Prioritizing Test Smells: An Empirical Evaluation of Quality Metrics and Developer Perceptions

Md Arif Hasan
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
bsse1112@iit.du.ac.bd

Toukir Ahammed
Institute of Information Technology
University of Dhaka
Dhaka, Bangladesh
toukir@iit.du.ac.bd

***Abstract*—Test smells, suboptimal patterns in test code, impair software maintainability and reliability, especially in resource-constrained open-source Python projects. While detection tools such as *PyNose* identify python-specific test smells, prioritizing them for refactoring remains a challenge due to the lack of test-specific frameworks. This study proposes a metric-driven approach that integrates Change Proneness (CP) and Fault Proneness (FP) metrics, computed via Spearman's rank correlation, to quantify maintenance and reliability risks across 15 test smells in 52 open-source Python projects. Complementing this, a survey of 45 developers captures subjective severity perceptions. By applying Martin Fowler's Technical Debt Quadrant, we classify smells based on empirical risk and developer insights into four categories, enabling better prioritization. Out of the 15 analyzed smells, *Conditional Test Logic*, *Duplicate Assert*, *Obscure In-Line Setup*, and *Redundant Assertion* belong to the highest-priority category for refactoring. These smells are characterized by both high empirical risk and strong developer agreement. This integrated framework advances test smell prioritization by combining data-driven analysis with practitioner perspectives, facilitating efficient refactoring decisions and improved test suite quality.**

***Index Terms*—Test Smells, Prioritization, Change Proneness, Fault Proneness, Software Testing**

I. Introduction

Just like production code, test code can suffer from design issues that make it harder to maintain, understand, and evolve [1]. These issues often appear as test smells, which are symptoms of poorly designed tests that reduce readability and increase development effort [2]. If left unaddressed, they can degrade test quality and make test suites more fragile and costly to manage [1, 2]. The concept was first introduced by van Deursen et al. [1], and has become increasingly relevant as automated testing plays a central role in modern software development.

Despite their importance, test code is often written with less discipline than production code [3], resulting in the accumulation of smells that compromise test suite quality. These issues are particularly pronounced in open-source projects, where limited developer resources and high evolution rates make test maintenance more challenging [4, 5]. The number of known test smells continues to grow. Many of these smells are linked to frequent modifications and defect likelihood [3, 6]. Prioritizing the most critical smells for refactoring is essential, yet remains largely understudied [7].

Recent advances in test smell detection, supported by tools like *tsDetect* for Java [8] and *PyNose* for Python [5], have improved the identification of test quality issues. However, existing prioritization methods such as SpIRIT [9] and developer-driven approaches [10] mainly target production code. These techniques often overlook test-specific concerns and fail to incorporate relevant test quality metrics [7]. This gap is particularly significant in Python ecosystems, where the prevalence and severity of test smells vary widely, highlighting the need for targeted prioritization strategies [5].

This paper presents an empirical evaluation of test smell prioritization in Python projects by integrating quality metrics with developer perceptions. We analyze 15 test smells using two key quality metrics: change proneness and fault proneness. To capture practitioner perspectives, we surveyed 45 experienced Python developers, 70% with over 10 years of experience and 25 to 30% in leadership roles, on test smell severity. Instead of directly correlating developer perceptions with metrics, we use Martin Fowler's *Technical Debt Quadrant* [11]. This helps categorize smells by intent and prudence, combining empirical risk with developer perspectives [12, 13]. This framework supports targeted refactoring by aligning measurable risks with real-world development experience, enhancing test suite maintainability and effectiveness.

To explore the effectiveness of this combined approach, we investigate two main aspects. First, we examine the extent to which change proneness and fault proneness metrics contribute to prioritizing test smells in open-source Python projects. Next, we explore how developer perceptions and empirical metrics can be combined to guide test smell prioritization.

Our data-driven results highlight *Conditional Test Logic* (CTL) and *Assertion Roulette* (AR) as high-priority test smells, showing strong associations with CP and FP (prioritization scores of 0.38 and 0.35, respectively). In contrast, smells such as *Lack of Cohesion of Test Cases* (LCTC) and *Empty Test* (ET) exhibit low or negative scores, suggesting a lower urgency for refactoring. Developer perceptions partially reflect these findings. For example, *Duplicate Assertion* ranks highly in both empirical analysis and developer feedback.

However, important gaps remain. Notably, *Redundant Assertion (RA)* is perceived as low-risk by developers despite its high empirical risk, suggesting hidden technical debt.

To address such discrepancies, we apply Martin Fowler's *Technical Debt Quadrant* to classify test smells into four categories. Among them, *Prudent and Deliberate* represents the highest-priority group for refactoring and includes *Conditional Test Logic*, *Duplicate Assert*, *Obscure In-Line Setup*, and *Redundant Assertion*. *Reckless and Deliberate* captures moderately high-risk smells (e.g., RA), *Prudent and Inadvertent* reflects moderately lower risks (e.g., Eager Test), and *Reckless and Inadvertent* comprises the lowest priority smells. This classification bridges empirical risk and developer perception, enabling context-aware and targeted prioritization of test smell remediation in Python projects. Future work can extend this approach to other languages, incorporate broader datasets, and explore additional metrics to enhance prioritization accuracy.

II. Related Work

Test smells are suboptimal structures in test code that reduce maintainability and effectiveness, analogous to code smells in production systems [1]. Since their introduction by Van Deursen et al. [1], test smells have been primarily investigated in Java [3, 8, 14], with more recent efforts addressing Python, C#, and Scala [5]. The taxonomy has expanded beyond foundational types like *Assertion Roulette* and *General Fixture* to include patterns such as *Conditional Test Logic* [2] and *Suboptimal Assert* [5], which are linked to reduced comprehensibility, fault-proneness, and flaky tests [3, 15].

Detection has evolved from manual inspection to static and dynamic analysis tools such as tsDetect, JNose, and PyNose [5, 8, 16]. While machine learning approaches have improved detection accuracy [17], there remains no established framework for prioritizing test smells, particularly in Python. Empirical evidence on their impact is limited, and existing tools offer little guidance beyond identification [3, 5].

In contrast, prioritization techniques for production code smells are comparatively well-developed. Vidal et al. [9] proposed SpIRIT, which uses change history, developer-assigned importance, and architectural impact to rank smells. Pecorelli et al. [18] introduced a developer-centric prioritization method that combines subjective feedback with code metrics. Fontana et al. [18] also developed a code smell intensity index to quantify severity and support prioritization. Verma et al. [7] conducted a systematic review of prioritization approaches, emphasizing that integrating multiple factors and perspectives improve technical debt management and refactoring guidance.

However, no established prioritization technique specifically targets test smells [7]. Although studies show variability in test smell frequency and impact [16, 19], existing tools lack prioritization support. As a result, developers must manually triage long lists of test smells, which is especially challenging in resource-constrained open-source projects [7]. Test smells increase change- and fault-proneness [3, 6], and frequent changes indicate higher maintenance effort [20], yet these metrics are rarely used for prioritization. Developer perception also affects test smell management, with many underestimating their severity or unaware of long-term impacts, contributing to their persistence [21]. Furthermore, despite growing adoption, research on Python test smells remains limited [4, 5].

To address existing gaps in test smell prioritization, a systematic approach is required that combines both empirical quality metrics and developer perceptions. While prior work on code smell prioritization has explored metric-based [9, 22] and developer-driven [18] strategies, such multidimensional approaches remain underexplored for test smells. Most existing studies focus on detection [5, 8], with limited support for prioritization that reflects both technical risk and practitioner insight. Bridging this gap requires frameworks aligned with perspectives like Martin Fowler's *Technical Debt Quadrant* [11], enabling more informed and actionable decisions.

III. Methodology

This study presents a unified prioritization approach for test smells by integrating change and fault proneness metrics [3, 6, 23] with developer perceptions [21] to inform prioritization through both empirical risk and real-world development experience. We analyze 52 open-source Python repositories, detecting 15 common test smell types using the *PyNose* tool [5]. CP and FP metrics are computed from modifications and bug fixes in production and test files to assess the effects of test smells on code quality separately. To complement this, we surveyed 45 experienced open-source developers from these projects to understand their perceptions of test smell severity and prioritization preferences. Using Martin Fowler's Technical Debt Quadrant, we group test smells by both their empirical risk and perceived severity to identify which should be addressed first. As illustrated in Figure 1, the methodology consists of five phases, which are described below:

A. Dataset Collection

Our analysis is based on a dataset of 52 open-source Python projects from GitHub, selected based on criteria ensuring active development and adequate test code. Each project had at least 50 stars, 1,000 commits, 10 contributors, over two years of development history, recent activity as of May 2025, and Python as the primary language.

The projects range from 4,390 to 2,794,766 lines of code (LOC), with a focus on those with robust test suites for reliable metric computation. Key attributes, including KLOC (thousands of lines of code), NOM (number of methods), and NOC (number of classes), are summarized in Table I. The complete dataset is available in the online appendix[1].

TABLE I: Overview of the Dataset

Code Type	KLOC		NOM		NOC	
	Total	Avg.	Total	Avg.	Total	Avg.
Production Code	15,623	244	516,786	8,075	126,020	1,969
Test Code	2,548	40	276,694	4,323	20,668	323

B. Test Smells Detection

Test smells are detected using static analysis to identify patterns that impair the maintainability and reliability of Python test suites. We focus on `unittest`-based test classes,

[1]https://github.com/Md-Arif-Hasan/Test-Smell-Prioritization

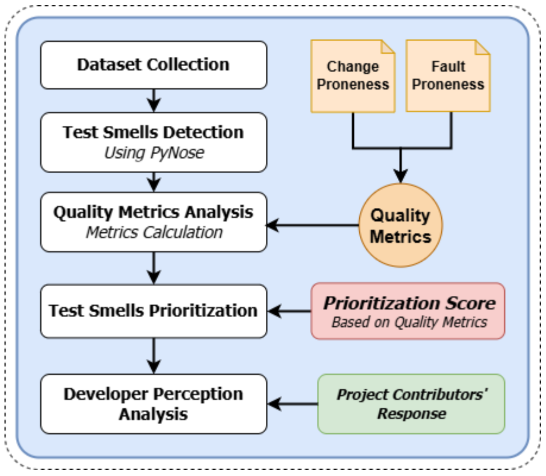

Fig. 1: Overview of Methodology

examining assertions, control flow, and setup practices. For example, *Conditional Test Logic* involves control statements like `if` or `for`, introducing multiple paths that may lead to flakiness and reduce reliability [2, 24]. We used *PyNose* [5], validated for detecting 18 test smells, as shown in prior work [25]. By applying heuristic thresholds on smell occurrences, we identified 15 test smells across the dataset, providing the foundation for computing quality metrics used in prioritization.

C. Quality Metrics Analysis

The Quality Metrics Analysis phase derives Change Proneness (CP) and Fault Proneness (FP) by combining test smell instances detected via *PyNose* with change and fault metrics extracted from Git logs. Each test smell instance is mapped to corresponding production and test files through analysis of co-change patterns and static dependencies, enabling the computation of relevant quality metrics. It aligns with prior research showing strong correlations between test smell occurrences and characteristics of related production and test code [19, 26].

Building on prior work linking test smells to change- and fault-proneness in Java [6, 19], we analyze 15 Python-specific test smells and change proneness using file-level metrics across 52 open-source Python projects. Previous Python research [25] used Bayesian inference on binary fault labels considering only production code. We improve this by applying Spearman's rank correlation on fault metrics from both production and test code for finer, interpretable risk assessment. Details of CP and FP computations follow.

1) Change Proneness (CP): Change Proneness quantifies the maintenance effort associated with a test smell by capturing how frequently and extensively its related code is modified [27, 28]. Following prior work [23, 28], we define CP using two components: change frequency and change extent. *Change frequency* measures the rate at which production and test files associated with a test smell are modified, normalized by total commits:

$$\text{ChgFreq} = \frac{\text{Prod\_Changes}}{\text{Prod\_TotalCommits}} + \frac{\text{Test\_Changes}}{\text{Test\_TotalCommits}} \quad (1)$$

Change extent captures the total code churn (lines added and deleted) in those files, also normalized by commit volume:

$$\text{ChgExt} = \frac{\text{Prod\_CodeChurn}}{\text{Prod\_TotalCommits}} + \frac{\text{Test\_CodeChurn}}{\text{Test\_TotalCommits}} \quad (2)$$

We compute Spearman's rank correlation coefficient (ρ) between smell presence and both change metrics. This non-parametric method is robust to skewed distributions and outliers [29]. The final CP score for a smell *CP(S)* is obtained by summing its correlation values with change frequency and change extent, providing a single interpretable metric for prioritization [3].

2) Fault Proneness (FP): Fault Proneness quantifies the extent to which test smells are linked to fault-related activity in code, serving as an indicator of test reliability issues [6, 25, 30]. Following established methods, FP is computed using two dimensions: fault frequency and fault extent. *Fault frequency* captures how often files linked to a test smell are involved in faulty changes, normalized by total commits:

$$\text{FaultFreq} = \frac{\text{Prod\_FaultyChanges}}{\text{Prod\_TotalCommits}} + \frac{\text{Test\_FaultyChanges}}{\text{Test\_TotalCommits}} \quad (3)$$

Fault extent reflects the volume of code churn caused by fault fixes in the associated files, also normalized:

$$\text{FaultExt} = \frac{\text{Prod\_FaultyChurn}}{\text{Prod\_TotalCommits}} + \frac{\text{Test\_FaultyChurn}}{\text{Test\_TotalCommits}} \quad (4)$$

We identify faulty changes by mining commit messages and issue titles for fault-indicative keywords such as "bug", "fix", or "error" [3]. Associations between test smells and the two fault metrics are then computed using Spearman's rank correlation coefficient (ρ), which is appropriate for skewed and non-normal data distributions [31].

The final FP score for a test smell *FP(S)* is calculated as the sum of its correlations with fault frequency and fault extent. This combined metric offers an interpretable view of a smell's association with FP and helps prioritization for refactoring [3].

D. Data Driven Test Smells Prioritization

To effectively prioritize test smells for refactoring, we compute a unified prioritization score *PS(S)* for each smell *S* by averaging its Change Proneness (CP) and Fault Proneness (FP) values, as shown in Equation 5:

$$PS(S) = \frac{CP(S) + FP(S)}{2} \quad (5)$$

This score captures both the maintenance effort and fault risk associated with each test smell, providing a more comprehensive assessment than analyzing CP and FP separately. Combining these metrics simplifies prioritization, highlights the most critical smells, and supports efficient decision-making for targeted refactoring. We rank all smells in descending order of $PS(S)$, where higher scores indicate greater impact and thus higher refactoring priority, ultimately improving test suite quality and reducing long-term maintenance efforts [10].

E. Developer Perception Analysis

To complement our metric-driven findings and understand developers' perceptions of test smells, we conducted an online survey of contributors to the 52 analyzed Python projects.

Survey Design: The survey is designed to collect participant's demographics, primary development roles, years of general programming and unit testing experience, familiarity with test smells, and views on data-driven prioritization. The participants were then asked to rate the importance of refactoring 15 test smell types on a 5-point Likert scale (1: Not important, 5: Critical to refactor) and report their encounter frequency of 15 test smells. An open-ended question is also provided to collect additional comments. For more details, the full questionnaire is available in the online appendix.

Pilot Testing: The survey was pilot tested with two software engineering researchers and two experienced developers. Feedback led to improvements in question clarity, flow, and estimated completion time (approximately 15 minutes).

Participant Selection: We extracted approximately 1,520 contributor email addresses from the analyzed projects' Git repositories and filtered out invalid or improperly formatted addresses (e.g., missing "@" or domain). Personalized invitations were sent to the remaining 752 valid contacts following software engineering survey best practices [32].

Data Collection and Analysis: The survey was conducted via Google Forms over two weeks, with a mid-term reminder. We received 45 complete responses, yielding a response rate of approximately 6%, which aligns with prior developer surveys [32]. The respondents were mostly experienced Python developers—about 70% had over 10 years of development experience, and 25–30% held leadership roles. Developers rated the importance of refactoring each test smell on a 5-point Likert scale (1 = Not important, 5 = Critical to refactor). The average rating for each smell was used to compute a developer-driven score (DDS), which was then used alongside empirical metrics for prioritization.

We used Martin Fowler's *Technical Debt Quadrant* [11] to classify test smells based on intent (deliberate or accidental) and prudence (careful or careless). Following prior studies [12, 13], we grouped smells into four categories: *Prudent and Deliberate*, *Reckless and Deliberate*, *Prudent and Inadvertent*, and *Reckless and Inadvertent*. This framework enabled the integration of empirical risk and developer perception to guide prioritization decisions.

IV. Results and Discussion

This section presents the prioritization of 15 test smells detected in 52 open-source Python projects by integrating empirical metrics with developer insights from 45 python open-source contributors. Our analysis covers 1,531 production files for Change Proneness (CP) and 1,010 production files for Fault Proneness (FP), each tested by one or more test cases under the `unittest` framework, providing a comprehensive assessment across a large dataset.

Our metric-based ranking identifies Conditional Test Logic (CTL), Assertion Roulette (AR), and Duplicate Assertion (DA) as the highest priority smells, as shown in Table IV. CTL ranks first with a Prioritization Score (PS) of 0.38, driven by its elevated CP (0.46) and FP (0.30), indicating it complicates test evolution and increases fault likelihood [14]. AR ranks second (PS: 0.35), with high CP (0.73) and low FP (-0.03), suggesting frequent modifications that obscure quality issues and complicate debugging. This is consistent with prior studies, which show AR is linked to test flakiness and hampers test maintainability, as it reduces test clarity for other developers [33, 34]. DA, with a PS of 0.24, shows moderate CP (0.46) and FP (0.24), reflecting its maintenance overhead due to redundant assertions.

Conversely, lower-ranked smells such as Lack of Cohesion of Test Cases (LCTC) and Empty Test (ET) present near-zero or negative prioritization scores. These findings align with recent research indicating that such smells may not directly lead to faults but can still hinder maintainability in ways that are not easily captured by standard metrics [35]. Specifically, developers perceive ET as having a significant impact on code maintenance, despite its low prioritization score [34].

Comparing these results with developer perceptions reveals both alignment and divergence. Developers consistently value CTL, which ranks highly in both metrics and practitioner assessments. However, AR is significantly underrated by developers despite its high empirical risk, potentially due to its low visibility or subtlety in daily development practice [36]. Conversely, Sleepy Test (ST) and Empty Test (ET) are ranked as important by developers, even though metrics rate them low [34]. This is because they cause frustration due to test instability, slow execution, and poor readability, not because they directly cause faults [33]. Similarly, Redundant Assertion (RA) ranks much higher in developer perception than metrics, highlighting the importance of practitioner insights to capture subjective pain points and usability issues.

To reconcile empirical and perceptual perspectives, we apply Martin Fowler's *Technical Debt Quadrant* framework [11]. Using PS and developer-driven scores (DDS) normalized by their means, smells are classified into four quadrants based on intent and prudence, as shown in Figure 2.

The *Prudent and Deliberate* quadrant includes smells like CTL, which combine high technical risk with strong developer recognition. CTL ranks first in prioritization score (PS: 0.38) and also shows moderate developer concern (DDS: 2.67), as shown in Table II under columns *PS* and *DDS*. This alignment across both empirical metrics and developer perception places it in the highest-priority group for refactoring.

The *Reckless and Deliberate* quadrant includes smells like AR and MNT. These smells pose high technical risk but are often underestimated by developers [36], leading to long-term maintainability challenges. AR has a high prioritization score (PS: 0.35) due to its strong association with change metrics, yet a low developer-driven score (DDS: 1.80), indicating it is often overlooked. It occurs when a test method contains multiple undocumented assertions, reducing readability and making it difficult to identify the cause of test failures. Similarly, MNT shows moderate empirical risk (PS: 0.23) but

TABLE II: Metric Values, Prioritization Scores and Developers Scores for Test Smells

Data Rank	TS	CF	CE	CP	FF	FE	FP	PS	Dev. Rank	DDS
1	Conditional Test Logic (CTL)	**0.18**	**0.27**	0.46	**0.09**	**0.22**	0.30	0.38	6	2.67
2	Assertion Roulette (AR)	**0.38**	**0.34**	0.73	**-0.11**	**0.08**	-0.03	0.35	15	1.80
3	Duplicate Assert (DA)	**0.25**	0.22	0.46	-0.03	0.06	0.03	0.24	5	2.84
4	Magic Number Test (MNT)	**0.18**	**0.26**	0.44	-0.04	0.06	0.02	0.23	12	2.18
5	Obscure In-Line Setup (OS)	0.12	0.12	0.24	**0.06**	0.11	0.17	0.20	4	2.89
6	Redundant Assertion (RA)	0.09	**0.12**	0.21	0.02	**0.08**	0.10	0.16	2	3.73
7	Exception Handling (EH)	**-0.01**	0.12	0.11	0.03	0.15	0.19	0.15	8	2.47
8	Constructor Initialization (CI)	**0.25**	**0.02**	0.27	**0.00**	0.02	0.02	0.14	10	2.22
9	Suboptimal Assert (SA)	**0.07**	**0.10**	0.17	-0.03	0.04	0.01	0.09	9	2.44
10	Test Maverick (TM)	**-0.22**	**0.01**	-0.22	0.23	**0.16**	0.38	0.08	14	1.82
11	Redundant Print (RP)	**-0.01**	**0.05**	0.04	0.03	**0.06**	0.08	0.06	7	2.51
12	General Fixture (GF)	0.06	-0.03	0.03	0.00	-0.02	-0.02	0.00	11	2.20
13	Sleepy Test (ST)	-0.03	-0.01	-0.04	-0.05	-0.01	-0.06	-0.05	1	4.02
14	Empty Test (ET)	**0.01**	**0.03**	0.03	-0.09	-0.06	-0.15	-0.06	3	3.53
15	Lack of Cohesion of Test Cases (LCTC)	**-0.29**	-0.18	-0.48	**-0.11**	**-0.06**	-0.17	-0.32	13	2.16

CF: Change Frequency, CE: Change Extent, CP: Change Proneness, FF: Fault Frequency, FE: Fault Extent, FP: Fault Proneness
PS: Prioritization Score, DDS: Developer-Driven Score; Bold values indicate statistically significant correlations ($p < 0.05$)

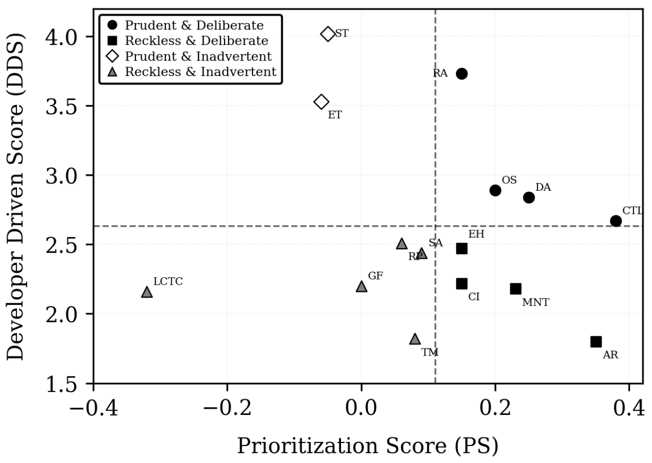

Fig. 2: Technical Debt Quadrant

is also underrated (DDS: 2.18). It involves assert statements with numeric literals (magic numbers), which obscure meaning and complicate maintenance. Refactoring these smells should be prioritized higher-moderately to enhance test clarity, reduce maintenance overhead, and prevent overlooked issues that could degrade the test suite's effectiveness.

The *Prudent and Inadvertent* quadrant includes smells like ST and ET. These have low technical risk but are highly concerning to developers. ST has a low prioritization score (PS: -0.05) but the highest developer-driven score (DDS: 4.02), reflecting strong concern over its tendency to introduce unnecessary delays that vary across devices and slow down test execution. ET similarly shows low empirical risk (PS: -0.06) but a high perceived impact (DDS: 3.53), as it always passes, creating false positives and cluttering the test suite with non-functional code. While neither smell is directly linked to faults, both reduce test efficiency and clarity. Refactoring them should be prioritized lower-moderately to eliminate misleading results, speed up testing, and improve suite maintainability.

Finally, smells in the *Reckless and Inadvertent* quadrant, like LCTC, show low empirical and developer perceived risk, supporting deferral or passive monitoring.

This combined approach supports recent recommendations to use both data and developer experience for managing technical debt effectively [10]. By combining measurable risk with developer insights, it helps prioritize test smells effectively and offers practical guidance for focused refactoring. It can be integrated into CI pipelines to flag high-risk smells, embedded into IDEs for real-time feedback, or extended in static analysis tools. This, in turn, reduces long-term maintenance efforts and fault risks, and improves Python test suite quality.

V. Threats to Validity

Construct Validity: CP and FP metrics may miss contextual or developer-effort factors. The Technical Debt Quadrant adds subjective insights, potentially influenced by survey bias.

External Validity: The study covers 52 open-source Python projects and 15 test smells detected by PyNose; results may not generalize to other languages or frameworks.

Reliability Validity: Reliability may be limited by PyNose detection accuracy and commit data quality. Survey responses are subjective; future studies with varied tools or manual checks can improve robustness.

VI. Conclusion

We propose a test smell prioritization technique that integrates Change Proneness (CP), Fault Proneness (FP), and developer perceptions via the Technical Debt Quadrant. Applied to 15 smells across 52 open-source Python projects, it identifies Conditional Test Logic (CTL) as a top priority, falling into the *Prudent and Deliberate* group due to its high risk and developer consensus.

Assertion Roulette (AR), though empirically high-risk, is underrated by developers and classified as *Reckless and Deliberate*, indicating hidden technical debt. Smells like *Sleepy Test (ST)* and *Empty Test* are perceived as frustrating despite low measured risk, placing them in the *Prudent and Inadvertent* group. This highlights the value of combining perception with metrics. Future work may extend this approach to ecosystems like Java and C#, incorporate factors such as project age and team structure, and assess its effectiveness on larger, real-world datasets.

References

[1] A. Van Deursen, L. Moonen, A. Van Den Bergh, and G. Kok, "Refactoring test code," in *Proceedings of the 2nd international conference on extreme programming and flexible processes in software engineering (XP2001)*, pp. 92–95, Citeseer, 2001.

[2] G. Meszaros, *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[3] D. Spadini, F. Palomba, A. Zaidman, M. Bruntink, and A. Bacchelli, "On the relation of test smells to software code quality," in *2018 IEEE international conference on software maintenance and evolution (ICSME)*, pp. 1–12, IEEE, 2018.

[4] M. A. Hasan and T. Ahammed, "Understanding the prevalence of test smells in open-source and industrial software: An empirical study on python projects," in *12th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2024), Chongqing, China, December 03-04, 2024*, CEUR-WS.org.

[5] T. Wang, Y. Golubev, O. Smirnov, J. Li, T. Bryksin, and I. Ahmed, "Pynose: A test smell detector for python," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE, 2021*, IEEE, 2021.

[6] A. Qusef, M. O. Elish, and D. W. Binkley, "An exploratory study of the relationship between software test smells and fault-proneness," *IEEE Access*, vol. 7, 2019.

[7] R. Verma, K. Kumar, and H. K. Verma, "Code smell prioritization in object-oriented software systems: A systematic literature review," *Journal of Software: Evolution and Process*, 2023.

[8] A. Peruma, K. Almalki, C. D. Newman, M. W. Mkaouer, A. Ouni, and F. Palomba, "tsdetect: an open source test smells detection tool," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference*, 2020.

[9] S. A. Vidal, C. A. Marcos, and J. A. D. Pace, "An approach to prioritize code smells for refactoring," *Autom. Softw. Eng.*, 2016.

[10] F. Pecorelli, F. Palomba, and Khomh, "Developer-driven code smell prioritization," in *Proc. of the 17th International Conf.e on Mining Software Repositories*, 2020.

[11] M. Fowler, "Technical debt quadrant." https://martinfowler.com/bliki/TechnicalDebtQuadrant.html, 2009.

[12] J. Bedi, "Technical debt and commits," *Journal of Software Engineering and Simulation*, 2020.

[13] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, 2015.

[14] A. Panichella, S. Panichella, G. Fraser, A. A. Sawant, and V. J. Hellendoorn, "Test smells 20 years later: detectability, validity, and reliability," *Empir. Softw. Eng.*, 2022.

[15] "Does refactoring of test smells induce fixing flaky tests?," in *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*, IEEE Computer Society, 2017.

[16] T. Virgínio, L. A. Martins, L. R. Soares, and R. Santana, "Jnose: Java test smell detector," in *34th Brazilian Symposium on Software Engineering, SBES 2020, Natal, Brazil, October 19-23, 2020* (E. Cavalcante, F. Dantas, and T. Batista, eds.), ACM, 2020.

[17] V. Pontillo, D. Amoroso d'Aragona, F. Pecorelli, D. Di Nucci, F. Ferrucci, and F. Palomba, "Machine learning-based test smell detection," *Empirical Software Engineering*, 2024.

[18] M. Pecorelli, F. Fontana, and S. Spinelli, "A machine learning based approach for code smells prioritization," in *Proc. of the 17th International Conf. on Mining Software Repositories (MSR)*, 2020.

[19] M. Tufano, F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, A. D. Lucia, and D. Poshyvanyk, "An empirical investigation into the nature of test smells," in *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016* (D. Lo, S. Apel, and S. Khurshid, eds.), ACM, 2016.

[20] S. Wong, Y. Cai, M. Kim, and M. Dalton, "Detecting software modularity violations," in *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011*, ACM, 2011.

[21] V. Garousi and B. Küçük, "Smells in software test code: A survey of knowledge in industry and academia," *J. Syst. Softw.*, 2018.

[22] M. Masudur Rahman, A. Satter, M. Mahbubul Alam Joarder, and K. Sakib, "Software metric based impact analysis of code smells - a large scale empirical study," *Software: Practice and Experience*, vol. 55, no. 5, pp. 925–945, 2025.

[23] M. Siam, M. N. Fuad, and K. Sakib, "An exploratory study on the impact of change-proneness as a metric in black-box test suite minimization," in *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER),2025*, IEEE Computer Society.

[24] F. Palomba and A. Zaidman, "Notice of retraction: Does refactoring of test smells induce fixing flaky tests?," in *2017 IEEE international conference on software maintenance and evolution (ICSME)*, IEEE, 2017.

[25] Y. Fushihara, H. Aman, S. Amasaki, T. Yokogawa, and M. Kawahara, "Fault-proneness of python programs tested by smelled test code," in *50th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2024, Paris, France, August 28-30, 2024*, IEEE, 2024.

[26] L. A. Martins and H. A. X. Costa, "On the diffusion of test smells and their relationship with test code quality of java projects," *J. Softw. Evol. Process.*, 2024.

[27] L. Kumar, S. Lal, A. Goyal, and N. L. B. Murthy, "Change-proneness of oo software using feature selection and ensemble learning," in *Proc. 12th Innovations in Software Eng. Conf. (ISEC)*, 2019.

[28] E. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "A method for assessing class change proneness," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE)*, 2017.

[29] A. S. A. Peruma, *What the smell? an empirical investigation on the distribution and severity of test smells in open source android applications*. Rochester Institute of Technology, 2018.

[30] F. Khomh, M. D. Penta, Y. Guéhéneuc, and G. Antoniol, "An exploratory study of the impact of antipatterns on class change- and fault-proneness," *Empir. Softw. Eng.*, 2012.

[31] A. Agrawal *et al.*, "A practical guide to statistical methods in software engineering," *Empirical Software Engineering Journal*, vol. 24, no. 1, pp. 123–153, 2019.

[32] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *2013 6th International workshop on cooperative and human aspects of software engineering (CHASE)*, pp. 89–92, IEEE.

[33] B. Camara, M. Silva, A. Endo, and 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*, pp. 46–54, 2021.

[34] D. Spadini, M. Schvarcbacher, A.-M. Oprescu, M. Bruntink, and A. Bacchelli, "Investigating severity thresholds for test smells," in *Proceedings of the 17th International Conference on Mining Software Repositories*, pp. 311–321, 2020.

[35] F. Palomba, A. Zaidman, and A. De Lucia, "Automatic test smell detection using information retrieval techniques," in *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 311–322, IEEE, 2018.

[36] G. R. Bai, K. Presler-Marshall, S. R. Fisk, and K. T. Stolee, "Is assertion roulette still a test smell? an experiment from the perspective of testing education," in *2022 IEEE Symposium on visual languages and human-centric computing (VL/HCC)*, IEEE, 2022.