

Understanding the Prevalence of Test Smells in Open-source and Industrial Software: An Empirical Study on Python Projects

Md Arif Hasan, Toukir Ahammed

Institute of Information Technology, University of Dhaka, Dhaka, Bangladesh

Abstract

Test smells, a type of anti-pattern, refer to poorly designed tests that can undermine the effectiveness, maintainability, and overall quality of test suites, as well as the reliability of production code. Test smells have been extensively investigated in the literature for open-source projects from different viewpoints such as detection, analysis, impact on maintainability and quality issues. However, whether the occurrence of test smells differs between open-source and industrial projects has not yet been investigated. This study aims to bridge that gap by investigating the types, occurrence and distribution of test smells in both open-source and industrial projects. Moreover, this study explores whether the volume of test code differs between open-source and industrial projects to gain insights into the test-writing practices in both environments. For this purpose, 64 open-source and 11 industrial Python projects were analyzed to detect and examine the prevalence of 18 distinct test smells. 14 types of test smells were identified in open-source projects, averaging 601.58 smells per project, while 10 types were identified in industrial projects, with an average of 48.72 smells per project. Assertion Roulette and Conditional Logic Test are the two most frequently occurring test smells observed in both environments. The results of the study indicate that test smells occur more frequently in open-source projects compared to industrial projects, as confirmed by one-tailed t-test, while a two-tailed t-test revealed no significant difference in the volume of test code between the two environments. These findings suggest that open-source projects and industrial projects have differences in maintaining clean and high-quality test code. Future work should explore how specific development practices contribute to this difference across both project types.

Keywords

Test Smells, Python Testing, Industrial Projects, Open Source Projects, Test Code Quality

1. Introduction

Test smells are suboptimal patterns in test code that can negatively impact the effectiveness [1] of software testing by reducing test maintainability, reliability, and eventually hampering software quality [2, 3, 4, 5]. Developers usually introduce test smells due to less awareness in design and program comprehensibility [6]. Test smells can occur in both categories of software development environments—industrial and open-source projects [7, 8]. Due to time, budget constraints, and software release pressure in the industry, systems are developed in a different setting compared to open-source projects. Additionally, open-source software grows more quickly than industrial projects which could have an impact on the occurrences of test smells and test writing practices between these two contexts [9].

Test smells were initially introduced to highlight issues in test code within software systems [10], but later they have been used as a measure of quality of software projects. Researchers found that test smells are associated with software quality [2, 11], fault-proneness [12, 13], and maintainability issues in software systems [2, 14, 15]. Although there has been extensive research on test smells in open-source projects, to the best of our knowledge, no studies have focused on test smell detection across both open-source and industrial projects. Identifying and addressing frequent test smells in both contexts can significantly improve software maintainability and reliability. This leads to the following research questions:

- **RQ1:** What types of test smells are prevalent in open-source and industrial Python projects?
- **RQ2:** How does the prevalence of test smells differ between open-source and industrial Python projects?

- **RQ3:** How does the volume of test code differ between open-source and industrial Python projects?

To address these research questions, an empirical analysis was conducted on 75 Python projects, including 64 popular open-source projects and 11 industrial projects. Open-source projects have been selected based on criteria such as consistent development activity, unique project purpose, and substantial contributions. On the other hand, industrial projects were selected using convenience sampling [4, 16] from software companies based on data accessibility and industry willingness to share information. The selected industrial projects have been developed following the standard Software Development Life Cycle (SDLC) model and are currently used by clients. A set of 18 test smells has been chosen for the investigation. To detect these test smells, PyNose [17] - a popular test smell detection tool for Python test code, is used.

After detecting test smells, frequent test smells are identified and assessed whether the occurrence of test smells differs between open source and industrial projects. The results of the study show that 14 types of test smells were found in open-source projects and 10 types of test smells in industrial projects. Assertion Roulette (AR) is the most common test smell found in both open-source and industrial projects, indicating widespread issues in assertion practices across environments. In industrial projects, the top 3 occurring test smells are Assertion Roulette (AR), Magic Number Test (MNT), and Conditional Test Logic (CTL). Meanwhile, in open-source projects, the top three are Assertion Roulette (AR), Conditional Test Logic (CTL), and Duplicate Assertion (DA). This finding highlights the distinct patterns of test smell occurrence in open-source and industrial projects.

Our analysis shows that test smells are more prevalent and concentrated in open-source projects, with an average of 601.58 test smells per project compared to 48.72 per project in industrial projects. Assertion Roulette (AR) is the most widespread test smell in both environments, though it

QuASoQ 2024: 12th International Workshop on Quantitative Approaches to Software Quality, December 03, 2024, Chongqing, China

✉ bsse1112@iit.du.ac.bd (M. A. Hasan); toukir@iit.du.ac.bd

(T. Ahammed)



© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

appears in 92% of open-source projects and 81% of industrial projects. On average, Assertion Roulette occurs for every 262 lines of test code in industrial projects and every 212 lines in open-source projects. Statistical analysis conducted using a one-tailed t-test shows that 8 out of 14 test smells occur more frequently in open-source projects compared to industrial projects, even though the volume of test code is not significantly higher in open-source projects. For the remaining 6 test smells, the analysis showed no significant difference in their frequency between open-source and industrial projects. Additionally, two-tailed hypothesis testing supports this finding, indicating minimal differences in test code volume across environments, with 6 out of 7 null hypotheses accepted.

The rest of the paper is organized as follows. Section 2 discusses existing work in the field of test smell analysis and detection. Section 3 describes the methodology used to design this empirical study. Section 4 presents the results and provides a detailed analysis based on the research questions. In Section 5, we address the threats to the validity of our study, and we conclude in Section 6.

2. Related Work

In recent years, test smells have become a crucial area of research in software engineering. Studies have focused on the automated detection, mitigation, and impact evaluation of test smells on software quality. Early work aimed to define various types of test smells [10], while more recent studies have explored detection techniques [15, 18] and the development of specialized tools for detecting test smells across different programming languages, such as Java, C#, Python, and Scala [19, 20, 17, 21]. Furthermore, several studies have investigated the evolution of test smells and their influence on software maintenance and quality [3, 22, 2, 23, 14].

The concept of test smells was introduced by Van Deursen et al. [10], who defined them as signs of poor practices in test code. They identified 11 test smells, which laid the groundwork for understanding how suboptimal testing can impact software quality and maintenance. Expanding on this, Hauptmann et al. [24] introduced Natural Language Test Smells (NLTS) to address issues related to the maintainability, readability, and efficiency of manual test cases. Soares et al. [25] further refined this idea by cataloging eight NLTS, validated through empirical research and natural language processing (NLP) analysis. Additionally, the Open Catalog of Test Smells offers a comprehensive resource by consolidating information from 127 sources to support the identification and understanding of test smells in various contexts [26].

In recent years, specialized tools have emerged to aid in the identification and mitigation of test smells. Initially, research concentrated on statically typed languages like Java and Scala. Virginio et al. [20] developed JNose, a tool for detecting 21 types of test smells in Java, followed by tsDetect [19], which detects 19 types. Later, tools for Python, including pytest-smell [21] and PyNose [17], were introduced, with PyNose also introducing the "suboptimal assert" test smell. Pontillo et al. [18] proposed machine learning-based detection methods, while Palomba et al. [15] expanded test smell detection to automatic techniques using information retrieval methods. More recently, Lucas et al. [27] explored the use of large language models (LLMs) like

ChatGPT-4, Mistral Large, and Gemini Advanced for test smell detection, demonstrating the potential of LLMs in this domain.

Test smells are often linked to flaky tests, which produce inconsistent results and undermine test reliability. Garousi et al. [7] and Martins et al. [11] found that certain test smells, such as external dependencies and non-deterministic behavior, contribute to flaky tests, masking genuine defects. Palomba and Zaidman [28] identified key smells that frequently co-occur with flaky tests, while Camara et al. [29] showed that incorporating test smells into flaky test prediction models outperforms traditional vocabulary-based approaches. These findings underscore the importance of mitigating test smells to enhance software reliability.

Further research has explored the impact of test smells on software quality, maintainability, and reliability. Spadini et al. [2, 6] investigated the relationship between test smells and software quality, while Kim et al. [23] examined how test smells evolve and affect software maintenance. Qusef et al. [12] established a connection between test smells and fault-proneness, indicating their potential negative impact on software reliability.

In the context of open-source versus industrial projects, Fushihara et al. [30] showed that test smells persist and evolve over time, particularly in open-source projects. The differences between these two environments are often attributed to varying levels of emphasis on test quality. Recent studies have also examined how test smells and refactoring are discussed within online communities like Stack Exchange, revealing insights from both industry and academia [11].

While previous research has explored test smells in various software environments, there is a notable lack of comparative studies focusing on how these smells manifest in open-source versus industrial projects. Given the differences in development practices and priorities, understanding these variations is crucial for tailoring detection techniques and improving test quality. To address this gap, we conduct an empirical analysis. This study investigates the types of test smells prevalent in both open-source and industrial projects. It also examines how the prevalence of these smells differs between the two environments, as well as how the volume of test code compares across them.

3. Methodology

This study aims to understand the occurrences, types and the distribution of test smells in open-source and industrial projects. It also seeks to assess the extent to which the distribution of test smells differs between open-source and industrial projects. To achieve this, the study analyzes 64 open-source projects and 11 industrial projects, all written in Python. Test smells within these projects are identified using a tool named PyNose¹. Then, the collected data are analyzed using the metrics Spreadity [31] and Inverse Smell Frequency (ISFi) [32]. Finally, statistical hypothesis tests are conducted to evaluate differences in test smell frequency and test code volume between open-source and industrial projects. The overall methodology is described in detail in the following subsections.

¹<https://github.com/JetBrains-Research/PyNose.git>

Table 1
Production Code Metrics

Projects Category	Number of Projects	KLOC		NOM		NOC	
		Total	Average	Total	Average	Total	Average
Open-Source Project	64	18012	281	603118	9428	145659	2276
Industrial Project	11	2592	236	86045	7822	23355	2123

Table 2
Test Code Metrics

Projects Category	Number of Projects	Test KLOC		Test NOM		NOC	
		Total	Average	Total	Average	Total	Average
Open Source Project	64	2882	45	317810	4966	23730	371
Industrial Project	11	62	6	7422	675	650	59

3.1. Dataset Collection

A dataset comprising 75 extensive Python projects—64 open-source projects and 11 industrial projects—serves as the foundation for our analysis. The open-source projects were chosen based on specific criteria: they must have more than 50 stars on GitHub, not be forked, have over 1,000 commits, include at least 10 contributors, possess a development history spanning more than two years, be actively maintained within the last two years, and primarily use Python as their programming language.

In selecting industrial project, convenience sampling [4] strategy was employed. This non-random technique was used because data collection on industrial projects largely depends on the accessibility, willingness, and readiness of software companies to share their data. In this study, 11 industrial projects were collected from different software companies of Bangladesh. Selection criteria included a minimum project size of 6,000 lines of code (LOC), a minimum of 500 commits to ensure development activity, and comprehensive test code (with at least 1,000 lines of test code).

The dataset was carefully curated to represent a diverse range of testing practices, providing a robust foundation for comparing test smells in both open-source and industrial projects. Key attributes, such as lines of code (KLOC), number of methods (NOM), and number of classes (NOC), are detailed in Table 1 and Table 2. These tables offer a clear overview of the projects' core characteristics, presented without any initial analysis. The complete dataset used in this study is available in the online appendix.²

3.2. Test Smell Detection

Detecting test smells in Python typically involves a static code analysis process that inspects the structure of test code to identify patterns known to impact test maintainability, readability, and reliability. This approach systematically scans Python test classes—particularly those derived from the `unittest` framework—focusing on elements such as assertion statements, control structures, and setup and teardown methods. By examining these components, static analysis tools can identify common test smells like Assertion Roulette, Eager Test, and Test Duplication, which complicate debugging and future modifications. Consider the code snippet in Listing 1, which provides an example of a common test smell.

Listing 1: Test Smell Example

```
import unittest

class ExampleTest(unittest.TestCase):
    def test_values(self):
        result = get_values()
        self.assertEqual(result[0], 10)
        self.assertTrue(result[1] > 5)
        self.assertEqual(result[2], "expected_value")

if __name__ == '__main__':
    unittest.main()
```

In this example, the test contains multiple assertions without descriptive messages, which is a classic case of the "Assertion Roulette" test smell. This occurs when multiple assertions are made without context, making it difficult to diagnose failures. Such smells are detected using rule-based methods, as described by Aljedaani et al. [33], which identify tests with multiple assertions lacking clarity. Metrics like cosine similarity can also highlight a "Lack of Cohesion" in the test case.

The methodology uses predefined thresholds for various metrics, such as local variables and hard-coded values, based on best practices in software testing. This structured approach identifies issues like redundant setups, poorly named tests, and excessive complexity. For our study, we utilize PyNose [17], which is effective in detecting a broad range of test smells, as demonstrated in previous research [34, 35, 36].

3.3. Test Smell Analysis

To systematically analyze the distribution of test smells within our dataset, we use two key metrics: Spreadity and Inverse Smell Frequency (ISFi) [32], along with statistical hypothesis testing techniques to examine the differences in test smell occurrences between open-source and industrial projects. These metrics are commonly used to evaluate the distribution and frequency of smells in software research [31].

Spreadity: To measure the presence of test smells across the analyzed projects, we utilize the metric called Spreadity as shown in Equation 1. Spreadity calculates the proportion of projects that contain a specific test smell, providing insight into how common a particular smell is within the dataset.

$$\text{Spreadity}(x) = \frac{n(x)}{N} \quad (1)$$

where x denotes the test smell, $n(x)$ is the number of projects containing the smell x , and N represents the total number of projects analyzed. A higher Spreadity value

²<https://figshare.com/s/4789bd212185042cdad0>

indicates that a larger proportion of projects are affected by the test smell, with a maximum value of 1 if all projects contain the smell.

Inverse Smell Frequency (ISFi): To gain a deeper understanding of the occurrence of test smells, we calculate the Inverse Smell Frequency (ISFi) metric using the Equation 2. This metric is used to analyze code smell frequency and occurrences [31]. It normalizes the frequency of a test smell relative to specific software metrics, such as Test Lines of Code (TLOC), Test Number of Methods (TNOM), and Test Number of Classes (TNOC).

$$\text{ISFi}(\text{metric}) = \frac{\text{Metric Value}}{\text{Smell Count}_i} \quad (2)$$

where the metric values could be TLOC, TNOM, or TNOC, and Smell Count_i represents the occurrence of a particular test smell i . Lower ISFi values indicate a higher frequency of the test smell, suggesting that it is more prevalent in the code.

Statistical Tests of Hypothesis: The differences in test smell occurrences between open-source and industrial projects are examined using t-tests, a widely accepted method for comparing the means of two independent groups.

A one-tailed t-test is performed to determine if there are significant differences in the prevalence of specific test smells between open-source and industrial projects. The one-tailed approach is used because we hypothesize that the occurrence of test smells will be higher in open-source projects compared to industrial projects, based on previous observations of faster development cycles and less formal testing practices in open-source environments. The alternative hypothesis suggests that the occurrence of test smells is higher in industrial projects, while the null hypothesis assumes no significant difference. The test is conducted at a 5% significance level.

Additionally, a two-tailed t-test was used to assess whether there were any significant differences in the volume of test code between open-source and industrial projects. In this analysis, the alternative hypothesis proposes a difference in test code volume, while the null hypothesis assumes no difference. The test was conducted at a 5% significance level, with specific metrics analyzed to determine any significant variations between the two environments.

4. Result Analysis

This section presents the findings from our study, addressing each research question by analyzing the occurrences of test smells in industrial and open-source projects. We highlight the types of test smells, their frequency, differences in occurrence between environments, and insights into test-writing practices in the software industry.

4.1. RQ1: What types of test smells are prevalent in open-source and industrial Python projects?

To address RQ1, we analyze the presence of test smells in both open-source and industrial projects. Our study identifies 14 distinct test smells across these environments, with notable differences in their presence and frequency, as shown in Table 4.

Table 3

Test Smell Occurrences in Industrial and Open-Source Projects

Test Smell	Number of affected projects	
	Industrial	Open source
Assertion Roulette	9	59
Conditional Test Logic	7	59
Duplicate Assertion	6	55
Exception Handling	4	51
Magic Number Test	8	49
Obscure In-Line Setup	4	40
Redundant Assertion	4	31
Redundant Print	2	29
Test Maverick	-	19
Sleepy Test	-	17
Empty Test	1	17
Lack of Cohesion	-	7
Suboptimal Assert	2	7
Constructor Initialization	-	2

Assertion Roulette (AR) emerges as the most frequently occurring test smell in both open-source and industrial projects based on the Table 3. It is occurred in 9 out of 11 industrial and 59 out of 64 open-source projects. This indicates a widespread issue with AR in both environments, suggesting a need for better assertion practices in test code. In open-source projects, the top 3 test smells based on project count are *Assertion Roulette (AR)*, *Conditional Test Logic (CTL)*, and *Duplicate Assertion (DA)*, which are consistently present and indicative of complex assertion patterns in open-source projects.

However, industrial projects display a slightly different ranking of test smells. When sorted by the number of projects in which they are found, the order is *Assertion Roulette (AR)* > *Magic Number Test (MNT)* > *Conditional Test Logic (CTL)*. In contrast, when ranked by their total occurrences (frequency) in the open-source projects, as shown in Table 4, the order shifts to *Assertion Roulette (AR)* > *Conditional Test Logic (CTL)* > *Magic Number Test (MNT)*. This variation highlights that while some test smells like AR are universally problematic, other smells vary in significance depending on the environment, indicating different testing practices or priorities between industrial and open-source projects.

The dominance of test smells like AR and CTL highlights common weaknesses in both open-source and industrial projects, pointing to the need for improved testing strategies that prioritize clarity and simplicity in test code. Identifying and addressing these frequent test smells can significantly enhance the maintainability and reliability of software projects in both environments.

Moreover, we identify a few test smells that are entirely absent in both open-source and industrial projects. These test smells include *Default Test (DT)*, *General Fixture (GF)*, *Ignored Test (IT)*, and *Unknown Test (UT)*. Additionally, four specific test smells—*Constructor Initialization (CI)*, *Lack of Cohesion of Test Cases (LCTC)*, *Sleepy Test (ST)*, and *Test Maverick (TM)*—are present exclusively in open-source projects and are completely absent from industrial projects. This absence indicates a divergence in testing practices between these environments, with certain test smells being specific to open-source development contexts.

Table 4
Comparison of Test Smell Occurrences between Industrial and Open-source Projects

Test Smell	Industrial Projects						Open-source Projects						p-value
	Total Freq.	Avg Freq./ Proj.	Spreadity	ISFi(TLOC)	ISFi(TNOM)	ISFi(TNOC)	Total Freq.	Avg Freq./ Proj.	Spreadity	ISFi(TLOC)	ISFi(TNOM)	ISFi(TNOC)	
Assertion Roulette(AR)	235	21.36	0.82	262	32	3	13607	212.61	0.92	212	23	2	0.068
Conditional Test Logic(CTL)	115	10.45	0.64	536	65	6	10136	158.38	0.92	284	31	2	0.002
Constructor Initialization(CI)	0	0	0.00	NA	NA	NA	130	2.03	0.03	22167	2445	183	0.119
Duplicate Assertion(DA)	51	4.64	0.55	1209	146	13	6185	96.64	0.86	466	51	4	0.0002
Empty Test(ET)	3	0.27	0.09	20561	2474	217	106	1.66	0.27	27186	2998	224	0.082
Exception Handling(EH)	10	0.91	0.36	6168	742	65	1494	23.34	0.80	1929	213	16	0.0001
Lack of Cohesion of Test Cases(LCTC)	0	0	0.00	NA	NA	NA	544	8.50	0.11	5297	584	44	0.010
Magic Number Test(MNT)	72	6.55	0.73	857	103	9	3058	47.78	0.77	942	104	8	0.156
Obscure In-Line Setup(OIS)	5	0.45	0.36	12337	1484	130	669	10.45	0.62	4308	475	35	0.049
Redundant Assertion(RA)	16	1.45	0.36	3855	464	41	114	1.78	0.48	25278	2788	208	0.178
Redundant Print(RP)	4	0.36	0.18	15421	1856	163	542	8.47	0.45	5317	586	44	0.036
Sleepy Test(ST)	0	0	0.00	NA	NA	NA	240	3.75	0.27	12007	1324	99	0.012
Suboptimal Assert(SA)	25	2.27	0.18	2467	297	26	744	11.63	0.11	3873	427	32	0.129
Test Maverick(TM)	0	0	0.00	NA	NA	NA	898	14.03	0.30	3209	354	26	0.001

4.2. RQ2: How does the prevalence of test smells differ between open-source and industrial Python projects?

The aim of RQ2 is to identify the most frequent and widespread test smells that significantly affect both industrial and open-source projects. Our analysis reveals that test smells are generally more frequent in open-source projects, with an average frequency of **601.57** test smells per project compared to **48.72** in industrial projects.

Assertion Roulette (AR) is observed as the most widespread test smell in both open-source and industrial projects, with spreadity values of **0.92** and **0.81**, depicted in Figure 1. Similarly, *Conditional Test Logic (CTL)* has a spreadity of **0.92** in open-source projects but only **0.63** in industrial projects. While *Magic Number Test (MNT)* ranks as the second most frequent smell in industrial projects, its position drops to fifth in open-source projects, despite having a slightly higher spreadity in open-source (**0.76**) than in industrial environments (**0.72**).

Certain test smells, such as *Test Maverick (TM)*, *Sleepy Test (ST)*, *Lack of Cohesion of Test Cases (LCTC)*, and *Constructor Initialization (CI)*, are found exclusively in open-source projects and are completely absent from industrial projects. This indicates a clear divergence in test practices, with these smells being specific to the open-source context.

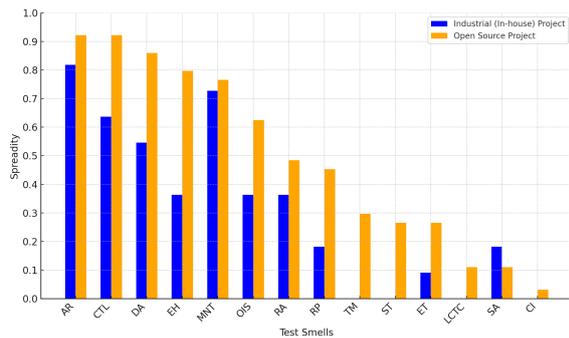


Figure 1: Spreadity of each test smell

The analysis reveals that test smells such as *Assertion Roulette (AR)* and *Conditional Test Logic (CTL)* are notably more prevalent in open-source projects compared to industrial projects. This conclusion is based on both the spread and average frequency metrics, as shown in Figure 2. Additionally, Figures 3, 4, and 5 illustrate the ISFi scores for TLOC, TNOM, and TNOC across all the test smells under study. The four common smells that did not appear in any system are excluded, as discussed in response to RQ1.

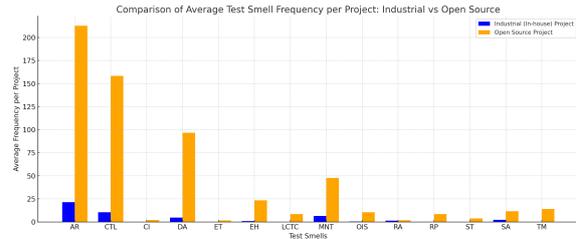


Figure 2: Average frequency of each test smell

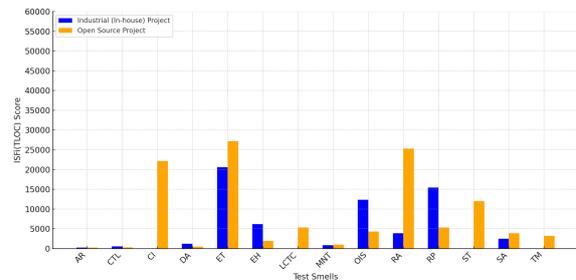


Figure 3: ISFi(TLOC) Score Comparison for each test smell

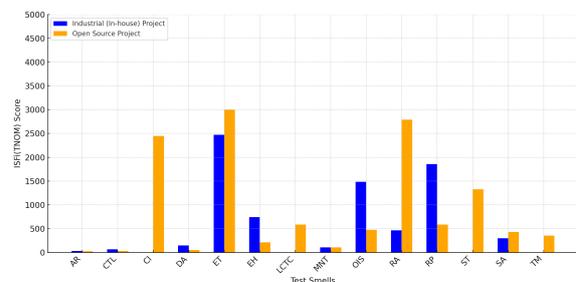


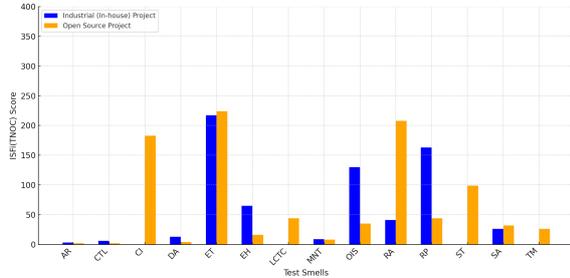
Figure 4: ISFi(TNOM) Score Comparison for each test smell

The results suggest that *Assertion Roulette* occurs in industrial projects for every 262 test lines of code, and in open-source projects for every 212 test lines of code. This pattern is shown in Figure 3 and Table 4. The second most frequent test smell is *Conditional Test Logic*, as for every 536 lines of test code in the industrial projects and 284 lines of test code in the open-source projects, the smell prevails. Conversely, *Empty Tests* are rare in both type of projects, occurring only for every 20,561 and 27,186 lines of test code, respectively. Considering the test methods, one *Assertion Roulette* test smell is found for every 32 test methods in the industrial projects and 23 test methods in the open-source projects, depicted in Figure 4. In terms of the number of

Table 5

t-test Results for Normalized Test Code Metrics (Mean) Between Industrial and Open-source Projects

Metric	Industrial Projects (Mean)	Open-source Projects (Mean)	t-test (2-tail) p-value
Total Test Files	0.086297	0.166895	0.0848
Test LOC	0.119178	0.181811	0.2889
Test NOC	0.175402	0.161391	0.8047
Test NOM	0.524443	0.576357	0.8027
Test LOC per File	1.02626	1.020543	0.9772
Test NOC per File	2.070511	1.01271	0.009
Test NOM per File	5.157113	3.488941	0.0592

**Figure 5:** ISFi(TNOC) Score Comparison for each test smell

classes, for every 3 test classes in the industrial projects and 2 test classes in the open-source projects, one Assertion Roulette test smell is seen.

The second most frequent test smell is Conditional Test Logic based on ISFi(TNOM) and ISFi(TNOC) as well. In fact, the smell is so frequent that for every 6 classes and 65 test methods in the industrial projects and 2 classes and 31 test methods in the open-source projects, one CTL test smell is found. In contrast, the third most frequent test smell in industrial projects is the Magic Number Test. Specifically, this smell is found once for every 857 test lines of code, 103 test methods, and 9 test classes. However, with 466 TLOC, 51 TNOM, and 4 TNOC, Duplicate Assertion is the third most occurring test smell in the open-source projects. Additionally, one of every 2,474 test methods and 217 test classes in the industrial projects and 2,998 test methods and 224 test classes in the open-source projects has been identified as an Empty Test. Other smells are not as prevalent, as highlighted in the observations of the study in Table 2.

In order to corroborate these observations, a one-tailed t-test with a 5% significance level is used to conduct a statistical hypothesis test. The p-value is displayed in the final column of Table 4, and the value that rejects the null hypothesis is bolded. The null hypothesis states that there is no difference in the occurrence of a particular test smell between open-source and industrial projects, while the alternative hypothesis refers that the open-source system has higher occurrences. The p-values in Table 4 indicate that the null hypothesis is rejected for 8 out of the 14 test smells, suggesting a statistically significant higher occurrence of these smells in open-source projects. These findings emphasize that open-source projects are more susceptible to frequent and diverse test smells compared to industrial projects, highlighting the need for targeted improvements in test-writing practices within the open-source community.

4.3. RQ3: How does the volume of test code differ between open-source and industrial Python projects?

To explore differences in test code volume between open-source and industrial Python projects, we analyzed key metrics including the number of test files, Test Lines of Code (TLOC), number of test classes (TNOC), and number of test methods (TNOM). These metrics, normalized for project size variations, were assessed to enable a fair comparison between the two environments. Analyzing test code volume provides a foundational understanding that complements the examination of test smells across these projects. Differences in test smell prevalence could be influenced by the ratio of test code to production code, making test code volume an essential aspect to assess. This approach sheds light not only on the presence of test smells but also on potential structural distinctions in test code organization and density across open-source and industrial projects.

The null hypothesis (H_0) posits that there is no significant difference in the volume of test code between open-source and industrial Python projects ($\mu_{\text{Industry}} = \mu_{\text{Open Source}}$). In contrast, the alternative hypothesis (H_1) suggests that the test code volume differs between the two environments ($\mu_{\text{Industry}} \neq \mu_{\text{Open Source}}$). Table 5 provides the t-test results comparing these metrics across open-source and industrial projects. While most metrics, such as Total Test Files, Test LOC, Test NOC, and Test NOM, did not show statistically significant differences, the *Test NOC per File* metric, with a p-value of 0.009, indicates a notable distinction in the number of test classes per file between the two environments. This finding suggests a potential structural difference in test code organization across open-source and industrial projects.

The t-test was conducted for all test code metrics extracted from both industrial and open-source projects. The null hypothesis was accepted for 6 metrics, indicating no statistically significant difference in test code volume or structure between the industrial and open-source projects for most metrics. However, the null hypothesis was rejected for 1 metric, *Test NOC per File* (marked in bold in Table 5), supporting the claim that the number of test classes per file in open-source projects is significantly different from that in industrial projects, suggesting a distinct approach to organizing test classes. These findings are summarized in Table 5, showing that for the majority of metrics (6 out of 7), there is no significant difference in test code volume between open-source and industrial projects. This suggests that, with respect to basic testing characteristics like the number of test files, lines of code, and methods, both environments follow relatively similar practices.

However, the significant difference in the *Test NOC per File* metric highlights that open-source projects may

organize test classes differently, potentially due to variations in project structure, coding standards, or collaborative development practices typical in open-source environments.

5. Threats to Validity

This section addresses construct, external, and reliability threats that may affect the validity of our empirical investigation [37].

5.1. Construct Validity

The accuracy of PyNose, our test smell detection tool, may affect construct validity. Any inaccuracies in detecting test smells could lead to incorrect conclusions regarding their presence and significance. To minimize this risk, we used PyNose, a widely recognized tool in test smell detection with reliability confirmed in prior studies.

5.2. External Validity

Our findings' generalizability is limited by several external validity threats. First, this study focuses on Python-based projects, so results may not fully apply to projects in other languages like Java or C#. Additionally, the dataset comprises 11 industrial projects from Bangladesh and 64 open-source projects, which may not represent the broader global industry practices. The industrial projects come from a specific sector that may have unique testing practices. To address these threats, we included well-known open-source projects with diverse testing practices.

5.3. Reliability Validity

Reliability concerns arise from the need to ensure replicability, particularly for industrial data. Due to confidentiality constraints, we cannot disclose details about the industrial projects, limiting direct replication. However, open-source project findings are fully replicable, as these projects and the PyNose tool are publicly accessible. For further validation, future studies could replicate the analysis with similar industrial datasets.

6. Conclusion

This research investigates the occurrence, distribution, and frequency of test smells in both industrial and open-source projects, highlighting the differences between these two environments. The inclusion of industrial projects alongside open-source ones adds depth to the study, as it reveals distinct patterns in the prevalence of test smells. Although Assertion Roulette (AR) is the most common test smell, appearing in 92% of open-source projects and 81% of industrial projects, a one-tailed t-test shows significant differences in the overall prevalence of test smells between these two types of projects.

Furthermore, 10 out of 18 test smells were observed in industrial projects, while 14 out of 18 test smells were detected in open-source projects. Notably, 8 out of the 14 test smells showed a significantly higher occurrence in open-source projects. Analyzing test writing practices, we found no significant differences in 6 out of 7 metrics between industrial and open-source projects, except for the number of test classes per file (TNOc).

These findings will assist developers in reducing test smells during software development and provide insights for researchers to refine smell refactoring techniques, particularly in terms of frequency and distribution. Future work should explore how specific development practices contribute to these differences between project types, offering a deeper understanding of how coding conventions and workflows influence test smell occurrence. Further research could also investigate automated refactoring techniques that account for these structural and volumetric differences to better support quality improvements in both open-source and industrial projects.

References

- [1] T. Virgínio, R. Santana, L. A. Martins, L. R. Soares, H. Costa, I. Machado, On the influence of test smells on test coverage, in: Proceedings of the XXXIII Brazilian Symposium on Software Engineering, 2019, pp. 467–471.
- [2] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, A. Bacchelli, On the relation of test smells to software code quality, in: 2018 IEEE international conference on software maintenance and evolution (ICSME), IEEE, 2018, pp. 1–12.
- [3] G. Meszaros, xUnit test patterns: Refactoring test code, Pearson Education, 2007.
- [4] S. Balthes, P. Ralph, Sampling in software engineering research: A critical review and guidelines, *Empirical Software Engineering* 27 (2022) 94.
- [5] A. Vahabzadeh, A. M. Fard, A. Mesbah, An empirical study of bugs in test code, in: 2015 IEEE international conference on software maintenance and evolution (ICSME), IEEE, 2015, pp. 101–110.
- [6] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, D. Binkley, An empirical analysis of the distribution of unit test smells and their impact on software maintenance, in: 2012 28th IEEE international conference on software maintenance (ICSM), IEEE, 2012, pp. 56–65.
- [7] V. Garousi, B. Küçük, Smells in software test code: A survey of knowledge in industry and academia, *Journal of systems and software* 138 (2018) 52–81.
- [8] A. Peruma, K. S. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, F. Palomba, On the distribution of test smells in open source android applications: An exploratory study (2019).
- [9] J. W. Paulson, G. Succi, A. Eberlein, An empirical study of open-source and closed-source software products, *IEEE transactions on software engineering* 30 (2004) 246–256.
- [10] A. Van Deursen, L. Moonen, A. Van Den Bergh, G. Kok, Refactoring test code, in: Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001), Citeseer, 2001, pp. 92–95.
- [11] L. Martins, D. Campos, R. Santana, J. M. Junior, H. Costa, I. Machado, Hearing the voice of experts: Unveiling stack exchange communities' knowledge of test smells, in: 2023 IEEE/ACM 16th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE), IEEE, 2023, pp. 80–91.
- [12] A. Qusef, M. O. Elish, D. Binkley, An exploratory study of the relationship between software test smells and fault-proneness, *IEEE Access* 7 (2019) 139526–139536.

- [13] Y. Fushihara, H. Aman, S. Amasaki, T. Yokogawa, M. Kawahara, Fault-proneness of python programs tested by smelled test code (2024).
- [14] D. J. Kim, T.-H. Chen, J. Yang, The secret life of test smells-an empirical study on test smell evolution and maintenance, *Empirical Software Engineering* 26 (2021) 1–47.
- [15] F. Palomba, A. Zaidman, A. De Lucia, Automatic test smell detection using information retrieval techniques, in: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2018, pp. 311–322.
- [16] D. Falessi, N. Juristo, C. Wohlin, B. Turhan, J. Münch, A. Jedlitschka, M. Oivo, Empirical software engineering experts on the use of students and professionals in experiments, *Empirical Software Engineering* 23 (2018) 452–489.
- [17] T. Wang, Y. Golubev, O. Smirnov, J. Li, T. Bryksin, I. Ahmed, Pynose: a test smell detector for python, in: 2021 36th IEEE/ACM international conference on automated software engineering (ASE), IEEE, 2021, pp. 593–605.
- [18] V. Pontillo, D. Amoroso d’Aragona, F. Pecorelli, D. Di Nucci, F. Ferrucci, F. Palomba, Machine learning-based test smell detection, *Empirical Software Engineering* 29 (2024) 55.
- [19] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, F. Palomba, Tsdetect: An open source test smells detection tool, in: Proceedings of the 28th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering, 2020, pp. 1650–1654.
- [20] T. Virginio, L. Martins, L. Rocha, R. Santana, A. Cruz, H. Costa, I. Machado, Jnose: Java test smell detector, in: Proceedings of the XXXIV Brazilian Symposium on Software Engineering, 2020, pp. 564–569.
- [21] A. Bodea, Pytest-smell: a smell detection tool for python unit tests, in: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, 2022, pp. 793–796.
- [22] G. Bavota, A. Qusef, R. Oliveto, A. De Lucia, D. Binkley, Are test smells really harmful? an empirical study, *Empirical Software Engineering* 20 (2015) 1052–1094.
- [23] D. J. Kim, An empirical study on the evolution of test smell, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings, 2020, pp. 149–151.
- [24] B. Hauptmann, M. Junker, S. Eder, L. Heinemann, R. Vaas, P. Braun, Hunting for smells in natural language tests, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 1217–1220.
- [25] E. Soares, M. Aranda, N. Oliveira, M. Ribeiro, R. Gheyi, E. Souza, I. Machado, A. Santos, B. Fonseca, R. Bonifácio, Manual tests do smell! cataloging and identifying natural language test smells, in: 2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE, 2023, pp. 1–11.
- [26] The open catalog of test smells, 2024. URL: <https://test-smell-catalog.readthedocs.io/en/latest/>, accessed: [2024-11-09].
- [27] K. Lucas, R. Gheyi, E. Soares, M. Ribeiro, I. Machado, Evaluating large language models in detecting test smells, arXiv preprint arXiv:2407.19261 (2024).
- [28] F. Palomba, A. Zaidman, The smell of fear: On the relation between test smells and flaky tests, *Empirical Software Engineering* 24 (2019) 2907–2946. URL: <https://link.springer.com/article/10.1007/s10664-018-9662-1>, retracted on 26 March 2020.
- [29] B. Camara, M. Silva, A. Endo, S. Vergilio, On the use of test smells for prediction of flaky tests, in: Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing, 2021, pp. 46–54. URL: <https://doi.org/10.1145/3482909.3482916>. doi:10.1145/3482909.3482916.
- [30] Y. Fushihara, H. Aman, S. Amasaki, T. Yokogawa, M. Kawahara, A trend analysis of test smells in python test code over commit history, in: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2023, pp. 310–314.
- [31] M. M. Rahman, A. Satter, M. M. A. Joarder, K. Sakib, An empirical study on the occurrences of code smells in open source and industrial projects, in: Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2022, pp. 289–294.
- [32] H. Schütze, C. D. Manning, P. Raghavan, Introduction to information retrieval, volume 39, Cambridge University Press Cambridge, 2008.
- [33] W. Aljedaani, A. Peruma, A. Aljohani, M. Alotaibi, M. W. Mkaouer, A. Ouni, C. D. Newman, A. Ghallab, S. Ludi, Test smell detection tools: A systematic mapping study, in: Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, 2021, pp. 170–180.
- [34] A. Eghbali, M. Pradel, Dynapyt: a dynamic analysis framework for python, in: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2022, pp. 760–771.
- [35] R. Sandouka, H. Aljamaan, Python code smells detection using conventional machine learning models, *PeerJ Computer Science* 9 (2023) e1370.
- [36] D. Fernandes, I. Machado, R. Maciel, Tempy: Test smell detector for python, in: Proceedings of the XXXVI Brazilian Symposium on Software Engineering, 2022, pp. 214–219.
- [37] P. Runeson, M. Host, A. Rainer, B. Regnell, Case study research in software engineering: Guidelines and examples, John Wiley & Sons, 2012.