github_Projects\2checkout-java\src\main\java\		16	8	0	0	0	0	11	0	0	0	
OptionsTest.java	D:\Master_Thesis\Github_Projects\dim\pdns-output\jdnssec-dnsjava\src\test\java	D:\Master_Thesis\Github_Projects\2checkout-java\src\main\java\		8	7	0	0	0	0	0	0	0	0	
ExampleTest.java	D:\Master_Thesis\Github_Projects\JacocoPlus\jacoco-maven-plugin.test\it-chec	D:\Master_Thesis\Github_Projects\3scale_ws_api_for_java\src\m		1	0	0	0	0	0	0	0	0	0	
ParameterEncoderTest.java	D:\Master_Thesis\Github_Projects\3scale_ws_api_for_java\src\test\java\threescal	D:\Master_Thesis\Github_Projects\3scale_ws_api_for_java\src\m		6	0	0	0	0	0	5	4	0	0	
ServiceApiDriverTest.java	D:\Master_Thesis\Github_Projects\3scale_ws_api_for_java\src\test\java\threescal	D:\Master_Thesis\Github_Projects\3scale_ws_api_for_java\src\m		24	8	0	0	0	0	20	19	0	0	
BenchmarkTest.java	D:\Master_Thesis\Github_Projects\Java-Device-Detection\device-detection-exam	D:\Master_Thesis\Github_Projects\actlhcbench\src\main\java\org		6	0	0	0	0	0	3	3	0	0	
CookieRfc6265UtilTest.java	D:\Master_Thesis\Github_Projects\actrx\f-couch\src\test\java\rx\fcouch\CookieRf	D:\Master_Thesis\Github_Projects\actrx\f-core\src\main\java\rx\fc		3	1	0	0	0	0	0	0	0	0	
DateHeaderParserTest.java	D:\Master_Thesis\Github_Projects\actrx\f-couch\src\test\java\rx\fcouch\DateHead	D:\Master_Thesis\Github_Projects\actrx\f-core\src\main\java\rx\fc		2	0	2	0	0	0	0	0	0	0	
Rfc822HeaderStateTest.java	D:\Master_Thesis\Github_Projects\actrx\f-couch\src\test\java\rx\fcouch\Rfc822Hea	D:\Master_Thesis\Github_Projects\actrx\f-core\src\main\java\rx\fc		7	5	0	0	0	0	0	0	0	0	
CouchServiceTest.java	D:\Master_Thesis\Github_Projects\actrx\f-couch\src\test\java\rx\fcouch\gen\Couc	D:\Master_Thesis\Github_Projects\actrx\f-couch\src\main\java\rx\f		21	13	11	0	0	0	17	0	0	0	
CouchDriverTest.java	D:\Master_Thesis\Github_Projects\actrx\f-couch\src\test\java\rx\fcouch\gen\Couc	D:\Master_Thesis\Github_Projects\actrx\f-couch\src\main\java\rx\f		8	2	0	0	0	0	1	5	0	0	
CouchServiceProviderTest.java	D:\Master_Thesis\Github_Projects\actrx\f-guice\src\test\java\rx\fcouch\guice\Cou	D:\Master_Thesis\Github_Projects\actrx\f-guice\src\main\java\rx\f		5	1	0	0	0	0	0	1	0	0	
RFServiceLayerModuleTest.jav	D:\Master_Thesis\Github_Projects\actrx\f-guice\src\test\java\rx\fcouch\guice\RFS	D:\Master_Thesis\Github_Projects\actrx\f-guice\src\main\java\rx\f		0	0	0	0	0	0	0	0	0	0	
FileLogTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		4	2	2	0	0	0	2	0	0	0	
AddWorkspaceItemsTaskTest.j	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		8	3	0	0	0	0	3	0	0	0	
CacheDataUpdatedTaskTest.ja	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		6	1	1	0	0	0	3	0	0	0	
DbDowngradeHelperTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		6	2	0	0	0	0	4	4	2	0	
GridSizeMigrationTaskTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		17	0	2	1	0	0	11	0	0	0	
LoaderCursorTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		11	7	0	0	0	0	0	8	0	0	
PackageInstallStateChangedTa	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		6	0	0	0	0	0	3	0	0	0	
PopupPopulatorTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		7	0	0	0	0	0	0	0	0	0	
RestoreDbTaskTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		3	1	1	0	0	0	2	0	2	0	
SwipeDetectorTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		7	0	0	0	0	0	5	4	0	0	
FocusLogicTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		10	4	0	0	0	1	0	0	0	0	
GridOccupancyTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		3	2	0	0	0	0	0	0	0	0	
PreconditionsTest.java	D:\Master_Thesis\Github_Projects\imu-server\src\test\java\io\muserver\rest\Precc	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		8	6	0	0	0	0	6	0	0	0	
CMakeDocumentProviderTest.j	D:\Master_Thesis\Github_Projects\cmakeed\com.cthing.cmakeed.ui.test\src\com	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		0	0	0	0	0	0	0	0	0	0	
WidgetsListAdapterTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test\java\com\android\lau	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		8	0	0	0	0	2	6	6	0	0	
AppTest.java	D:\Master_Thesis\Github_Projects\floss-parent\src\it\check-site-jacoco\src\test\jav	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		1	0	0	0	0	0	0	0	0	0	
SecurityHelperTest.java	D:\Master_Thesis\Github_Projects\cosmo\cosmo-core\src\test\unit\java\org\unitec	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		11	5	0	0	0	0	5	7	0	0	
LocaleUtilsTest.java	D:\Master_Thesis\Github_Projects\BBQTimer\app\src\test\java\com\oneflesh\wob	D:\Master_Thesis\Github_Projects\ALauncher\app\src\main\java\		5	3	3	0	0	0	0	3	0	0	

Figure 4.3. Elements of columns_to_read list

```
ApiTest.java:
  TestSmellDetectorTool: Assertion Roulette: 21
  TestSmellDetectorTool: Conditional Test Logic: 0
  TestSmellDetectorTool: Constructor_Initialization: 0
  TestSmellDetectorTool: Default Test: 0
  TestSmellDetectorTool: EmptyTest: 0
  TestSmellDetectorTool: Exception Catching Throwing: 20
  TestSmellDetectorTool: General Fixture: 1
  TestSmellDetectorTool: Mystery Guest: 0
  TestSmellDetectorTool: Print Statement: 0
  TestSmellDetectorTool: Redundant Assertion: 0
  TestSmellDetectorTool: Sensitive Equality: 1
  TestSmellDetectorTool: Verbose Test: 0
  TestSmellDetectorTool: Sleepy Test: 0
  TestSmellDetectorTool: Eager Test: 0
  TestSmellDetectorTool: Lazy Test: 0
  TestSmellDetectorTool: Duplicate Assert: 0
  TestSmellDetectorTool: Unknown Test: 0
  TestSmellDetectorTool: IgnoredTest: 0
  TestSmellDetectorTool: Resource Optimism: 0
  TestSmellDetectorTool: Magic Number Test: 21
  TestSmellDetectorTool: Dependent Test: 0
MainActivityTest.java:
  TestSmellDetectorTool: Assertion Roulette: 1
  TestSmellDetectorTool: Conditional Test Logic: 1
  TestSmellDetectorTool: Constructor_Initialization: 0
  TestSmellDetectorTool: Default Test: 0
  TestSmellDetectorTool: EmptyTest: 0
  TestSmellDetectorTool: Exception Catching Throwing: 2
  TestSmellDetectorTool: General Fixture: 0
  TestSmellDetectorTool: Mystery Guest: 0
  TestSmellDetectorTool: Print Statement: 0
  TestSmellDetectorTool: Redundant Assertion: 0
  TestSmellDetectorTool: Sensitive Equality: 0
  TestSmellDetectorTool: Verbose Test: 0
  TestSmellDetectorTool: Sleepy Test: 0
  TestSmellDetectorTool: Eager Test: 0
  TestSmellDetectorTool: Lazy Test: 0
  TestSmellDetectorTool: Duplicate Assert: 0
```

Figure 4.4. Part of contents of Output_of_TestSmellDetector_Tool.txt

read_csv_for_Jnose_tool(input_folder_path, output_folder_path) function: The read_csv_for_Jnose_tool function is intended to parse csv files output by JNose Tool (filenames follows a pattern "{project_name}_result_byclasstest_testsmells.csv") as shown in Figure 4.5. A particular list, which is named columns_to_read, will define the data parsing procedure's focus. The list will be called 'name' and 'testSmellName', marking the specific parts of the JNose Tool output that will be carried out the process of parsing and analysis.

Accessing each csv file, the function parses the data related to the quantity of each type of test smell present in the files. This extraction process is implemented dynamically because of accommodating changing structures and contents of the csv files corresponding to different projects. Once the relevant data is gathered, the function proceeds to compile these findings into a text document, and saved as "{project_name}_Output.txt" as shown in Figure 4.6 within a designated output folder.

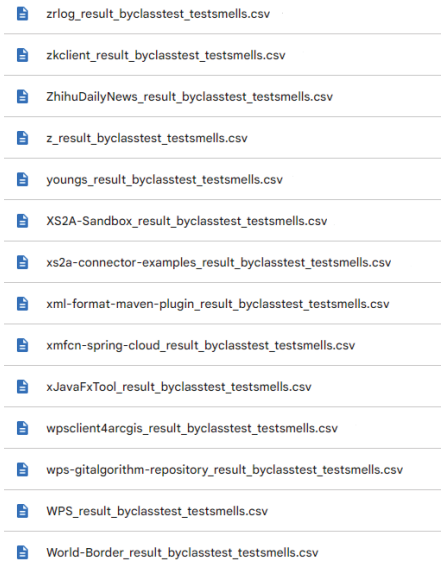


Figure 4.5. Output of JNose Tool after analysis

This document is generated as a process of extensive gathering of the test smell analysis results. They are presented in a format that is both accessible and useful for further review as shown in Figure 4.7.

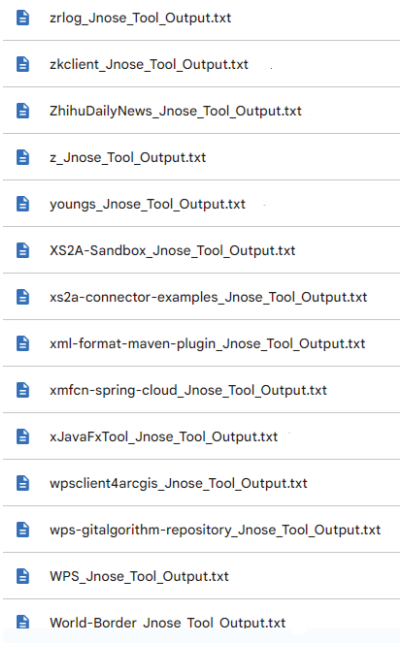


Figure 4.6. Output of read_csv_for_Jnose_tool function

```
ConsentMapperTest.java:
  JNoseTool: Assertion Roulette: 3
CmsDbNativeServiceImplTest.java:
  JNoseTool: Exception Catching Throwing: 2
ConsentServiceImplTest.java:
  JNoseTool: Assertion Roulette: 1
JWTAuthenticationFilterTest.java:
  JNoseTool: Assertion Roulette: 1
  JNoseTool: Unknown Test: 1
OauthServerLinkResolverTest.java:
  JNoseTool: Assertion Roulette: 2
  JNoseTool: Lazy Test: 2
  JNoseTool: Eager Test: 2
AISControllerTest.java:
  JNoseTool: Assertion Roulette: 10
```

Figure 4.7. Part of contents of XS2A-Sandbox_Jnose_Tool_Output.txt file

merge_txt_files(file_paths, output_file) function: The merge_txt_files is the function to bring software testing analysis of these two tools, JNoseTool and TestSmellDetector, together and merge. The purpose of this role is to combine the facts acquired from the dual output text files, which are corresponding to the generated results by two different tools, into one conclusive file titled “Merged_output_txt_file.txt”.

updated_merge_txt_files(input_file_path, output_file_path) function: “Merged_output_txt_file.txt.” file may yield a database organization of data that is not chronological regarding files. Therefore, the JNoseTool and TestSmellDetector findings

might not be next to each other, as a result, the analytical clarity might decline. The updated_merge_txt_files_function reorganizes its output so that the results of the test files from both tools will be presented in order and saved as "Merged_output_txt_file_updated.txt".

remove_empty_lines(input_file_path, output_file_path) function: This function covers a slight nuisance and at the same time a rather important problem that emerges due to the process of merging and applying the outputs of JNoseTool and TestSmellDetector tools — empty lines that produce disturbance and disarrangement of the document. This function removes the space lines no longer used after merging files and saved as “Result_output.txt” as shown Figure 4.8.

```
KeyRingServiceTest.java:
  JNoseTool: Unknown Test: 3
  JNoseTool: Verbose Test: 2
  JNoseTool: Conditional Test Logic: 2
  JNoseTool: Exception Catching Throwing: 1
  JNoseTool: Print Statement: 15
  TestSmellDetectorTool: Conditional Test Logic: 1
  TestSmellDetectorTool: Exception Catching Throwing: 1
  TestSmellDetectorTool: Print Statement: 3
  TestSmellDetectorTool: Unknown Test: 3
  TestSmellDetectorTool: Magic Number Test: 3
TwocheckoutTest.java:
  JNoseTool: Assertion Roulette: 23
  JNoseTool: Exception Catching Throwing: 9
  TestSmellDetectorTool: Assertion Roulette: 8
  TestSmellDetectorTool: Exception Catching Throwing: 11
  TestSmellDetectorTool: Magic Number Test: 15
ParameterEncoderTest.java:
  JNoseTool: Verbose Test: 1
  JNoseTool: Assertion Roulette: 5
  JNoseTool: General Fixture: 1
  TestSmellDetectorTool: Exception Catching Throwing: 5
  TestSmellDetectorTool: General Fixture: 4
  TestSmellDetectorTool: Magic Number Test: 5
ServiceApiDriverTest.java:
  JNoseTool: Verbose Test: 6
  JNoseTool: Assertion Roulette: 55
  JNoseTool: General Fixture: 2
  JNoseTool: Magic Number Test: 3
  JNoseTool: Lazy Test: 4
  TestSmellDetectorTool: Assertion Roulette: 8
  TestSmellDetectorTool: Exception Catching Throwing: 20
  TestSmellDetectorTool: General Fixture: 19
  TestSmellDetectorTool: Magic Number Test: 22
```

Figure 4.8. Result_output.txt file

- **comparing_results_of_each_tool.py**

The python script, comparing_results_of_each_tool.py, is an analytical tool which is created for the purpose of comparing the results of different testing methods which are used in the detection of smells in software development. The following elucidates the constituent elements and operational capabilities of the script:

Data Preparation: The script starts with the definition of the lists with the counts of total occurrences of various test smells by two tools—"JNose Tool" and "TestSmellDetector Tool"— from test files. The co-occurrence relationship of the test smells within their test files. As a first phase, this step is a prerequisite to the following analysis, encouraging a comprehensive distinction of the utilities of the tools in detecting test smells.

Co-occurrence Analysis: The script contains the calculation of co-occurrence matrices for each tool as separate and the calculation of their combination. The comparison of test smells frequencies allows us to identify test smells where often they are found together in test files. The script includes a function that plots these co-occurrence matrices as heatmaps, thus conveying these associations and correlations in a visual and interpretable form for the test smells.

Ratio Calculation and Comparison: The script puts into operation the ratios that represent the frequency of each smell of each of the tools that are used both individually and together. The sensitivity values are displayed correspondingly to the bars of the chart thus assisting in the direct comparison of the tool's sensitivity to different test smells.

Visualization: The script works with matplotlib which is a Python plotting library to create visual representations of the analysis. These visualizations show us results of the co-occurrence analysis and the results of ratio analysis in a clear and accessible way.

File Management and Output: Finally, the script is implemented for managing output directories and saving the generated visualizations as file.

In summary, the Python script is an implementation of a methodological way of finding out which tools are better at locating test code smells inside software test files. Through the implementation of co-occurrence analysis, ratio calculation and comparison, data visualization, and file management, this script can provide important information regarding the efficiency of using the tools and the future development of software testing and quality assurance.

- **jnose_website.py**

This script 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", using the tool's capability to analyze each project for the presence of test smells. Then, when the analysis is done, the results are downloaded in the .csv format, which is a formatted and structured documentation of test smell data per project.

4.3. Results and Discussion

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. As you can see, they detected a similar number of total test smells. Figure 4.9 shows Ratio of Total Test Smell for Each Tool.

Figure 4.10 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.

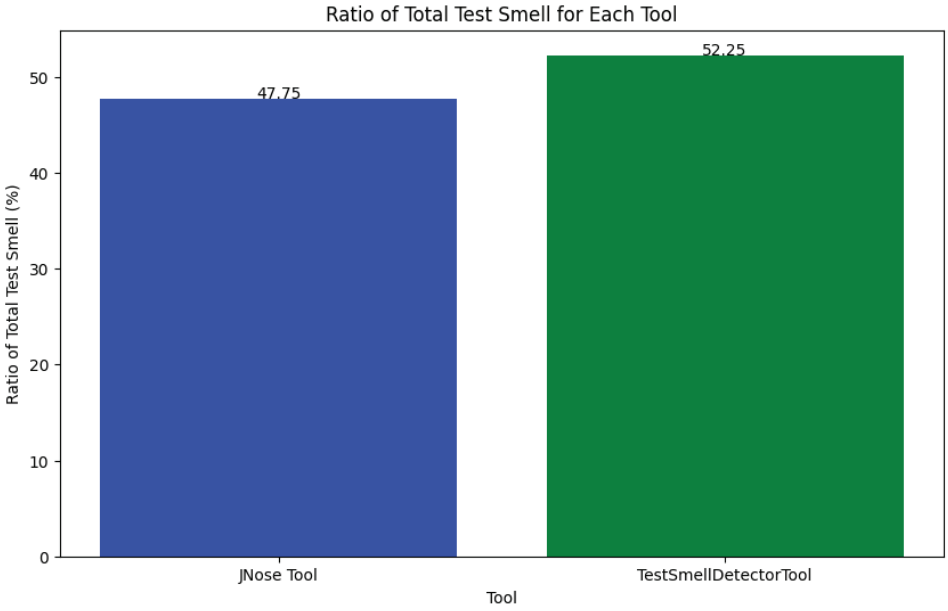


Figure 4.9. Ratio_of_Total_Test_Smells_for_Each_Tool

Figure 4.10 shows that the 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.

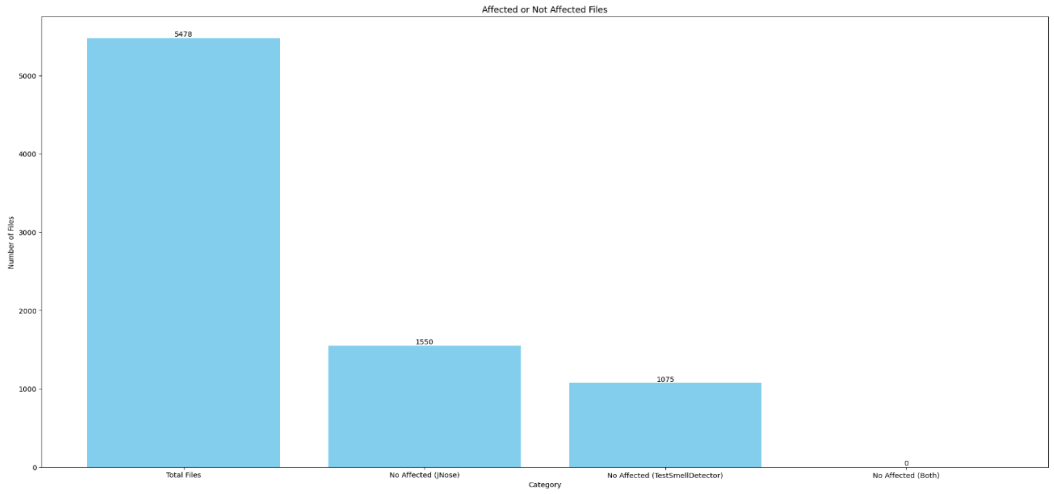


Figure 4.10. Number of Affected and not Affected Files

4.1.3.1. Total Number 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 4.11. The 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 test 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 smells.

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 of the same. 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.

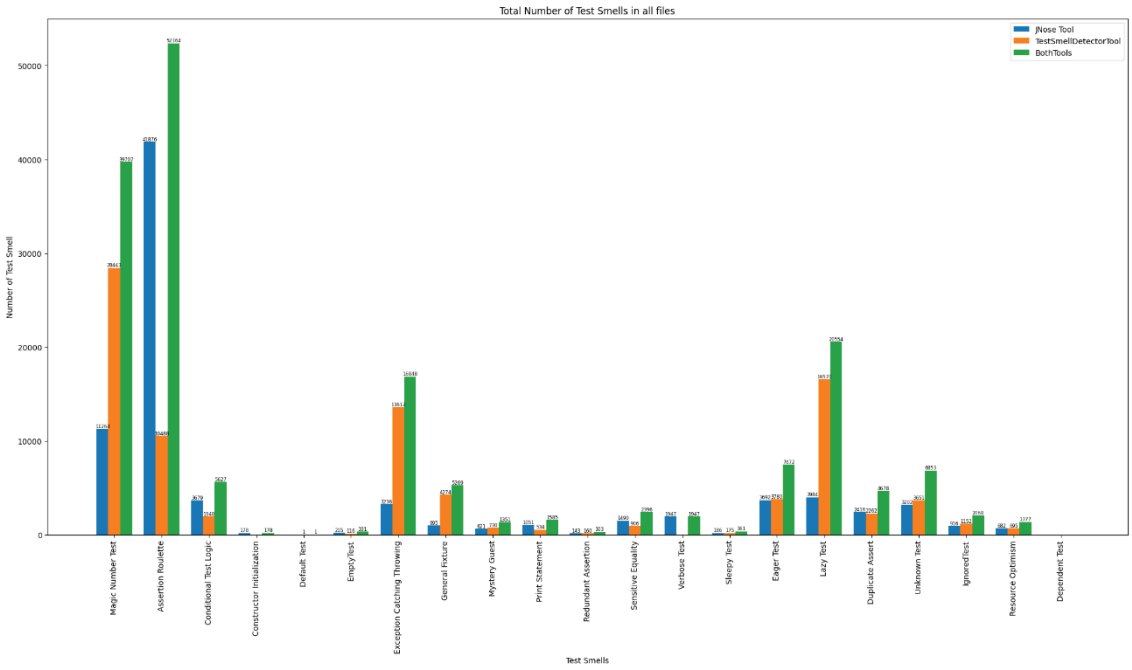


Figure 4.11. Total Number of Test Smells with using JNose and TestSmellDetector Tools in all files

For the other test smells as following, JNose tool performed high detection rates: 'Eager Test' with detection rate of 3692, 'Conditional Test Logic' with detection rate of 3679, 'Exceptional Catching Throwing' with detection rate of 3236, 'Unknown Test' with detection rate of 3202, 'Duplicate Assert' with detection rate of 2416, 'Verbose Test' with detection rate of 1947, and 'Sensitive Equality' with detection rate of 1490.

Similarly, TestSmellDetector Tool was more effective for following test smells and detected them frequently: 'Assertion Roulette' with 10488 occurrence, 'General Fixture' with 4274 occurrence, 'Eager Test' with 3780 occurrence, 'Unknown Test' with 3651 occurrence, 'Duplicate Assert' with 2262 occurrence, 'Conditional Test Logic' with 1948 occurrence and 'Ignored Test' with 1152 occurrence.

As you can see in the results, both tools detected almost the same test smells with different occurrences. In addition, there are differences for detection of 'Verbose Test', 'Sensitive Equality', 'General Fixture' and 'Ignored Test'. The JNose tool detected 'Verbose Test' and 'Sensitive Equality' (1847 and 1490 respectively) more than the TestSmellDetector (0 and 906 respectively). Similarly, 'General Fixture' and 'Ignored Test' are detected more by TestSmellDetector (4274 and 1152 respectively) than by JNose Tool (995 and 916 respectively). Detection of these smells varies depending on the

tool used. We can say that used GitHub projects have the smells that we mentioned above mostly and have bad code quality. The possible common problems in test code may be:

- For ‘Assertion Roulette’, there are added several assertions to a single test to check multiple conditions, often neglecting to add explanatory messages.
- For ‘Magic Number Test’, there are usages of hardcoded, unexplained numeric values, which can easily slip into code as developers hard-code expected results or parameters.
- For ‘Lazy Test’, there are not fully coverages for the expected functionalities, often because tests are not updated to reflect changes in the application's requirements or functionality.
- For ‘Eager Test’, there are trials to check too many functionalities at once, which is a typical result of trying to reduce the number of test methods without considering the isolation of functionalities.
- For ‘Conditional Test Logic’ there are complex conditional logics within tests. Particularly in scenarios where different outcomes need to be validated under varied conditions.
- For ‘Exception Catching Throwing’, there are improper handlings or testing of exceptions. The test may fail to adequately assert the throwing of exceptions or might overly generalize exception handling, catching more than it should.
- For ‘Unknown Test’, there are tests where documentation and descriptive naming conventions are overlooked. So, it leads to tests that others find difficult to understand or relate to specific requirements.
- For ‘Duplicate Assert’ there are multiple times usage of same assertions within a single test or across several tests. It could be due to copy-paste errors or a misunderstanding of what needs to be tested.

  The possible problems in the test code for other commonly detected test smells depending on the tools used:

- For ‘Verbose Test’, there is unnecessary information or complexity, often making it hard to understand what the test aims to verify. This could result from overly complex setup code or multiple responsibilities within a single test.
- For Sensitive Equality, there are usages of overly strict equality checks for their assertions that can lead to brittle tests that fail whenever there are minor, irrelevant

changes in the output. This smell is common in tests that do not focus on the actual requirements but rather on matching exact outputs.

- For ‘General Fixture’, there are extensive setup procedures that are intended to cover multiple test scenarios. Over time, as tests evolve, not all tests need all parts of the setup, leading to inefficiencies and unnecessary complexity in test execution.
- For ‘Ignored Test’, there are tests that are often ignored or skipped during execution due to failures that developers plan to address later, or when the test no longer aligns with current project requirements. This practice can lead to a buildup of unused or outdated tests, particularly in fast-paced development environments.

With considering both tools’ results, following 5 test smells are detected frequently: ‘Assertion Roulette’ with detection rate of 52364, ‘Magic Number Test’ with detection rate of 39707, ‘Lazy Test’ with detection rate of 20554, ‘Exception Catching Throwing’ with detection rate of 16848, ‘Eager Test’ with detection rate of 7472.

Moreover, drawbacks in identifying specific types of test smells were demonstrated by both tools. Significantly, 'Dependent Test' was not identified by both tools, and this was a finding that demands a more in-depth study to determine the extent to which these tools are able to detect such an instrument. Similarly, with using JNose Tool, following test smells were detected rarely: ‘Default Test’ with detection rate of 0, ‘Redundant Assertion’ with detection rate of 143, 'Constructor Initialization' with detection rate of 178, ‘Sleepy Test’ with detection rate of 186 and ‘Empty Test’ with detection rate of 215.

With using TestSmellDetector results are very close to results of JNose Tool. Almost the same test smells were detected rarely except ‘Verbose Test’. ‘Constructor Initialization’, and ‘Verbose Test’ were not detected, and ‘Default Test’ is detected only once by the Test Smell Detector Tool. Similarly, ‘Empty Test’ was detected 116 times, and ‘Sleepy Test’ was detected 175 times. were the test smells that were the least detected, with very few instances identified, possibly attributed to their lower prevalence among software projects or the specific difficulty in their detection.

These were the test smells that were the least detected, with very few instances identified, possibly attributed to their lower prevalence among software projects or the specific difficulty in their detection.

Finding for RQ1: For JNose tool, ‘Magic Number Test’ and ‘Assertion Roulette’ test smells are detected mostly as 11264 and 41876 respectively. For TestSmellDetector tool, ‘Magic Number Test’ and ‘Lazy Test’ test smells are detected frequently as 28443 and 16570 respectively. Moreover, TestSmellDetector tool detected “89497” more test smells than JNose tool “81773”.

Figure 4.12. Finding for RQ1

4.1.3.2. Ratios of Test Smells by Using Each Tools in All Files

The comparison of the test smells observed in the JNose Tool and the TestSmellDetector Tool in the projects of different software shows us some fascinating aspects about the distribution of frequent testing antipatterns. This analysis is based on the ratio of the total number of test smells detected by each tool across to the number of each test smells type detected by each tool across in all examined files. The percentages show the frequency of each test smell as in Figure 4.13, as a number that reflects the occurrence of these elements within software testing environments.

For the JNose Tool, the 'Assertion Roulette' is found to be the most common with a percentage being 51.21% of the overall test smells. Such a trend implies that most of the tests for the project are composed of multiple assertions of the tests which could create confusion on which assertion was responsible for the test failure. Like 'Assertion Roulette', the following test smells are detected as common in all files: 'Magic Number Test', 'Lazy Test', 'Eager Test', ‘Conditional Test Logic’, 'Exception Catching Throwing', and 'Unknown Test', with a percentage of 13.77, 4.87, 4.51, 4.5, 3.96, and 3.92, respectively.

On the other hand, 'Default Test' and ‘Dependent Test’ are not observed at all in all files. In addition, ‘Redundant Assertion’, 'Constructor Initialization', ‘Sleepy Test’,

and ‘Empty Test’ are the other least observed test smells with a percentage being 0.17%, 0.22%, 0.23% and 0.26%.

Conversely, the TestSmellDetector Tool identified 'Lazy Test' and 'Magic Number Test' as the most common test smell, constituting 31.78% and 18.51% of all detected test smells. In addition, the following test smells are detected as common in all files: 'Exception Catching Throwing', 'Assertion Roulette', ‘General Fixture’ and 'Eager Test' with a percentage of 15.21, 11.72, 4.78, and 4.22, respectively.

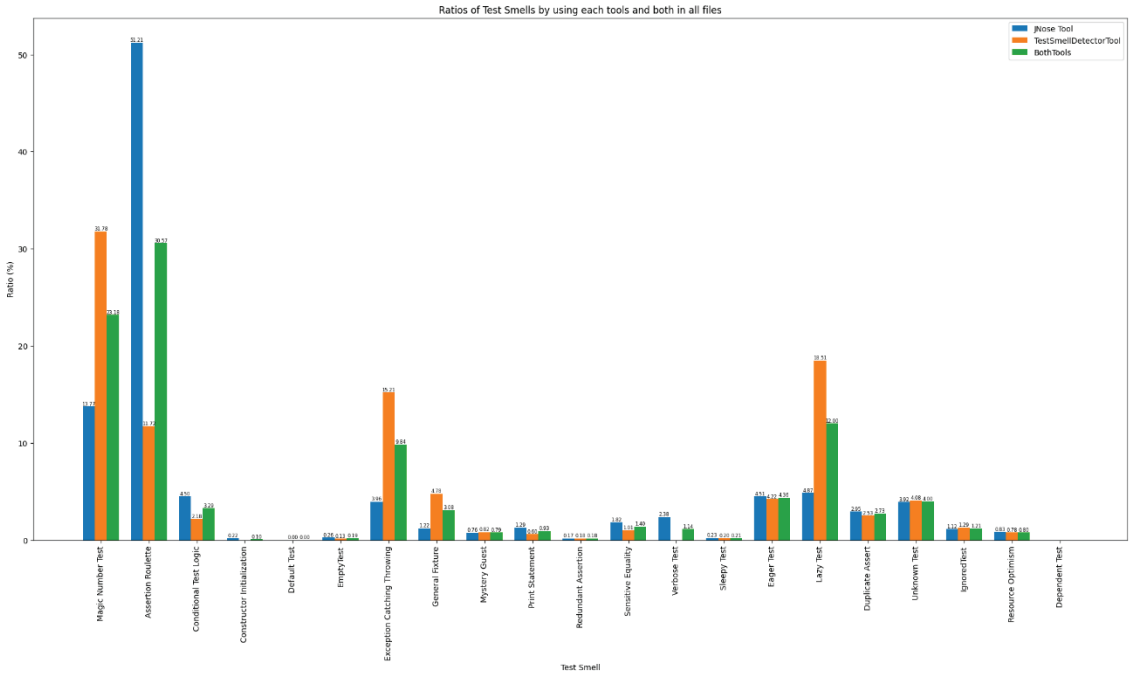


Figure 4.13. Ratios of Test Smells by Using Each Tools in All Files

The least prevalent test smells for the TestSmellDetector Tool are 'Constructor Initialization', 'Default Test', ‘Dependent Test’, ‘Verbose Test’ and ‘Empty Test’. They are not detected at all and the result for 'Default Test' and ‘Dependent Test’ are observed like the same with using JNose Tool. ‘Verbose Test’ and ‘Constructor Initialization’ test smells are also not detected by the TestSmellDetector tool, and ‘Empty Test’ is observed with a percentage of 0.17%.

The large difference between the most frequently detected test smell from each tool 'Assertion Roulette' for JNose and 'Lazy Test' for TestSmellDetector implies the tools are focusing on different parts or have different detection competencies, with each tool possibly to better at identify certain kind of test smells. Moreover, a 13.77% detection

rate for tests 'Magic Number Test' test smell with JNose tool, while TestSmellDetector showed a notably higher 31.78%. Also, a 11.72% detection rate for tests smells like 'Assertion Roulette' with TestSmellDetector, as compared to a notably higher 51.21% with the JNose tool.

4.1.3.3. Number of Affected Files by Each Test Smells

Figure 4.14 shows the JNose Tool and TestSmellDetector Tool in software projects, as it relates to the Number of Affected Files by Each Test Smell in software projects. This analysis provides the absolute number of files affected 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 4.14.

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.

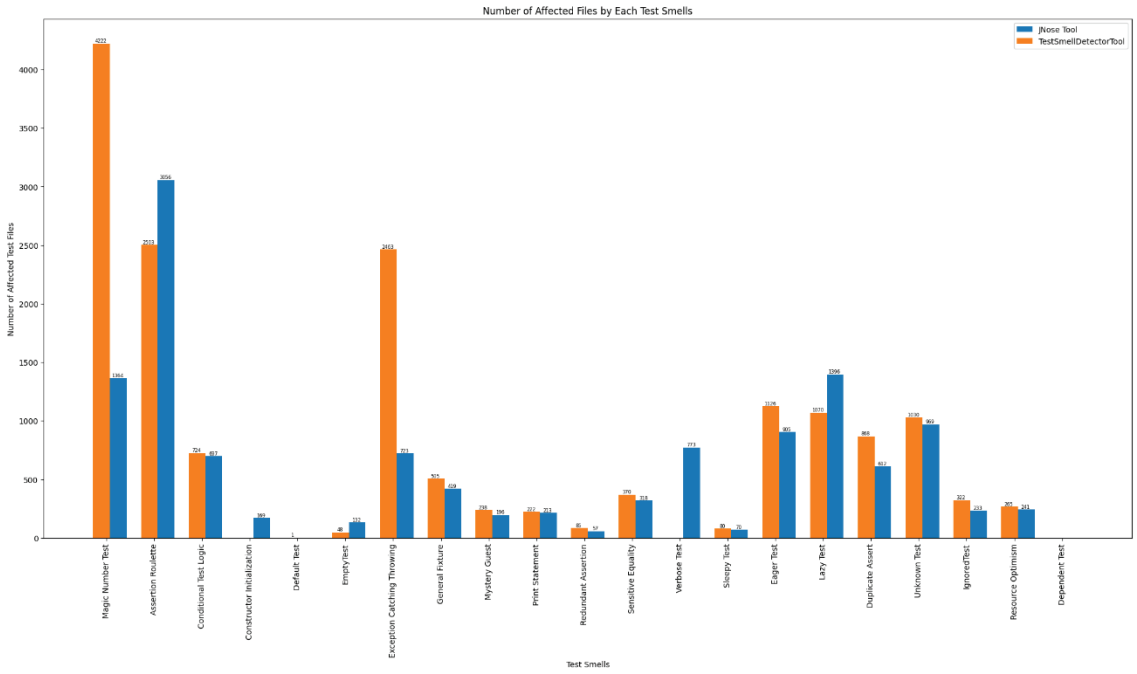


Figure 4.14. Number of Affected Files by Each Test Smells

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.

Finding for RQ2: For JNose tool, test code files are affected by Magic Number Test and Lazy Test test smells mostly as 3056 and 1396 respectively. For TestSmellDetector tool, test code files are affected by Magic Number Test and Assertion Roulette test smells frequently as 4222 and 2503 respectively.

Figure 4.15. Finding for RQ2

4.1.3.4. Ratios of Affected Files by Each Test Smells

Figure 4.16 represents the Ratios of Affected Files by Test Smells using two tools: JNose Tool and TestSmellDetector Tool. Each bar in Figure 4.16 represents a specific type of test smell, with the ratio of files affected by that smell in percentage as shown in Figure 4.16. These ratios were calculated using a formula that considers the total number of files in which a specific test smell was detected, divided by the total number of files analyzed, and then multiplied by 100.

The large difference between the most affected files by 'Assertion Roulette' with a detection rate of %55.79 for the JNose tool (for TestSmellDetector tool, detection rate is %45.69) and by 'Magic Number Test' with a detection rate of %77.07 for the TestSmellDetector tool (for JNose tool, detection rate is %24.9) implies the tools are focusing on different parts or have different detection competencies, with each tool possibly being better at identifying certain kinds of test smells. Moreover, 17.69%, 16.52%, and 13.2% detection rates for 'Unknown Test', 'Eager Test', and "Exception Catching Throwing' test smells with the JNose tool, respectively, while TestSmellDetector showed higher detection rates of 18.8%, 20.55%, and 44.96%. Also, there is a 19.53% detection rate for 'Lazy' with TestSmellDetector, as compared to a higher 25.48% with the JNose tool.

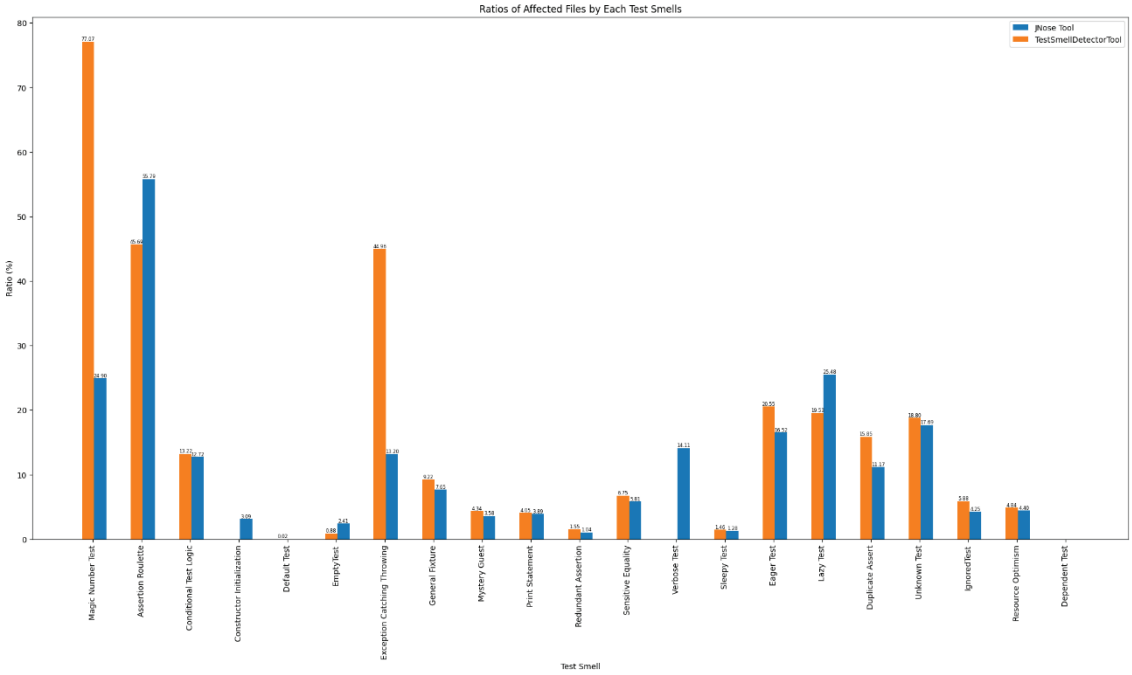


Figure 4.16. Ratios of Affected Files by Each Test Smells

In the prevalence of test smells, 'Assertion Roulette', 'Eager Test', 'Lazy Test', and 'Unknown Test' appear as some of the most frequently detected across both tools, with ratios exceeding 15% in many instances.

4.1.3.5. Co-occurrence of Test Smells based on JNose and TestSmellDetector 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 4.17, 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. The dependency is tight since both smell varieties spring from a higher-level approach of violating an isolated, atomic, and single-purpose rule pattern for tests. When Conditional Test Logic is involved in the testing process, it is just by nature that test cases will cover multiple possible results, which will differ depending on the changing conditions of the requirements. The fact that a test like this often tries to establish so many things at once sets it up to be identified as an 'Eager Test.' It is likely that as soon as a test starts to incorporate 'Conditional Test Logic', it begins to take on multiple responsibilities, hence becoming an 'Eager Test.'

'Conditional Test Logic' may lead to an 'Eager Test' because the test writer, after imposing multiple scenarios within a single test, will continue to extend his test to manage other behaviors, thus the correlation between these two test smells will be increasing.

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. This correlation could arise because both smells stem from a lack of specificity and intentionality in test design. An exception is not asserted in the detector of 'Exception Catching Throwing'. A generic catch-all style with no specific exceptions is used, and this is not good enough to verify the actual behavior of the code under emergency circumstances. On the other hand, 'Unknown Test' in most cases is used for phenomena where the reason for the test was not clear or a purpose of the test is not written explicitly, so the test does not communicate its intent, nor explains on what grounds it is verifying the conditions. When both smells are present, it is likely that the tests are not only poorly documented and unclear but also that they are not effectively validating the error handling paths of the code. The high co-occurrence rate of 'Exception Catching Throwing' with 'Unknown Test' can be explained by the overlap in their root causes: poor test design, inadequate documentation, and the tendency to apply quick fixes under pressure. Addressing these issues involves improving test practices, ensuring clear test purposes, and avoiding unnecessary exception handling in test code. This dual shortage makes the design of test cases more difficult and calls for a revised test design approach that focuses on the optimization of test cases with strict intentions and explicit assertions, especially concerning exception handling and verification, resulting in more enhancements in the test clarity and maintainability.

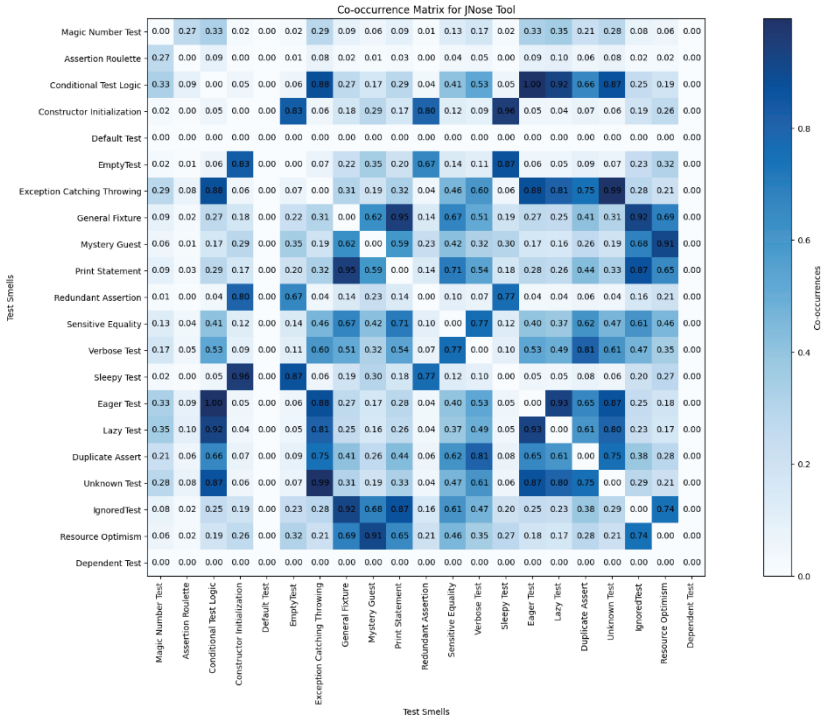


Figure 4.17. Co-occurrence Matrix for JNose Tool

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. The common denominator for the correlation between 'Sleepy Test' and 'Constructor Initialization' is a combination of inadequate handling of test setup and a lack of understanding or utilization of more robust synchronization and initialization mechanisms.

Conversely, 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]. The reasons for the low correlation between these two smells could be:

- Different Origins: 'Magic Number Test' and 'Redundant Assertion' originate from various kinds of coding lapses. Magic numbers often result from a lack of documentation or understanding of the code under test, while redundant assertions tend to stem from an overly cautious approach to ensuring a certain condition is met or from copy-pasting test code without proper refinement.
- No Direct Interaction: The presence of magic numbers does not require or logically lead to redundant assertions. A test can have magic numbers without any need to assert a condition more than once.

- Independent Nature: Both smells can independently exist without affecting each other. A test can be poorly documented with magic numbers yet have a perfectly concise and non-redundant set of assertions.

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. The reasons for the low correlations between 'Mystery Guest' and 'Assertion Roulette', and 'Empty Test' and 'Assertion Roulette' might include:

- Differing Problems: The issues these smells represent do not relate to one another. 'Mystery Guest' deals with unclear test dependencies, while 'Assertion Roulette' concerns the clarity of the assertions within the test.
- Absence of Assertions: Both 'Empty Test' and 'Assertion Roulette' cannot co-occur simply because an 'Empty Test' has no assertions, and therefore cannot create a situation where it's unclear which assertion might fail.
- Independent Correction Pathways: To fix a 'Mystery Guest' failure, a developer could refactor the test so that it is provided enough context or should be less dependent on the external dependencies. To face the issue of an 'Assertion Roulette', the developer will need to showcase clear messages for each of assertions or break down the whole test into several tests with fewer assertions if any. 'Empty Test' requires it to be filled out or extracted. The solutions of the first global warming problem do not stir the same solutions of the other one, and therefore, don't influence the concurrent nature of the smells. These solutions do not intersect and so do not affect the co-occurrence of these smells.

Results for the TestSmellDetector Tool as shown in Figure 4.18, the notable correlation observed in this case is between 'Unknown Test' and 'Eager Test' and their co-occurrence value of [0.97]. The common facet between these two smells is that their correlation is strong because pursuing broad objectives 'Unknown Test' naturally serves the purpose of tests that are overextended in the scope 'Eager Test'. When the purpose of a test is not clearly defined, it becomes much easier for the test to accumulate assertions related to various functionalities or scenarios, effectively becoming an 'Eager Test.' As a result, the test turns out as an 'Eager Test' that has no focus or clarity. In such a situation, a developer may mistake thoroughness or efficiency for the scope of the test. Consequently, they may unwittingly start bundling more checks into the test. On top of that, the lack of a specific aim is a weak point, which makes it impossible to distinguish

the ones which are essential for the assessment, from those which belong to the array of other functionalities. The fact that the test is sprawled in a fashion that is not just a result of an attempt at thoroughness but also indicative of the underlying uncertainty questioning what it is that needs to be evaluated gives further acreage to the 'Eager Test' characteristic.

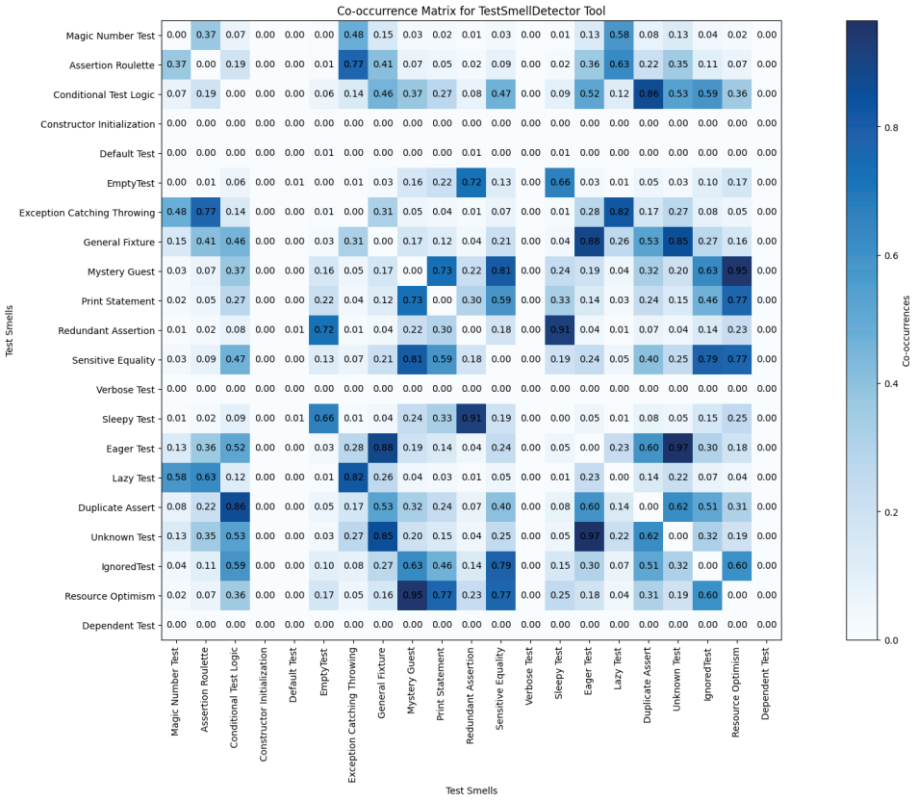


Figure 4.18. Co-occurrence Matrix for TestSmellDetector Tool

The pairing of 'Source Optimism' with 'Mystery Guest' has a strong co-occurrence rate of [0.95] with using TestSmellDetector Tool. Because of the several reasons, strong correlation is observed between 'Source Optimism' and 'Mystery Guest':

- Mutual dependence on External Dependencies: Test files which contain both test smells depend on external resources, such as files or databases, leads to a natural overlap in tests that cause both smells.
- Assumptions about Resource State: 'Source Optimism' is characterized by optimistic assumptions about the availability and state of external resources. 'Source Optimism' is inherently risky and often not explicitly addressed or

documented. So, it is aligning with the uncertain nature of 'Mystery Guest'. The strong correlation is observed because both smells come from a problematic handling of external resources in test cases. It is marked by optimistic assumptions and a lack of clear documentation, which leads to tests that are difficult to maintain.

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. Following reasons that why these test smells do not commonly affect each other:

- 'Magic Number Test' and 'Redundant Assertion': Magic numbers concern unclear literal values in tests, while redundant assertions involve repeating the same check. The use of unclear literals doesn't necessarily lead to repeating assertions, and vice versa.
- 'Magic Number Test' and 'Sleepy Test': The presence of arbitrary literal values ('Magic Number Test') in a test is unrelated to the use of unnecessary wait times ('Sleepy Test'), as these issues stem from fundamentally different testing practices and concerns.
- 'Assertion Roulette' and 'Empty Test': 'Assertion Roulette' involves tests with multiple unclear assertions, whereas an 'Empty Test' contains no executable statements or assertions at all, making their co-occurrence unlikely.
- 'Empty Test' and 'Exception Catching Throwing': Since 'Empty Test' lacks implementation, it cannot concurrently exhibit specific behaviors such as improperly managing exceptions ('Exception Catching Throwing').
- 'Empty Test' and 'Lazy Test': 'Lazy Test' implies a test that inadequately verifies the functionality it's intended to test, often through overly simplistic or incomplete assertions. In contrast, an 'Empty Test' doesn't perform any action or assertion, precluding any form of testing behavior, inadequate or otherwise.

In summary, the co-occurrence matrix not only functions as a diagnostic tool to show detected test smell combinations but also as a strategic asset in identifying which test smells can be separated in practice. In addition, it is beneficial for advancing the

efficacy of testing suites, contributing to the development of more robust and reliable software systems.

Finding for RQ3: There is a considerable relationship between different types of test smells, especially when considering their co-occurrence. Test smells often do not exist in isolation; instead, the presence of one can be a predictor or indicator of others within the same test suite. Between 'Conditional Test Logic' and 'Eager Test' has most strong relationship with a co-occurrence value of [1.00] with using JNose tool. Also, 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. On the other hand, the notable correlation observed in this case is between 'Unknown Test' and 'Eager Test' and their co-occurrence value of [0.97] with using TestSmellDetector tool. Additionally, the pairing of 'Source Optimism' with 'Mystery Guest' has a strong co-occurrence rate of [0.95] with using TestSmellDetector Tool.

Figure 4.19. Finding for RQ3

CHAPTER 5

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 rate 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.

I. The 'Assertion Roulette' is the most prevalent smell in the JNose Tool with 41,876 detections, accounting for 51.21% of all test smells detected by the JNose tool. 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. They are observed in all files, with respective percentages of 13.77, 4.87, 4.51, 4.50, 3.96, and 3.92. 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, accounting for 31.78% and 18.51% of all detected test smells, respectively. 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, with respective percentages of 15.21, 11.72, 4.78, and 4.22.

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.

ENDNOTES

[1] Available at: https://github.com/tassiovirginio/jnose.git

[2] Available at: https://www.eclemma.org/jacoco/

[3] Available at: https://javaparser.org/

[4] Available at: https://wicket.apache.org/

[5] Available at: https://maven.apache.org/

[6] Available at: https://testsmells.org/

[7] Available at: https://doi.org/10.48550/arXiv.2003.05613

[8] Available at: https://arxiv.org/abs/2107.13902

[9] Available at: https://doi.org/10.1145/3379597.3387453

[10] Available at: https://ieeexplore.ieee.org/document/7582740?arnumber=7582740

[11] Available at: https://doi.org/10.1145/3425174.3425212

[12] Available at: https://doi.org/10.1109/ICSME46990.2020.00056

[13] Available at: https://doi.org/10.1109/MSR52588.2021.00071

[14] Available at: https://github.com/arieslab/jnose.

[15] Available at: https://github.com/TestSmells/TestSmellDetector/tree/master

[16] Available at: https://github.com/arieslab/TSSM

REFERENCES

Aberdour, M. 2007. “Achieving Quality in Open-Source Software.” IEEE Software, Software, IEEE, IEEE Softw 24 (1): 58–64.
DOI: https://doi.org/10.1109/MS.2007.2

Bavota, Gabriele, 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.” 2012 28th IEEE International Conference on Software Maintenance (ICSM), Software Maintenance (ICSM), 2012 28th IEEE International Conference On, September, 56–65.
DOI: https://doi.org/10.1109/ICSM.2012.6405253

Bavota, Gabriele, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. “Are Test Smells Really Harmful? An Empirical Study.” Empirical Software Engineering: An International Journal 20 (4): 1052–94.
DOI: https://doi.org/10.1007/s10664-014-9313-0

Campos, Denivan, Larissa Rocha, and Ivan Machado. 2021. “Developers Perception on the Severity of Test Smells: An Empirical Study,” July.
Available: https://research.ebsco.com/linkprocessor/plink?id=3b01ec23-85fb-33b6-a84f-6b9b82b578ad.

Cebeci, Ismail. 2024. “Automated Python Scripts to Use JNose and TestSmellDetector Tools.”.
Available: https://github.com/ismailcebeci/Master_Thesis_Project

Durieux, Thomas, Cesar Soto-Valero, and Benoit Baudry. 2021. “Duets: A Dataset of Reproducible Pairs of Java Library-Clients.” 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), Mining Software Repositories (MSR), 2021 IEEE/ACM 18th International Conference on, MSR, May, 545–49.
DOI: https://doi.org/10.1109/MSR52588.2021.00071

Duvall, P. M., S. Matyas, P. Duvall, and A. Glover. Continuous Integration: Improving Software Quality and Reducing Risk. A Martin Fowler Signature Book. Addison-Wesley, 2007.
Available: https://books.google.com.tr/books?id=MA8QmAEACAAJ.

Garousi, V., and B. Küçük. 2018. “Smells in Software Test Code: A Survey of Knowledge in Industry and Academia.” Journal of Systems and Software 138 (April): 52–81.
DOI: https://doi.org/10.1016/j.jss.2017.12.013

Gopinath, Rahul, Carlos Jensen, and Alex Groce. 2014. “Code Coverage for Suite Evaluation by Developers.” Proceedings of the 36th International Conference on Software Engineering, May, 72–82.
DOI: https://doi.org/10.1145/2568225.2568278

Grano, G., F. Palomba, H.C. Gall, D. Di Nucci, and A. De Lucia. 2019. “Scented since the Beginning: On the Diffuseness of Test Smells in Automatically Generated Test Code.” Journal of Systems and Software 156 (October): 312–27.
DOI: https://doi.org/10.1016/j.jss.2019.07.016

Greiler, Michaela, Arie van Deursen, and Margaret-Anne Storey. 2013. “Automated Detection of Test Fixture Strategies and Smells.” 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference On, March, 322–31
DOI: https://doi.org/10.1109/ICST.2013.45

Harrold, Mary Jean. ‘Testing: A Roadmap’. In Proceedings of the Conference on The Future of Software Engineering, 61–72. ICSE ’00. New York, NY, USA: Association for Computing Machinery, 2000. DOI: https://doi.org/10.1145/336512.336532

Junior, Nildo Silva, Larissa Rocha, Luana Almeida Martins, and Ivan Machado. 2020. “A Survey on Test Practitioners’ Awareness of Test Smells,” March. Available: https://research.ebsco.com/linkprocessor/plink?id=dc10ffb1 -9d27-3e54-a6b4- f27eabc597fc.

Loriot, Benjamin, Fernanda Madeiral, and Martin Monperrus. 2022. “Styler: Learning Formatting Conventions to Repair Checkstyle Violations.” Empirical Software Engineering: An International Journal 27 (6). DOI: https://doi.org/10.1007/s10664-021-10107-0

Martins, L., I. Machado, and H. Costa. 2024. “On the Diffusion of Test Smells and Their Relationship with Test Code Quality of Java Projects.” Journal of Software: Evolution and Process 36 (4). DOI: https://doi.org/10.1002/smr.2532

Meszaros, Gerard, Shaun M. Smith, and Jennitta Andrea. ‘The Test Automation Manifesto’. In Extreme Programming and Agile Methods – XP/Agile Universe 2003, edited by Frank Maurer and Don Wells, 73–81. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. DOI: https://doi.org/10.1007/978-3-540-45122-8_9

Meszaros, Gerard G. 2010. “XUnit Test Patterns and Smells : Improving the ROI of Test Code.” Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, October, 299–300. DOI: https://doi.org/10.1145/1869542.1869622

Murphy, Gail C. 2007. “Houston: We Are in Overload.” 2007 IEEE International Conference on Software Maintenance, Software Maintenance, 2007. ICSM 2007. IEEE International Conference On, October, 1. DOI: https://doi.org/10.1109/ICSM.2007.4362611

Palomba, Fabio, and Andy Zaidman. 2017. “Notice of Retraction: Does Refactoring of Test Smells Induce Fixing Flaky Tests?” 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on, ICSME, September, 1–12. DOI: https://doi.org/10.1109/ICSME.2017.12

Palomba, Fabio, and Andy Zaidman. 2019. “RETRACTED ARTICLE: The Smell of Fear: On the Relation between Test Smells and Flaky Tests.” Empirical Software Engineering: An International Journal 24 (5): 2907–46. DOI: https://doi.org/10.1007/s10664-019-09683-z

Panichella, Annibale, Sebastiano Panichella, Gordon Fraser, Anand Ashok Sawant, and Vincent J. Hellendoorn. 2020. “Revisiting Test Smells in Automatically Generated Tests: Limitations, Pitfalls, and Opportunities.” 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), Software Maintenance and Evolution (ICSME), 2020 IEEE International Conference on, ICSME, September, 523–33. DOI: https://doi.org/10.1109/ICSME46990.2020.00056

Peruma, A., M.W. Mkaouer, K. Almalki, C.D. Newman, A. Ouni, and F. Palomba. 2020. “On the Distribution of Test Smells in Open Source Android Applications: An Exploratory Study.” CASCON 2019 Proceedings - Conference of the Centre for Advanced Studies on Collaborative Research - Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering, January, 193–202. Available: https://research.ebsco.com/linkprocessor/plink?id=bdc5aa42-a340- 3590-9fe6-cbd28094b85e.

Peruma, Anthony, Khalid Almalki, Christian D. Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2020. “TsDetect: An Open Source Test Smells Detection Tool.” Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, November, 1650–54.
DOI: https://doi.org/10.1145/3368089.3417921

Peruma, Anthony Shehan Ayam. "What the Smell? An Empirical Investigation on the Distribution and Severity of Test Smells in Open Source Android Applications" (2018). Thesis. Rochester Institute of Technology.
Available: https://repository.rit.edu/theses/9774

Rothermel, G., R.H. Untch, Chengyun Chu, and M.J. Harrold. 2001. “Prioritizing Test Cases for Regression Testing.” IEEE Transactions on Software Engineering, Software Engineering, IEEE Transactions on, IIEEE Trans. Software Eng 27 (10): 929–48.
DOI: https://doi.org/10.1109/32.962562

Soares, Elvys, Márcio Ribeiro, Guilherme Amaral, Rohit Gheyi, Leo Fernandes, Alessandro Garcia, Baldoino Fonseca, and André Santos. 2020. “Refactoring Test Smells : A Perspective from Open-Source Developers.” Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing, October, 50–59.
DOI: https://doi.org/10.1145/3425174.3425212

Spadini, Davide, Fabio Palomba, Andy Zaidman, Magiel Bruntink, and Alberto Bacchelli. 2018. “On the Relation of Test Smells to Software Code Quality.” 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), Software Maintenance and Evolution (ICSME), 2018 IEEE International Conference on, ICSME, September, 1–12.
DOI: https://doi.org/10.1109/ICSME.2018.00010

Spadini, Davide, Martin Schvarcbacher, Ana-Maria Oprescu, Magiel Bruntink, and Alberto Bacchelli. 2020. “Investigating Severity Thresholds for Test Smells.” 2020 IEEE/ACM 17th International Conference on Mining Software Repositories (MSR), Mining Software Repositories (MSR), 2020 IEEE/ACM 17th International Conference on, MSR, May, 311–21. DOI: https://doi.org/10.1145/3379597.3387453

Tufano, Michele, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2016. “An Empirical Investigation into the Nature of Test Smells.” 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Automated Software Engineering (ASE), 2016 31st IEEE/ACM International Conference On, September, 4–15. Available: https://research.ebsco.com/linkprocessor/plink?id=828fc978-05ac-3151-b198-b434a6343283.

Vahabzadeh, Arash, Amin Milani Fard, and Ali Mesbah. 2015. “An Empirical Study of Bugs in Test Code.” 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference On, September, 101–10. DOI: https://doi.org/10.1109/ICSM.2015.7332456

van Deursen, Arie, Leon Moonen, Alex van den Bergh, and Gerard Kok. Refactoring Test Code. Software Engineering [SEN]. CWI, 2001. Available: https://ir.cwi.nl/pub/4324

Virgínio, Tássio, Railana Santana, Luana Almeida Martins, Larissa Rocha Soares, Heitor Costa, and Ivan Machado. 2019. “On the Influence of Test Smells on Test Coverage.” Proceedings of the XXXIII Brazilian Symposium on Software Engineering, September, 467–71. DOI: https://doi.org/10.1145/3350768.3350775

Virgínio, Tássio, Luana Martins, Larissa Rocha, Railana Santana, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. “JNose : Java Test Smell Detector.” Proceedings of the 34th Brazilian Symposium on Software Engineering, October, 564–69. DOI: https://doi.org/10.1145/3422392.3422499

APPENDICES

APPENDIX A

A.1. Project Database List

Table A.1. List of GitHub projects

Full_name_modified	clone_url
00-matt_moneropool	https://github.com/00-matt/moneropool.git
0428402001_hbase-oss	https://github.com/0428402001/hbase-oss.git
0rtis_mochimo-farm-manager	https://github.com/0rtis/mochimo-farm-manager.git
0xCopy_RelaxFactory	https://github.com/0xCopy/RelaxFactory.git
0xERR0R_jmstool	https://github.com/0xERR0R/jmstool.git
0xl2oot_harbor-java-client	https://github.com/0xl2oot/harbor-java-client.git
0xnm_BTC-e-client-for-Android	https://github.com/0xnm/BTC-e-client-for-Android.git
0xpr03_VocableTrainer-Android	https://github.com/0xpr03/VocableTrainer-Android.git
0xZhangKe_ShiZhong	https://github.com/0xZhangKe/ShiZhong.git
1000Memories_photon-core	https://github.com/1000Memories/photon-core.git
100rabhkr_DownZLibrary	https://github.com/100rabhkr/DownZLibrary.git

Cont. on next page

Table A.1 (cont.)

1024-lab_smart-admin	https://github.com/1024-lab/smart-admin.git
10clouds_InifiniteRecyclerView	https://github.com/10clouds/InifiniteRecyclerView.git
1123_johnson	https://github.com/1123/johnson.git
12315jack_j1st-mqtt	https://github.com/12315jack/j1st-mqtt.git
1336037686_software-demo	https://github.com/1336037686/software-demo.git
151376liujie_wechat-core	https://github.com/151376liujie/wechat-core.git
15knots_cmake4eclipse	https://github.com/15knots/cmake4eclipse.git
15knots_cmakeed	https://github.com/15knots/cmakeed.git
1902-feb04-java_training-code	https://github.com/1902-feb04-java/training-code.git
1and1_cosmo	https://github.com/1and1/cosmo.git
1and1_dim	https://github.com/1and1/dim.git
1and1_foss-parent	https://github.com/1and1/foss-parent.git
1and1_go-maven-poller	https://github.com/1and1/go-maven-poller.git
1and1_Troilus	https://github.com/1and1/Troilus.git
1C-Company_dt-example-plugins	https://github.com/1C-Company/dt-example-plugins.git
1c-syntax_sonar-bsl-plugin-community	https://github.com/1c-syntax/sonar-bsl-plugin-community.git
1fish2_BBQTimer	https://github.com/1fish2/BBQTimer.git
1hakr_ALauncher	https://github.com/1hakr/ALauncher.git

Cont. on next page

Table A.1 (cont.)

1m5_1m5-core	https://github.com/1m5/1m5-core.git
1Ridav_PengueeBot	https://github.com/1Ridav/PengueeBot.git
1tontech_intellij-spring-assistant	https://github.com/1tontech/intellij-spring-assistant.git
200Puls_darksky-forecast-api	https://github.com/200Puls/darksky-forecast-api.git
2020NCOV_MiniProgram-server-JAVA	https://github.com/2020NCOV/MiniProgram-server-JAVA.git
20n_act	https://github.com/20n/act.git
294678380_cloudDemo	https://github.com/294678380/cloudDemo.git
2Checkout_2checkout-java	https://github.com/2Checkout/2checkout-java.git
33cn_chain33-sdk-java	https://github.com/33cn/chain33-sdk-java.git
360jinrong_chronus	https://github.com/360jinrong/chronus.git
3breadt_dd-plist	https://github.com/3breadt/dd-plist.git
3cky_bkemu-android	https://github.com/3cky/bkemu-android.git
3pillarlabs_spring-data-simpledb	https://github.com/3pillarlabs/spring-data-simpledb.git
3pillarlabs_spring-integration-aws	https://github.com/3pillarlabs/spring-integration-aws.git
3redronin_mu-server	https://github.com/3redronin/mu-server.git
3scale-labs_3scale_ws_api_for_java	https://github.com/3scale-labs/3scale_ws_api_for_java.git
3sidedcube_Android-GeoGson	https://github.com/3sidedcube/Android-GeoGson.git
435242634_Spring-Boot-Demo	https://github.com/435242634/Spring-Boot-Demo.git

Cont. on next page

Table A.1 (cont.)

4ncov_service-4ncov	https://github.com/4ncov/service-4ncov.git
4PERTURE_despacito_launcher	https://github.com/4PERTURE/despacito_launcher.git
50onRed_mock-jedis	https://github.com/50onRed/mock-jedis.git
512433465_JacocoPlus	https://github.com/512433465/JacocoPlus.git
51Degrees_Java-Device-Detection	https://github.com/51Degrees/Java-Device-Detection.git
51nb_marble	https://github.com/51nb/marble.git
527088995_dian	https://github.com/527088995/dian.git
52inc_Scoops	https://github.com/52inc/Scoops.git
52North_ArcGIS-Server-SOS-Extension	https://github.com/52North/ArcGIS-Server-SOS-Extension.git
52North_arctic-sea	https://github.com/52North/arctic-sea.git
52North_ecmwf-dataset-crawl	https://github.com/52North/ecmwf-dataset-crawl.git
52North_javaPS	https://github.com/52North/javaPS.git
52North_matlab-control	https://github.com/52North/matlab-control.git
52North_ows-11-instagram	https://github.com/52North/ows-11-instagram.git
52North_SensorPlanningService	https://github.com/52North/SensorPlanningService.git
52North_sensorweb-server-helgoland	https://github.com/52North/sensorweb-server-helgoland.git
52North_SensorWebClient	https://github.com/52North/SensorWebClient.git
52North_SES	https://github.com/52North/SES.git

Cont. on next page

Table A.1 (cont.)

52North_SOS	https://github.com/52North/SOS.git
52North_sos-importer	https://github.com/52North/sos-importer.git
52North_Supervisor	https://github.com/52North/Supervisor.git
52North_WeatherDataCollector	https://github.com/52North/WeatherDataCollector.git
52North_WPS	https://github.com/52North/WPS.git
52North_wps-gitalgorithm-repository	https://github.com/52North/wps-gitalgorithm-repository.git
52North_wpsclient4arcgis	https://github.com/52North/wpsclient4arcgis.git
52North_youngs	https://github.com/52North/youngs.git
58code_Gaea	https://github.com/58code/Gaea.git
5calls_android	https://github.com/5calls/android.git
5waynewang_diamond	https://github.com/5waynewang/diamond.git
616slayer616_gradle-minify-plugin	https://github.com/616slayer616/gradle-minify-plugin.git
632team_EasyHousing	https://github.com/632team/EasyHousing.git
6tail_lunar-java	https://github.com/6tail/lunar-java.git
724_irbill	https://github.com/724/irbill.git
734839030_seezoon-framework-all	https://github.com/734839030/seezoon-framework-all.git
75py_DisableManager	https://github.com/75py/DisableManager.git
7ep_demo	https://github.com/7ep/demo.git

Cont. on next page

Table A.1 (cont.)

7upcat_agile-wroking-backend	https://github.com/7upcat/agile-wroking-backend.git
84n4n4_FloatSight	https://github.com/84n4n4/FloatSight.git
850759383_ZhihuDailyNews	https://github.com/850759383/ZhihuDailyNews.git
864381832_xJavaFxTool	https://github.com/864381832/xJavaFxTool.git
8BitPlus_BitPlus	https://github.com/8BitPlus/BitPlus.git
8enet_apkeditor	https://github.com/8enet/apkeditor.git
8kdata_phoebe	https://github.com/8kdata/phoebe.git
918xj_spring-boot-seed	https://github.com/918xj/spring-boot-seed.git
9231058_AP101	https://github.com/9231058/AP101.git
94fzb_zrlog	https://github.com/94fzb/zrlog.git
99soft_autobind	https://github.com/99soft/autobind.git
a-pavlov_jed2k	https://github.com/a-pavlov/jed2k.git
a-r-d_Bellman-Form-BTCe-Arbitrager	https://github.com/a-r-d/Bellman-Form-BTCe-Arbitrager.git
a-r-d_java-1-class-demos	https://github.com/a-r-d/java-1-class-demos.git
a-schild_jave2	https://github.com/a-schild/jave2.git
a-schild_nextcloud-java-api	https://github.com/a-schild/nextcloud-java-api.git
A-Zaiats_android-mvvm	https://github.com/A-Zaiats/android-mvvm.git
a11n_CustomLintRules	https://github.com/a11n/CustomLintRules.git

Cont. on next page

Table A.1 (cont.)

a11n_devfest-2016-realm	https://github.com/a11n/devfest-2016-realm.git
a11n_lint-junit-rule	https://github.com/a11n/lint-junit-rule.git
a2ndrade_q-intellij-plugin	https://github.com/a2ndrade/q-intellij-plugin.git
a970066364_spring-cloud-alibaba-seata	https://github.com/a970066364/spring-cloud-alibaba-seata.git
AAA-AA_basic-tools	https://github.com/AAA-AA/basic-tools.git
aaberg_sql2o	https://github.com/aaberg/sql2o.git
aadnk_ProtocolLib	https://github.com/aadnk/ProtocolLib.git
AAkira_ExpandableLayout	https://github.com/AAkira/ExpandableLayout.git
aankur_spring-authentication-session-oauth2	https://github.com/aankur/spring-authentication-session-oauth2.git
Aaron1011_WhoWas	https://github.com/Aaron1011/WhoWas.git
aaronshan_hive-third-functions	https://github.com/aaronshan/hive-third-functions.git
aaronweihe_ThreeTen-Backport-Gson-Adapter	https://github.com/aaronweihe/ThreeTen-Backport-Gson-Adapter.git
aaryn101_lol4j	https://github.com/aaryn101/lol4j.git
aasaru_drools-training	https://github.com/aasaru/drools-training.git
aaschmid_taskwarrior-java-client	https://github.com/aaschmid/taskwarrior-java-client.git
aashrai_GET-TO-WORK	https://github.com/aashrai/GET-TO-WORK.git
aatarasoff_spring-cloud-marathon	https://github.com/aatarasoff/spring-cloud-marathon.git

Cont. on next page

Table A.1 (cont.)

aatarasoff_spring-one-nio	https://github.com/aatarasoff/spring-one-nio.git
aatarasoff_spring-thrift-api-gateway	https://github.com/aatarasoff/spring-thrift-api-gateway.git
aatarasoff_spring-thrift-starter	https://github.com/aatarasoff/spring-thrift-starter.git
ab-book_code_yunfei	https://github.com/ab-book/code_yunfei.git
abahgat_suffixtree	https://github.com/abahgat/suffixtree.git
abashev_spring-workflow	https://github.com/abashev/spring-workflow.git
abdullahalrifat_FoodOrderingSystem	https://github.com/abdullahalrifat/FoodOrderingSystem.git
abego_treelayout	https://github.com/abego/treelayout.git
abel533_ECharts	https://github.com/abel533/ECharts.git
abel533_guns	https://github.com/abel533/guns.git
abelidze_planner-server	https://github.com/abelidze/planner-server.git
AbFab3D_AbFab3D	https://github.com/AbFab3D/AbFab3D.git
abh1nav_styx	https://github.com/abh1nav/styx.git
abhijitparida_bunk	https://github.com/abhijitparida/bunk.git
abhijitvalluri_fitnotifications	https://github.com/abhijitvalluri/fitnotifications.git
abhishek-ch_Awesome_Algorithm	https://github.com/abhishek-ch/Awesome_Algorithm.git
abid-khan_spring-security-rest	https://github.com/abid-khan/spring-security-rest.git
ably_ably-java	https://github.com/ably/ably-java.git

Cont. on next page

Table A.1 (cont.)

abohomol_cookietray	https://github.com/abohomol/cookietray.git
aboutsip_pkts	https://github.com/aboutsip/pkts.git
aboutsip_sipstack	https://github.com/aboutsip/sipstack.git
abranhe_allalgorithms-java	https://github.com/abranhe/allalgorithms-java.git
abrensch_brouter	https://github.com/abrensch/brouter.git
abstools_abstools	https://github.com/abstools/abstools.git
abstractj_kalium	https://github.com/abstractj/kalium.git
abuchanan920_historybook	https://github.com/abuchanan920/historybook.git
abused_World-Border	https://github.com/abused/World-Border.git
ac2cz_FoxTelem	https://github.com/ac2cz/FoxTelem.git
acadet_springbok	https://github.com/acadet/springbok.git
AcadiaSoft_simm-lib	https://github.com/AcadiaSoft/simm-lib.git
acanda_eclipse-pmd	https://github.com/acanda/eclipse-pmd.git
acanda_spring-banner-plugin	https://github.com/acanda/spring-banner-plugin.git
AccelerationNet_access2csv	https://github.com/AccelerationNet/access2csv.git
acciente_oacc-core	https://github.com/acciente/oacc-core.git
Accordance_microservice-dojo	https://github.com/Accordance/microservice-dojo.git

Cont. on next page

Table A.1 (cont.)

accountingSoftwareCSE343Group4_accounting_Soft	https://github.com/accountingSoftwareCSE343Group4/accounting_Soft.git
acebaggins_guava-collectors	https://github.com/acebaggins/guava-collectors.git
acegi_xml-format-maven-plugin	https://github.com/acegi/xml-format-maven-plugin.git
acelera-dev_acelera-dev-brasil-2019-01	https://github.com/acelera-dev/acelera-dev-brasil-2019-01.git
acgray_jplow	https://github.com/acgray/jplow.git
acharapko_retroscope-lib	https://github.com/acharapko/retroscope-lib.git
achaussende_tp-2D-cutting-stock-problem	https://github.com/achaussende/tp-2D-cutting-stock-problem.git
achuzhmarov_test-tutorial	https://github.com/achuzhmarov/test-tutorial.git
aclement_spring-boot-graal-feature	https://github.com/aclement/spring-boot-graal-feature.git
aclemons_hibernate-hdb-demo	https://github.com/aclemons/hibernate-hdb-demo.git
acmerobotics_relic-recovery	https://github.com/acmerobotics/relic-recovery.git
acmi_L2io	https://github.com/acmi/L2io.git
Acosix_alfresco-simple-content-stores	https://github.com/Acosix/alfresco-simple-content-stores.git
acquia_http-hmac-java	https://github.com/acquia/http-hmac-java.git
acr31_features-javac	https://github.com/acr31/features-javac.git
actions-on-google_dialogflow-conversation-components-java	https://github.com/actions-on-google/dialogflow-conversation-components-java.git
actions-on-google_dialogflow-webhook-boilerplate-java	https://github.com/actions-on-google/dialogflow-webhook-boilerplate-java.git

Cont. on next page

Table A.1 (cont.)

activelylazy_coverage-example	https://github.com/activelylazy/coverage-example.git
activeviam_autopivot	https://github.com/activeviam/autopivot.git
activeviam_pivot-spring-boot	https://github.com/activeviam/pivot-spring-boot.git
activey_licket	https://github.com/activey/licket.git
activityworkshop_GpsPrune	https://github.com/activityworkshop/GpsPrune.git
actorapp_actor-curve25519	https://github.com/actorapp/actor-curve25519.git
actorapp_actor-platform	https://github.com/actorapp/actor-platform.git
ACWI-SSWD_nldi-services	https://github.com/ACWI-SSWD/nldi-services.git
ad-freiburg_pdfact	https://github.com/ad-freiburg/pdfact.git
ad-tech-group_openssp	https://github.com/ad-tech-group/openssp.git
adaa-polsl_RuleKit	https://github.com/adaa-polsl/RuleKit.git
Adamant-im_adamant-android	https://github.com/Adamant-im/adamant-android.git
adamantoise_robocrosswords	https://github.com/adamantoise/robocrosswords.git
AdamBien_airfield	https://github.com/AdamBien/airfield.git
AdamBien_breakr	https://github.com/AdamBien/breakr.git
AdamBien_cors	https://github.com/AdamBien/cors.git
AdamBien_enhydrator	https://github.com/AdamBien/enhydrator.git
AdamBien_firehose	https://github.com/AdamBien/firehose.git

Cont. on next page

Table A.1 (cont.)

AdamBien_javaee-bce-archetype	https://github.com/AdamBien/javaee-bce-archetype.git
AdamBien_javaee-bce-pom	https://github.com/AdamBien/javaee-bce-pom.git
AdamBien_javaee-calculator	https://github.com/AdamBien/javaee-calculator.git
AdamBien_jc2	https://github.com/AdamBien/jc2.git
AdamBien_loadr	https://github.com/AdamBien/loadr.git
AdamBien_nano	https://github.com/AdamBien/nano.git
AdamBien_perceptor	https://github.com/AdamBien/perceptor.git
AdamBien_porcupine	https://github.com/AdamBien/porcupine.git
AdamBien_rulz	https://github.com/AdamBien/rulz.git
AdamBien_wad	https://github.com/AdamBien/wad.git
adamcin_httpsig-java	https://github.com/adamcin/httpsig-java.git
adamcin_oakpal	https://github.com/adamcin/oakpal.git
adamfisk_DNSSEC4J	https://github.com/adamfisk/DNSSEC4J.git
adamheinrich_native-utils	https://github.com/adamheinrich/native-utils.git
adamldavis_hellojava8	https://github.com/adamldavis/hellojava8.git
adamldavis_reactive-streams-in-java	https://github.com/adamldavis/reactive-streams-in-java.git
adamldavis_z	https://github.com/adamldavis/z.git
adamyork_wiremock-velocity-transformer	https://github.com/adamyork/wiremock-velocity-transformer.git

Cont. on next page

Table A.1 (cont.)

adaptris_interlok	https://github.com/adaptris/interlok.git
adavis_sample-android-testing	https://github.com/adavis/sample-android-testing.git
adchilds_JythonScript	https://github.com/adchilds/JythonScript.git
addhen_serializer	https://github.com/addhen/serializer.git
addo47_AbilityBots	https://github.com/addo47/AbilityBots.git
addo47_ExampleBots	https://github.com/addo47/ExampleBots.git
AddstarMC_Prism-Bukkit	https://github.com/AddstarMC/Prism-Bukkit.git
addthis_aho-corasick	https://github.com/addthis/aho-corasick.git
addthis_basis	https://github.com/addthis/basis.git
addthis_cronus	https://github.com/addthis/cronus.git
addthis_hydra	https://github.com/addthis/hydra.git
addthis_meshy	https://github.com/addthis/meshy.git
addthis_MetricCatcher	https://github.com/addthis/MetricCatcher.git
addthis_stream-lib	https://github.com/addthis/stream-lib.git
adelbs_ISO8583	https://github.com/adelbs/ISO8583.git
adempiere_adempiere	https://github.com/adempiere/adempiere.git
AdeptJ_adeptj-modules	https://github.com/AdeptJ/adeptj-modules.git
adessoAG_BrainySnake	https://github.com/adessoAG/BrainySnake.git

Cont. on next page

Table A.1 (cont.)

adessoAG_JenkinsHue	https://github.com/adessoAG/JenkinsHue.git
adessoAG_project-board	https://github.com/adessoAG/project-board.git
adessoAG_wicked-forms	https://github.com/adessoAG/wicked-forms.git
adijo_programming-pearls	https://github.com/adijo/programming-pearls.git
adilcan_simple-erp-springboot	https://github.com/adilcan/simple-erp-springboot.git
Adipa-G_joquery	https://github.com/Adipa-G/joquery.git
aditya-chaturvedi_spark-on-spring-boot	https://github.com/aditya-chaturvedi/spark-on-spring-boot.git
aditya-sridhar_simple-rest-apis	https://github.com/aditya-sridhar/simple-rest-apis.git
aditzel_spring-security-csrf-filter	https://github.com/aditzel/spring-security-csrf-filter.git
adjust_android_sdk	https://github.com/adjust/android_sdk.git
adlered_Picuang	https://github.com/adlered/Picuang.git
adlered_Voter	https://github.com/adlered/Voter.git
adlnet_jxapi	https://github.com/adlnet/jxapi.git
adnovum_sonar-build-breaker	https://github.com/adnovum/sonar-build-breaker.git
Adobe-Consulting-Services_curly	https://github.com/Adobe-Consulting-Services/curly.git
Adobe-Marketing-Cloud_aem-dialog-conversion	https://github.com/Adobe-Marketing-Cloud/aem-dialog-conversion.git
Adobe-Marketing-Cloud_aem-forms	https://github.com/Adobe-Marketing-Cloud/aem-forms.git

Cont. on next page

Table A.1 (cont.)

Adobe-Marketing-Cloud_analytics-live-stream-api-samples	https://github.com/Adobe-Marketing-Cloud/analytics-live-stream-api-samples.git
Adobe-Marketing-Cloud_audiencemanager-api-sample-app	https://github.com/Adobe-Marketing-Cloud/audiencemanager-api-sample-app.git
adobe-sign_AdobeSignJavaSdk	https://github.com/adobe-sign/AdobeSignJavaSdk.git
adobe_adobe-dx	https://github.com/adobe/adobe-dx.git
adobe_aem-cloud-migration	https://github.com/adobe/aem-cloud-migration.git
adobe_aem-eclipse-developer-tools	https://github.com/adobe/aem-eclipse-developer-tools.git
adobe_aem-modernize-tools	https://github.com/adobe/aem-modernize-tools.git
adobe_aem-testing-clients	https://github.com/adobe/aem-testing-clients.git
adobe_commerce-cif-connector	https://github.com/adobe/commerce-cif-connector.git
adobe_commerce-cif-graphql-client	https://github.com/adobe/commerce-cif-graphql-client.git
AdoptOpenJDK_jdk9-jigsaw	https://github.com/AdoptOpenJDK/jdk9-jigsaw.git
AdoptOpenJDK_jitwatch	https://github.com/AdoptOpenJDK/jitwatch.git
adorsys_datasafe	https://github.com/adorsys/datasafe.git
adorsys_keycloak-config-cli	https://github.com/adorsys/keycloak-config-cli.git
adorsys_keystore-management	https://github.com/adorsys/keystore-management.git
adorsys_ledgers	https://github.com/adorsys/ledgers.git
adorsys_oauth	https://github.com/adorsys/oauth.git

Cont. on next page

Table A.1 (cont.)

adorsys_psd2-accelerator	https://github.com/adorsys/psd2-accelerator.git
adorsys_secure-token-service	https://github.com/adorsys/secure-token-service.git
adorsys_xs2a-connector-examples	https://github.com/adorsys/xs2a-connector-examples.git
adorsys_XS2A-Sandbox	https://github.com/adorsys/XS2A-Sandbox.git
adr_e-adr	https://github.com/adr/e-adr.git
adragomir_hbase-indexing-library	https://github.com/adragomir/hbase-indexing-library.git
AdrianBZG_N_Queens_Puzzle	https://github.com/AdrianBZG/N_Queens_Puzzle.git
AdrianCitu_GenericCSRFFilter	https://github.com/AdrianCitu/GenericCSRFFilter.git
adriancretu_beacons-android	https://github.com/adriancretu/beacons-android.git
adrianeboyd_BrillMooreSpellChecker	https://github.com/adrianeboyd/BrillMooreSpellChecker.git
adrianmo_jmeter-backend-azure	https://github.com/adrianmo/jmeter-backend-azure.git
adrianobrito_errare-humanum-est	https://github.com/adrianobrito/errare-humanum-est.git
adrianulbona_hmm	https://github.com/adrianulbona/hmm.git
adrianulbona_jts-discretizer	https://github.com/adrianulbona/jts-discretizer.git
adridadou_eth-contract-api	https://github.com/adridadou/eth-contract-api.git
adrobisch_brainslug	https://github.com/adrobisch/brainslug.git
adrobisch_raml-converter	https://github.com/adrobisch/raml-converter.git
Adrodoc_MPL	https://github.com/Adrodoc/MPL.git

Cont. on next page

Table A.1 (cont.)

AdRoll_cantor	https://github.com/AdRoll/cantor.git
ADSC-Resa_resa	https://github.com/ADSC-Resa/resa.git
advantageous_boon	https://github.com/advantageous/boon.git
advantageous_ddp-client-java	https://github.com/advantageous/ddp-client-java.git
advantageous_konf	https://github.com/advantageous/konf.git
advantageous_qbit	https://github.com/advantageous/qbit.git
advantageous_reakt	https://github.com/advantageous/reakt.git
adyliu_jafka	https://github.com/adyliu/jafka.git
adyliu_zkclient	https://github.com/adyliu/zkclient.git
Aegeaner_kafka-connector-redis	https://github.com/Aegeaner/kafka-connector-redis.git
aionnetwork_aion_api	https://github.com/aionnetwork/aion_api.git
airbnb_AirMapView	https://github.com/airbnb/AirMapView.git
airbnb_airpal	https://github.com/airbnb/airpal.git
airbnb_dynein	https://github.com/airbnb/dynein.git
airbnb_epoxy	https://github.com/airbnb/epoxy.git
airbnb_kafka-statsd-metrics2	https://github.com/airbnb/kafka-statsd-metrics2.git
airbnb_lottie-android	https://github.com/airbnb/lottie-android.git
airbnb_plog	https://github.com/airbnb/plog.git

Cont. on next page

Table A.1 (cont.)

airbnb_reair	https://github.com/airbnb/reair.git
airbnb_RxGroups	https://github.com/airbnb/RxGroups.git
airbnb_SpinalTap	https://github.com/airbnb/SpinalTap.git
airbrake_airbrake-java	https://github.com/airbrake/airbrake-java.git
airbrake_javabrake	https://github.com/airbrake/javabrake.git
airbus-cyber_graylog-plugin-aggregation-count	https://github.com/airbus-cyber/graylog-plugin-aggregation-count.git
airbus-cyber_graylog-plugin-correlation-count	https://github.com/airbus-cyber/graylog-plugin-correlation-count.git
airbus-cyber_graylog-plugin-logging-alert	https://github.com/airbus-cyber/graylog-plugin-logging-alert.git
airfey_spring-drools-demo	https://github.com/airfey/spring-drools-demo.git
airicyu_Fortel	https://github.com/airicyu/Fortel.git
airlift_airlift	https://github.com/airlift/airlift.git
airlift_drift	https://github.com/airlift/drift.git
airlift_slice	https://github.com/airlift/slice.git
airminer_jnlua	https://github.com/airminer/jnlua.git
Airsaid_AndroidLocalizePlugin	https://github.com/Airsaid/AndroidLocalizePlugin.git
Airsaid_ChordView	https://github.com/Airsaid/ChordView.git
airufei_xmfcn-spring-cloud	https://github.com/airufei/xmfcn-spring-cloud.git

Cont. on next page

Table A.1 (cont.)

aisrael_jcombinatorics	https://github.com/aisrael/jcombinatorics.git
aisrael_junit-rules	https://github.com/aisrael/junit-rules.git
aistomin_jenkins-sdk	https://github.com/aistomin/jenkins-sdk.git
AITestingOrg_banking-microservices-tutorial	https://github.com/AITestingOrg/banking-microservices-tutorial.git
aitusoftware_recall	https://github.com/aitusoftware/recall.git
aivanov-ua_Paytomat-Crypto	https://github.com/aivanov-ua/Paytomat-Crypto.git
ajalt_reprint	https://github.com/ajalt/reprint.git
ajanata_PretendYoureXyzzy	https://github.com/ajanata/PretendYoureXyzzy.git
ajantis_java-crdt	https://github.com/ajantis/java-crdt.git
ajbrown_name-machine	https://github.com/ajbrown/name-machine.git
ajermakovics_backflow	https://github.com/ajermakovics/backflow.git
ajermakovics_crdts	https://github.com/ajermakovics/crdts.git
ajermakovics_eclipse-instasearch	https://github.com/ajermakovics/eclipse-instasearch.git
ajermakovics_json	https://github.com/ajermakovics/json.git
ajiang-open_jpaquery	https://github.com/ajiang-open/jpaquery.git
ajitsing_ExpenseManager	https://github.com/ajitsing/ExpenseManager.git
akoskm_bouncy-castle-sha3	https://github.com/akoskm/bouncy-castle-sha3.git
akperkins_Game-of-thrones-trivia	https://github.com/akperkins/Game-of-thrones-trivia.git

Cont. on next page

Table A.1 (cont.)

akquinet_androlog	https://github.com/akquinet/androlog.git
akquinet_needle	https://github.com/akquinet/needle.git
akquinet_vaadinator	https://github.com/akquinet/vaadinator.git
akquinet_vaangular	https://github.com/akquinet/vaangular.git
akranga_kube-workshop	https://github.com/akranga/kube-workshop.git
akraskovski_product-management-system	https://github.com/akraskovski/product-management-system.git
akraxx_gitlab-jira-integration	https://github.com/akraxx/gitlab-jira-integration.git
aksakalli_EsperIoT	https://github.com/aksakalli/EsperIoT.git
aksalj_africastalking-android	https://github.com/aksalj/africastalking-android.git
aksalj_africastalking-java	https://github.com/aksalj/africastalking-java.git
AKSW_KBox	https://github.com/AKSW/KBox.git
AKSW_LODVader	https://github.com/AKSW/LODVader.git
AKSW_Mandolin	https://github.com/AKSW/Mandolin.git
AKSW_RDFUnit	https://github.com/AKSW/RDFUnit.git
AKSW_rocker	https://github.com/AKSW/rocker.git
akubach_phyloviewer	https://github.com/akubach/phyloviewer.git
akullpp_algodat	https://github.com/akullpp/algodat.git
AKuznetsov_russianmorphology	https://github.com/AKuznetsov/russianmorphology.git

Cont. on next page

Table A.1 (cont.)

akvo_akvo-flow	https://github.com/akvo/akvo-flow.git
akvo_akvo-flow-mobile	https://github.com/akvo/akvo-flow-mobile.git
al-broco_bare-bones-digest	https://github.com/al-broco/bare-bones-digest.git
al-liu_OCat-MobilePlatform	https://github.com/al-liu/OCat-MobilePlatform.git
alaabenfatma_Diaballik	https://github.com/alaabenfatma/Diaballik.git
alabeduarte_mypodcasts-android	https://github.com/alabeduarte/mypodcasts-android.git
alaisi_nalloc	https://github.com/alaisi/nalloc.git
alaisi_postgres-async-driver	https://github.com/alaisi/postgres-async-driver.git
alalwww_SpawnChecker	https://github.com/alalwww/SpawnChecker.git
Alan-Gomes_mcspring-boot	https://github.com/Alan-Gomes/mcspring-boot.git
AlanDelip_SpringBoot-Template	https://github.com/AlanDelip/SpringBoot-Template.git
alanhay_html-exporter	https://github.com/alanhay/html-exporter.git
AlanHohn_java-intro-course	https://github.com/AlanHohn/java-intro-course.git
alansun2_happyframework-pay	https://github.com/alansun2/happyframework-pay.git
alb-i986_selenium-tinafw	https://github.com/alb-i986/selenium-tinafw.git
albertattard_gson-typeadapter-example	https://github.com/albertattard/gson-typeadapter-example.git
albertlatacz_java-repl	https://github.com/albertlatacz/java-repl.git
albertodelazzari_flink-neo4j	https://github.com/albertodelazzari/flink-neo4j.git

Cont. on next page

Table A.1 (cont.)

albertogoffi_toradocu	https://github.com/albertogoffi/toradocu.git
Albertoimpl_k8s-for-the-busy	https://github.com/Albertoimpl/k8s-for-the-busy.git
albertopastormr_greengo	https://github.com/albertopastormr/greengo.git
albertoruibal_carballo	https://github.com/albertoruibal/carballo.git
albertus82_router-logger	https://github.com/albertus82/router-logger.git
albfernandez_itext2	https://github.com/albfernandez/itext2.git
albfernandez_javadbf	https://github.com/albfernandez/javadbf.git
albfernandez_juniversalchardet	https://github.com/albfernandez/juniversalchardet.git
alblue_com.packtpub.e4	https://github.com/alblue/com.packtpub.e4.git
albogdano_elasticsearch-river-amazonsqs	https://github.com/albogdano/elasticsearch-river-amazonsqs.git
albrechtf_mcf2pdf	https://github.com/albrechtf/mcf2pdf.git
alcampos_graylog-plugin-function-csv	https://github.com/alcampos/graylog-plugin-function-csv.git
NileshJarad_TDD_Demo	https://github.com/NileshJarad/TDD_Demo.git
Nilhcem_bblfr-android	https://github.com/Nilhcem/bblfr-android.git
Nilhcem_devfestnantes-2016	https://github.com/Nilhcem/devfestnantes-2016.git
Nilhcem_devoxxfr-2016	https://github.com/Nilhcem/devoxxfr-2016.git
Nilhcem_droidconat-2016	https://github.com/Nilhcem/droidconat-2016.git
Nilhcem_droidconde-2016	https://github.com/Nilhcem/droidconde-2016.git

Cont. on next page

Table A.1 (cont.)

nimble-platform_identity-service	https://github.com/nimble-platform/identity-service.git
NimbleDroid_FriendlyDemo	https://github.com/NimbleDroid/FriendlyDemo.git
ninjaframework_ninja-rythm	https://github.com/ninjaframework/ninja-rythm.git
ninjanetworks_contacts	https://github.com/ninjanetworks/contacts.git
nipafx_demo-java-9-migration	https://github.com/nipafx/demo-java-9-migration.git
nipafx_demo-junit-5	https://github.com/nipafx/demo-junit-5.git
nipafx_java-after-eight	https://github.com/nipafx/java-after-eight.git
nipafx_JDeps-Maven-Plugin	https://github.com/nipafx/JDeps-Maven-Plugin.git
nipafx_LibFX	https://github.com/nipafx/LibFX.git
nisrulz_sensey	https://github.com/nisrulz/sensey.git
NitorCreations_DomainReverseMapper	https://github.com/NitorCreations/DomainReverseMapper.git
NitorCreations_java8-utils	https://github.com/NitorCreations/java8-utils.git
NitorCreations_javaxdelta	https://github.com/NitorCreations/javaxdelta.git
zhanggh_mtools	https://github.com/zhanggh/mtools.git

APPENDIX B

B.1. Usage of JNose and TestSmellDetector Tools

B.1.1. JNose Tool

Before installing Jnose (Source: Virgínio et al., 2020), follow the step-by-step instructions below.

- git clone https://github.com/arieslab/jnose-core
- cd jnose-core
- mvn install

Installation requirements for the Jnose Test are Java Development Kit (JDK) 1.8 and Maven 3 (or above). Installing it gives users access to Jetty (which is part of Maven), which they may use to create and run the Jnose Test. Figure B.1 displays the Jnose Test's main interface.

To run the Jnose Test, the way to follow is described:

- git clone https://github.com/arieslab/jnose
- cd jnose
- mvn jetty:run
- acessar: http://127.0.0.1:8080

Users are prompted to configure Data Input at initialization (see Figure B.2). First, as indicated in Figure B.1, Step 1, they choose the analysis mode: ClassTest, TestSmells, or Evolution. There is a field in each mode where users can enter the project path for analysis. Users have the option to change the tool's default identification of twenty-one test smell kinds (see Figure B.1, Step 2).

The tool launches Project Analysis after the execution setting is finished (Figure 3.1). It considers the analysis mode that the user has chosen, which is described as follows:

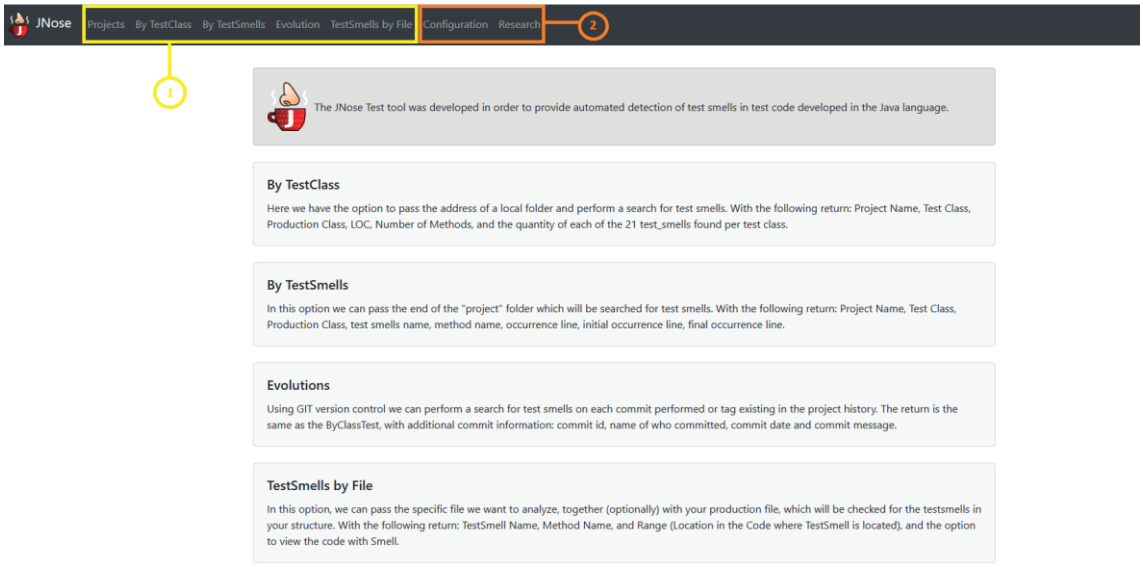


Figure B.1. Main View of the Jnose Test

(1) Project. Users can enter GitHub repository link and clone it in the project folder (see Figure B.2, Step 2 and 3).

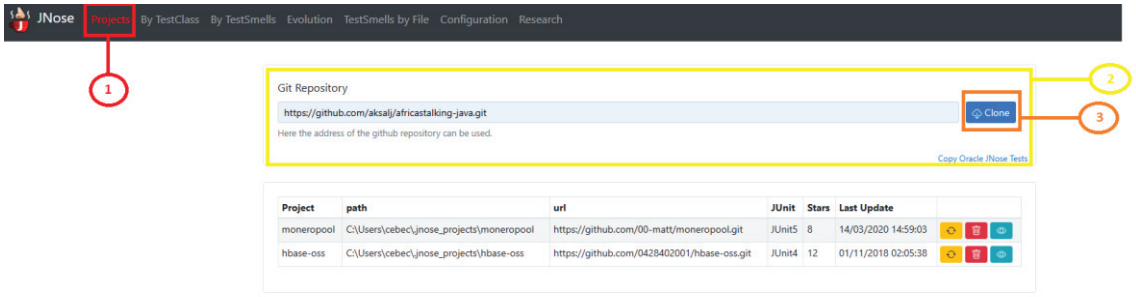


Figure B.2. Project View of the Jnose Test

(2) By ClassTest: The next step in the ProcedureAnalysis method runs the TestSmellDetection module followed by CodeCoverage module to do a project analysis at test class level. Therefore, since the calculation is performed in the local project, details related to other team members or project versions are not considered. The Data Output

procedure shows a comparative graph which demonstrates the finished analysis process according to the status of execution of the project taken from the entered GITHUB repository link (see it in Figure B.3, Step 2). The data analytic outputs of the trial class will eventually be developed into a .csv file (see Figure B.3, Step 3). A test class is represented by each row in the.csv file, which also includes five columns containing coverage data, twenty-one types of test smells, the number of test class lines, the number of test methods, and the project name and location of the test class and production class (see Figure B.4).

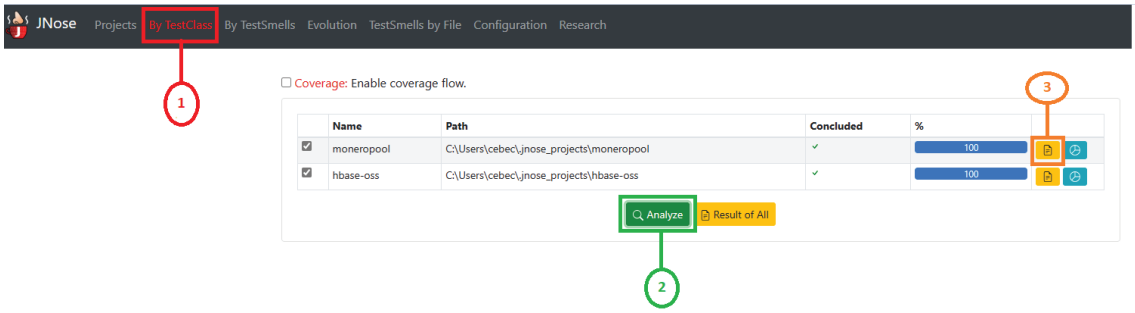


Figure B.3. View of the Execution by TestClass

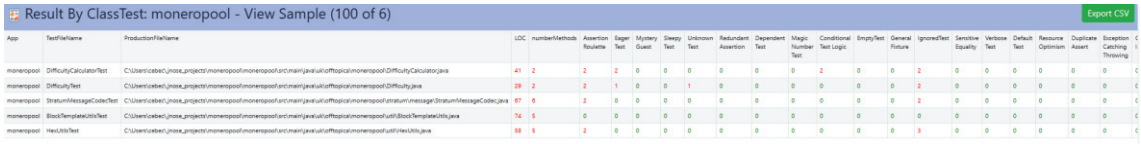


Figure B.4. Output Test Class Analysis

(3) By TestSmell: The Project Analysis procedure only uses the Test Smell Detection module when analyzing the project using test smell. As in the case of the ClassTest analysis, users must designate the projects' local routes in the Data Input stage (Figure B.2). The Data Output provides a progress view and aids in tracking progress after analyzing GITHUB repositories (Figure B.5, Step 2). The outcomes of the data analysis by test smells are then created into a.csv file (Figure B.5, Step 3). This method offers the precise location of the test smell, unlike the prior study. A test smell is represented by each row in the.csv file, which has six columns that show the type of parameter that is collected, the project name, the location of the test class and production

class, the name of the test smell, the line number of the test smell location, and the method name (see Figure B.6)

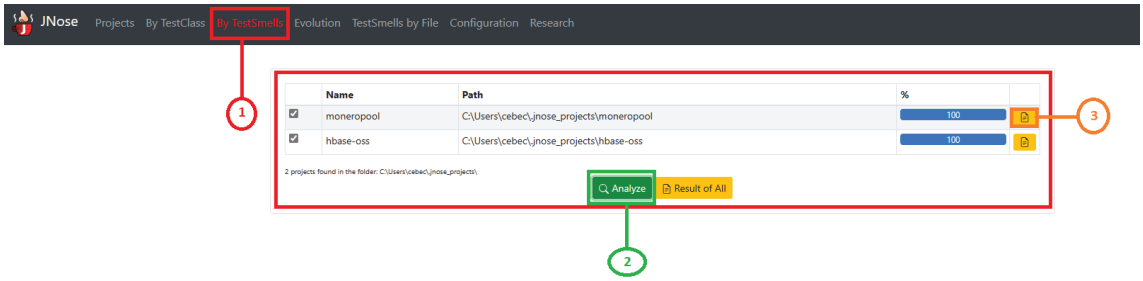


Figure B.5. View of the Execution by TestSmell

Figure B.6. Output of TestSmell Analysis

(4) Evolution: The Project Analysis method uses the Git Mining module to evaluate the project across its iterations. A cloned version of the project is used for the analysis since this module requires information about project versions. As a result, total test smells can be obtained by commits during the Data Input phase after analyzing (see Figure B.7, Step 2). When the procedure is finished, a.csv file (see Figure B.7, Step 3) with the findings of the test smells data analysis is produced (see Figure B.8).

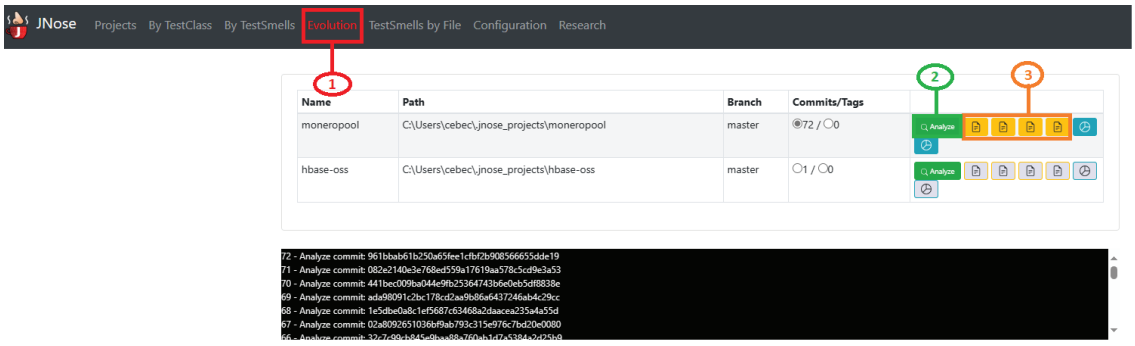


Figure B.7. View of the Execution by Evolution

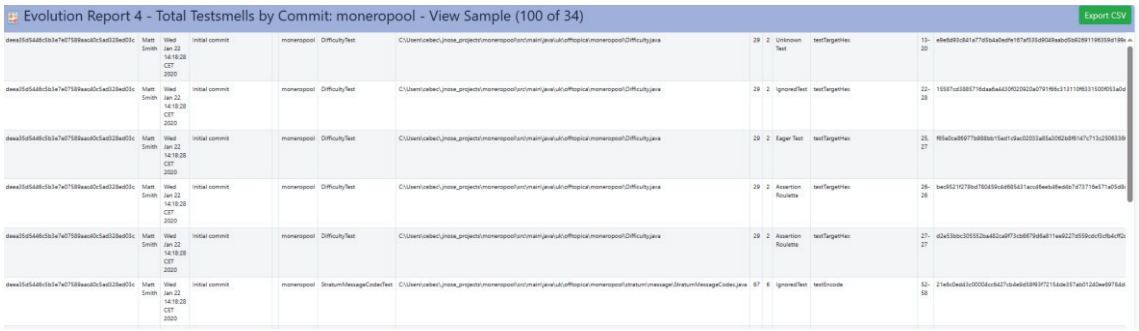


Figure B.8. Output of Evolution Analysis

In this study, the TestSmell analysis mode is used. After adding all selected GITHUB projects, output .csv files will be saved, and results can be obtained and investigated in output .csv files.

B.1.2. TestSmellsDetector Tool

A strategy design pattern is used in the creation of the test smell detection mechanism (UML class diagrams are accessible on the project website). Every smell is applied and operated separately from the others. Each sort of smell has a unique detection method that is stored within a separate module. In the future, new smell detectors can be seamlessly added thanks to this design pattern. The TestSmellDetector tool internally

calls the JavaParser library to parse the source code files. The unit test file that JavaParser is analyzing is used to create an AST. Each of the available smell detection modules then examines the AST using the established detection rules. To detect a smell, the appropriate visit() method is overridden, depending on the type of smell. To identify all test methods in the class, for instance, a MethodDeclaration visitor must first be built to detect the Redundant Print smell.

Regarding a MethodCallExpr visitor and the tracking of the methods being called inside each test method, a MethodCallExpr visitor is accordingly generated for each detected test method. The next step is to determine whether the file is smelly by comparing the name of each called method to a Java print method.

At the end of the submissions, the outcomes are saved in a Comma-Separated Values (csv) file. The preprocessed TestSmellDetector tool outputs single Boolean values for each class of smells to indicate whether the smell exists in the file or not. Because csv is technology-independent and users can easily import this format into any system, for example, SQL, to conduct data analysis, csv is considered the best option for output.

TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\CustomUrlRequestTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\CustomUrlRequest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\SolarEventCalculatorTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\SolarEventCalculator.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\SystemStateTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\SystemState.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\Space3DTransformer.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\Space3DTransformer.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\LoginActivityTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\LoginActivityTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\LoggingTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\LoggingTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\RLPTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\RLPTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\ResultActivityTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\ResultActivityTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\RetrofitApiClientTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\RetrofitApiClientTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\NmeaSentenceTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\NmeaSentenceTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\LastPassParserTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\LastPassParserTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\AbrissTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\AbrissTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\XmlSanitizerTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\XmlSanitizerTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\EventsScraperTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\EventsScraperTest.java
TestSmellDetector	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\GitAsyncTaskTest.java	D:\Master_Thesis\TestSmellDetector\source_and_test_codes\GitAsyncTaskTest.java

Figure B.9. Input .csv file format of TestSmellDetector Tool

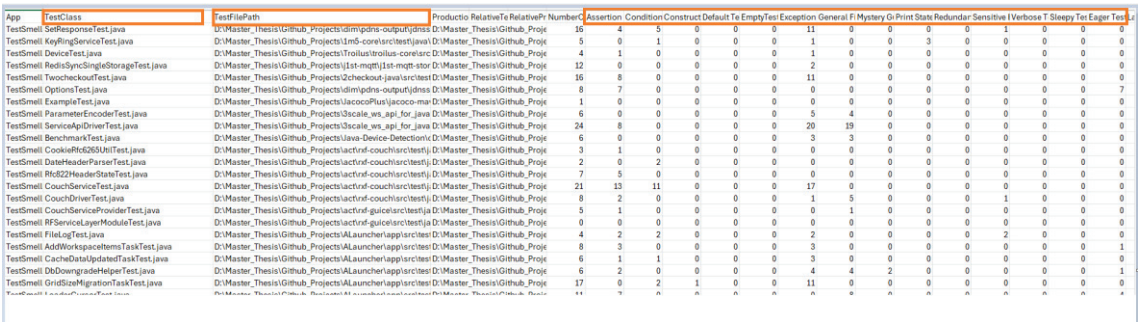
Test File Detection: The JUnit naming convention advises developers to prefix or attach the word "Test" to the production file name that must be tested (e.g., Test*.java and *Test.java) as shown in Figure B.9. First, our tool finds all ".java" files whose filename contains the term "test" at the beginning or end. The AST of each recognised Java source file is then processed using JavaParser. This strategy accomplishes two goals: First, it lets us remove Java files that have syntax issues. Secondly, it lets us determine with precision whether the file has JUnit-based unit test methods. A method that contains a unit test must have a public access modifier, be named with 'test' (JUnit 3) or be annotated with @Test (JUnit 4).

Production File Detection: Certain test smells, like the Lazy and Eager tests, require that the production file associated with the unit test file be identified. The project structure searches for files with the same name as the test file but without the term "test" to find the production file. The program creates an AST for every production file it finds to confirm that the file is syntactically accurate.

Use of TestSmellDetector tool: The following command can be used to run TestSmellDetector tool as a command line tool:

java -jar <path_to_test_files>.\TestSmellDetector.jar

The TestSmellDetector tool doesn't need the user to interact again once it's started. A csv file containing the detection process's results will be generated and sent back as output once the process has finished as shown in Figure B.10.



App	TestClass	TestFilePath	Productio	RelativeTe	RelativePr	NumberC	Assertion	Condition	Construct	Default Te	EmptyTest	Exception	General Fi	Mystery Gi	Print State	Redundar	Sensitive I	Verbose T	Sleepy Tes	Eager Test	Lz
TestSmell	SetResponseTest.java	D:\Master_Thesis\Github_Projects\dim\pdns-output\pdns	D:\Master_Thesis\Github_Proje			16	4	5	0	0	0	11	0	0	0	0	1	0	0	0	
TestSmell	KeyRingServiceTest.java	D:\Master_Thesis\Github_Projects\1m5-core\src\test\java\	D:\Master_Thesis\Github_Proje			5	0	1	0	0	0	1	0	0	3	0	0	0	0	0	
TestSmell	DeviceTest.java	D:\Master_Thesis\Github_Projects\Troilus\troilus-core\src	D:\Master_Thesis\Github_Proje			4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	
TestSmell	RedisSyncSingleStorageTest.java	D:\Master_Thesis\Github_Projects\j1st-mqtt\j1st-mqtt-stor	D:\Master_Thesis\Github_Proje			12	0	0	0	0	0	2	0	0	0	0	0	0	0	0	
TestSmell	TwocheckoutTest.java	D:\Master_Thesis\Github_Projects\2checkout-java\src\test	D:\Master_Thesis\Github_Proje			16	8	0	0	0	0	11	0	0	0	0	0	0	0	0	
TestSmell	OptionsTest.java	D:\Master_Thesis\Github_Projects\dim\pdns-output\pdns	D:\Master_Thesis\Github_Proje			8	7	0	0	0	0	0	0	0	0	0	0	0	0	7	
TestSmell	ExampleTest.java	D:\Master_Thesis\Github_Projects\JacocoPlus\jacoco-ma	D:\Master_Thesis\Github_Proje			1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
TestSmell	ParameterEncoderTest.java	D:\Master_Thesis\Github_Projects\3scale_ws_api_for_java	D:\Master_Thesis\Github_Proje			6	0	0	0	0	0	5	4	0	0	0	0	0	0	0	
TestSmell	ServiceApiDriverTest.java	D:\Master_Thesis\Github_Projects\3scale_ws_api_for_java	D:\Master_Thesis\Github_Proje			24	8	0	0	0	0	20	19	0	0	0	0	0	0	0	
TestSmell	BenchmarkTest.java	D:\Master_Thesis\Github_Projects\Java-Device-Detection\v	D:\Master_Thesis\Github_Proje			6	0	0	0	0	0	3	3	0	0	0	0	0	0	0	
TestSmell	CookieRfc6265UtilTest.java	D:\Master_Thesis\Github_Projects\act\rxf-couch\src\test\j	D:\Master_Thesis\Github_Proje			3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	
TestSmell	DateHeaderParserTest.java	D:\Master_Thesis\Github_Projects\act\rxf-couch\src\test\j	D:\Master_Thesis\Github_Proje			2	0	2	0	0	0	0	0	0	0	0	0	0	0	0	
TestSmell	Rfc822HeaderStateTest.java	D:\Master_Thesis\Github_Projects\act\rxf-couch\src\test\j	D:\Master_Thesis\Github_Proje			7	5	0	0	0	0	0	0	0	0	0	0	0	0	0	
TestSmell	CouchServiceTest.java	D:\Master_Thesis\Github_Projects\act\rxf-couch\src\test\j	D:\Master_Thesis\Github_Proje			21	13	11	0	0	0	17	0	0	0	0	0	0	0	0	
TestSmell	CouchDriverTest.java	D:\Master_Thesis\Github_Projects\act\rxf-couch\src\test\j	D:\Master_Thesis\Github_Proje			8	2	0	0	0	0	1	5	0	0	0	1	0	0	0	
TestSmell	CouchServiceProviderTest.java	D:\Master_Thesis\Github_Projects\act\rxf-guice\src\test\ja	D:\Master_Thesis\Github_Proje			5	1	0	0	0	0	0	1	0	0	0	0	0	0	0	
TestSmell	RFServiceLayerModuleTest.java	D:\Master_Thesis\Github_Projects\act\rxf-guice\src\test\ja	D:\Master_Thesis\Github_Proje			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
TestSmell	FileLogTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test	D:\Master_Thesis\Github_Proje			4	2	2	0	0	0	2	0	0	0	0	2	0	0	0	
TestSmell	AddWorkspaceItemsTaskTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test	D:\Master_Thesis\Github_Proje			8	3	0	0	0	0	3	0	0	0	0	0	0	0	1	
TestSmell	CacheDataUpdatedTaskTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test	D:\Master_Thesis\Github_Proje			6	1	1	0	0	0	3	0	0	0	0	0	0	0	0	
TestSmell	DbDowngradeHelperTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test	D:\Master_Thesis\Github_Proje			6	2	0	0	0	0	4	4	2	0	0	0	0	0	1	
TestSmell	GridSizeMigrationTaskTest.java	D:\Master_Thesis\Github_Projects\ALauncher\app\src\test	D:\Master_Thesis\Github_Proje			17	0	2	1	0	0	11	0	0	0	0	0	0	0	0	

Figure B.10. Output .csv file of TestSmellDetector Tool

APPENDIX C

C.1. Outputs of JNose and TestSmellDetector Tools

Table C.1. Total number of test smells with using JNose and TestSmellDetector tools in all files

Test Smells	Jnose Tool	TestSmellDetector Tool	Both Tool
Magic Number Test	11264	28443	39707
Assertion Roulette	41876	10488	52364
Conditional Test Logic	3679	1948	5627
Constructor Initialization	178	0	178
Default Test	0	1	1
Empty Test	215	116	331
Exception Catching Throwing	3236	13612	16848
General Fixture	995	4274	5269
Mystery Guest	621	730	1351
Print Statement	1051	534	1585
Redundant Assertion	143	160	303
Sensitive Equality	1490	906	2396
Verbose Test	1947	0	1947
Sleepy Test	186	175	361
Eager Test	3692	3780	7472

Cont. on next page

Table A.1 (cont.)

Lazy Test	3984	16570	20554
Duplicate Assert	2416	2262	4678
Unknown Test	3202	3651	6853
Ignored Test	916	1152	2068
Resource Optimism	682	695	1377
Dependent Test	0	0	0
Total	81773	89497	171270

Table C.2. Ratios of test smells by using each tool in all files

Test Smells	Jnose Tool	TestSmellDetector Tool	Both Tool
Magic Number Test	%13.77	%31.78	%23.18
Assertion Roulette	%51.21	%11.72	%30.57
Conditional Test Logic	%4.5	%2.18	%3.29
Constructor Initialization	%0.22	%0	%0.10
Default Test	%0	%0	%0
Empty Test	%0.26	%0.13	%0.19
Exception Catching Throwing	%3.96	%15.21	%9.84
General Fixture	%1.22	%4.78	%3.08
Mystery Guest	%0.76	%0.82	%0.79

Cont. on next page

Table C.2 (cont.)

Print Statement	%1.29	%0.60	%0.93
Redundant Assertion	%0.17	%0.18	%0.18
Sensitive Equality	%1.82	%1.01	%1.40
Verbose Test	%2.38	%0	%1.14
Sleepy Test	%0.23	%0.20	%0.21
Eager Test	%4.51	%4.22	%4.36
Lazy Test	%4.87	%18.51	%12.00
Duplicate Assert	%2.95	%2.53	%2.73
Unknown Test	%3.92	%4.08	%4.0
Ignored Test	%1.12	%1.29	%1.21
Resource Optimism	%0.83	%0.78	%0.80
Dependent Test	%0	%0	%0

Table C.3. Number of affected files by each test smells

Test Smells	Jnose Tool	TestSmellDetector Tool
Magic Number Test	1364	4222
Assertion Roulette	3056	2503
Conditional Test Logic	697	724
Constructor Initialization	169	0

Cont. on next page

Table C.3 (cont.)

Default Test	0	0
Empty Test	132	48
Exception Catching Throwing	723	2463
General Fixture	419	505
Mystery Guest	196	238
Print Statement	213	222
Redundant Assertion	57	85
Sensitive Equality	318	370
Verbose Test	773	0
Sleepy Test	70	80
Eager Test	905	1126
Lazy Test	1396	1070
Duplicate Assert	612	868
Unknown Test	969	1030
Ignored Test	233	322
Resource Optimism	241	265
Dependent Test	0	0
Total	12543	16141

Table C.4. Ratios of affected files by each test smells

Test Smells	Jnose Tool	TestSmellDetector Tool
Magic Number Test	%24.90	%77.07
Assertion Roulette	%55.79	%45.69
Conditional Test Logic	%12.72	%13.22
Constructor Initialization	%3.09	%0
Default Test	%0	%0.02
Empty Test	%2.41	%0.88
Exception Catching Throwing	%13.20	%44.96
General Fixture	%7.65	%9.22
Mystery Guest	%3.58	%4.34
Print Statement	%3.89	%4.05
Redundant Assertion	%1.04	%1.55
Sensitive Equality	%5.81	%6.75
Verbose Test	%14.11	%0
Sleepy Test	%1.28	%1.46
Eager Test	%16.52	%20.55
Lazy Test	%25.48	%19.53
Duplicate Assert	%11.17	%15.85
Unknown Test	%17.69	%18.8
Ignored Test	%4.25	%5.88
Resource Optimism	%4.40	%4.84
Dependent Test	%0	%0