DNose: Dart Test Smell Detector

Tássio Virgínio
Federal University of Bahia
Salvador, Brazil
tassio.virginio@ifto.edu.br

Marcio Ribeiro
Federal University of Alagoas
Maceió, Brazil
marcio@ic.ufal.br

Ivan Machado
Federal University of Bahia
Salvador, Brazil
ivan.machado@ufba.br

ABSTRACT

Test smells, anti-patterns in test code, have emerged as key indicators of test quality, attracting significant attention in software engineering research. Automated detection of test smells is crucial for maintaining effective and readable test suites, yet existing tools often lack comprehensive language support. At the same time, sentiment analysis is gaining prominence as a technique for understanding developers' emotional states during coding activities, providing deeper insights into software quality. Motivated by these research gaps, we present DNose, the first dedicated tool for detecting test smells in Dart-based projects. DNose identifies 14 types of test smells, correlates their introduction with developer sentiment derived from commit messages, and provides integrated insights into test code quality and developer emotions. Our approach contributes to enhancing test maintainability while illuminating emotional aspects of software development practices. Link to the video: https://youtu.be/fCYdDZA8Hkg.

KEYWORDS

Quality of Tests, Test Suite Evolution, Test Smells, Code Coverage

1 Introduction

The increasing dependence of society on software systems has amplified the consequences of software failures, making robust software quality essential [3, 33]. Recent reports estimate the global cost associated with poor-quality software at approximately $2 trillion annually [4]. This highlights the critical need for effective software testing practices capable of identifying and addressing issues early in the development lifecycle [12, 21, 31].

A core aspect of modern software engineering is maintaining test suites that evolve in tandem with production code. High-quality test code facilitates continuous development, minimizes regression risks, and reduces long-term maintenance costs [13, 39]. However, maintaining such quality often demands additional development time and resources, which can introduce inefficiencies if improperly managed.

Previous research has extensively addressed code anti-patterns, widely known as *code smells*, and their impact on software maintainability [8, 38, 40]. Recent studies have extended this investigation to anti-patterns specifically present in test code, known as *test smells* [6, 15, 18, 22, 35]. Test smells degrade test readability and maintainability, raising the risk of faults propagating into production [1, 12].

To address these issues, researchers have developed test smell taxonomies and detection techniques for multiple programming languages, including Java [23, 36], Scala [5], JavaScript [20], and Python [9, 37]. This body of work has led to several automated tools designed to facilitate test smell detection and fixing [9, 20, 34].

Complementing this technical approach, sentiment analysis has emerged as a valuable tool in software engineering, helping to capture developer emotions embedded within textual artifacts such as commit messages [7, 29]. Originally applied in consumer feedback analysis, sentiment analysis is now leveraged to understand how emotions influence software developers' productivity, decision-making, and collaboration dynamics [14]. A commonly employed sentiment resource is the AFINN-165 wordlist [19], widely recognized for effectively scoring emotional polarity in text.

Integrating these insights, this paper introduces **DNose**, an innovative tool specifically developed for detecting test smells in Dart-based software projects and correlating their occurrence with developer sentiment. DNose identifies commits responsible for introducing test smells and analyzes commit messages to gain emotional context surrounding these insertions, providing deeper insights into test quality issues.

The significance of this research is highlighted by the expanding role of mobile software applications and cross-platform frameworks in the digital economy [11, 25, 28]. Frameworks like Flutter (Google), React Native (Meta), and Kotlin Multiplatform (JetBrains) have significantly influenced mobile development, reducing development costs and simplifying maintenance through cross-platform support [32]. Dart, the language behind Flutter, has gained substantial traction, highlighting a clear gap in dedicated tools for test smell detection specific to Dart-based applications.

To address this gap, we present DNose, developed entirely in Dart as a lightweight, web-based application. DNose provides automated detection of 14 test smell types, performs static code and statistical analyses, and integrates sentiment analysis, offering a comprehensive approach to improving test quality in Dart projects.

Contributions. The primary contributions of this paper are:

- DNose, the first Dart-native tool for detecting test smells in Dart-based projects.
- Integration of sentiment analysis with test smell detection to enrich understanding of developer behavior.
- Empirical evaluation of DNose using a real-world Dart project, illustrating its effectiveness and utility.

The remainder of the paper is structured as follows. Section 2 introduces and describes the test smells detected by DNose. Section 3 details DNose's architecture and workflow. Section 4 provides a practical usage example. Section 5 compares DNose to existing detection tools, and Section 6 concludes the paper with future research directions.

2 Test Smells

Test smells were first introduced by Deursen et al. [6] and have since become a well-established means of assessing software test quality [2, 16, 17, 24, 26, 27, 30, 31]. These smells represent poor

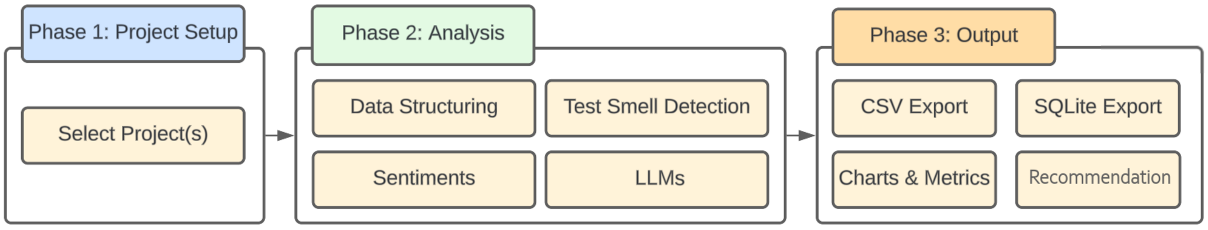


Figure 1: Schematic overview of the DNose process.

practices in test code that may compromise its readability, maintainability, or effectiveness. Over the years, an extensive catalog of test smells has been compiled across different programming languages [10].

Below, we describe the 14 types of test smells currently detected by the DNose tool:

- **Assertion Roulette (AR)**: Occurs when a test contains multiple `expect` statements without descriptive messages. This makes it harder to identify which assertion failed, reducing test clarity and maintainability.
- **Conditional Test Logic (CTL)**: Refers to the presence of conditional statements (e.g., `if`, `switch`) in test code. Such logic can lead to inconsistent behavior and potentially mask defects.
- **Duplicate Assert (DA)**: Happens when the same function or condition is asserted multiple times within a single test, introducing redundancy and clutter.
- **Empty Test (ET)**: Represents a test method that contains no executable code, suggesting incomplete implementation or oversight.
- **Exception Handling (EH)**: Occurs when the correctness of a test depends on exceptions thrown by the production code, which can obscure test intent and lead to brittle tests.
- **Ignored Test (IT)**: Tests that are explicitly disabled or skipped may indicate unfinished work or obsolete test cases, increasing the maintenance burden.
- **Magic Number (MN)**: The use of unexplained numeric literals in test code can obscure meaning and reduce readability. Such values should be replaced with named constants.
- **Print Statement Fixture (PSF)**: The use of print statements in test code is unnecessary and considered a poor practice in automated testing environments.
- **Resource Optimism (RO)**: Assumes that external resources (e.g., files, databases, network connections) are available without verifying their presence, potentially causing tests to fail unexpectedly.
- **Sensitive Equality (SE)**: Involves comparing objects using methods like `toString()`, which may not reliably reflect object equivalence and can lead to fragile tests.
- **Sleepy Fixture (SF)**: Relies on hard-coded delays (e.g., `sleep` calls) to wait for system responses, making tests slower and less reliable across platforms.
- **Test Without Description (TWD)**: A test that lacks a descriptive name or annotation makes it harder to understand its purpose, reducing readability.
- **Unknown Test (UT)**: A test method that does not contain any assertions, thereby failing to verify any expected behavior or outcome.
- **Verbose Test (VT)**: Tests that exceed 30 lines of code may suffer from low readability and high complexity, making them harder to maintain.

3 DNose Tool

This section introduces **DNose**, the tool developed to support this study. DNose enables automated detection of 14 test smell types in Dart-based projects and provides additional insights through sentiment analysis, static code metrics, and developer activity analysis. Below, we describe its architecture, features, and operational workflow.

3.1 Workflow Overview

Figure 1 illustrates the DNose processing pipeline. The workflow begins with project selection, where users submit one or more Dart repositories, optionally via GitHub URLs. Once loaded, and after project selection, the test files are identified and filtered using the "_test.dart" suffix, which is a convention commonly used in the Dart language to designate test files.

- **Test Smell Detection:** Identifies recurring anti-patterns in test code that may hinder maintainability or readability.
- **Sentiment Analysis:** Analyzes commit messages associated with test smells using the AFINN-165 wordlist to infer developer sentiment. To identify the commits responsible for the test smells, we used Git's blame feature.
- **Structural Metadata Extraction:** Collects contextual data from the codebase, such as commit authorship, method location, and other relevant attributes.
- **LLM Recommendations:**[1] Test smell recommendations using three types of LLMs, Gemini, ChatGPT and Ollama.

The outputs are consolidated into a structured dataset, enabling further processing for: (i) Export to CSV files for further analysis; (ii) Visualization of test smell metrics and distributions; and (iii) Local persistence via an embedded SQLite database[2] for future queries.

[1]LLM Recommendations outputs a stream in a separate window, and its data is not merged into the final dataset.
[2]https://www.sqlite.org/

3.2 User Interface and Data Output

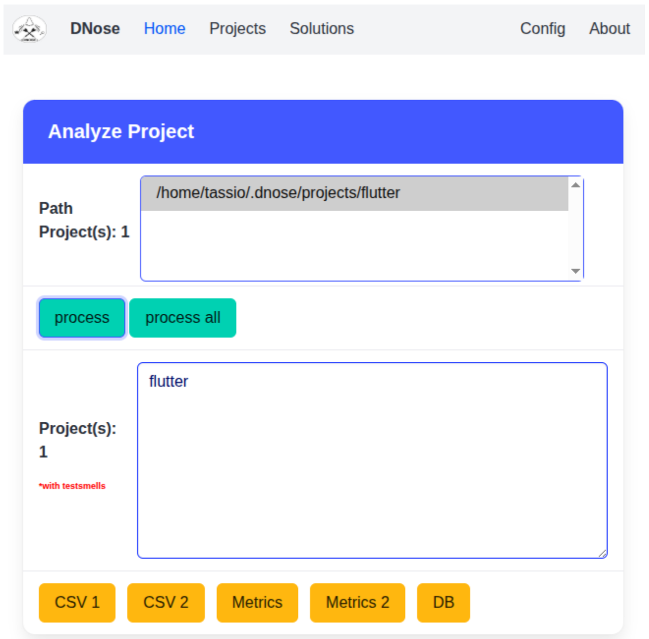


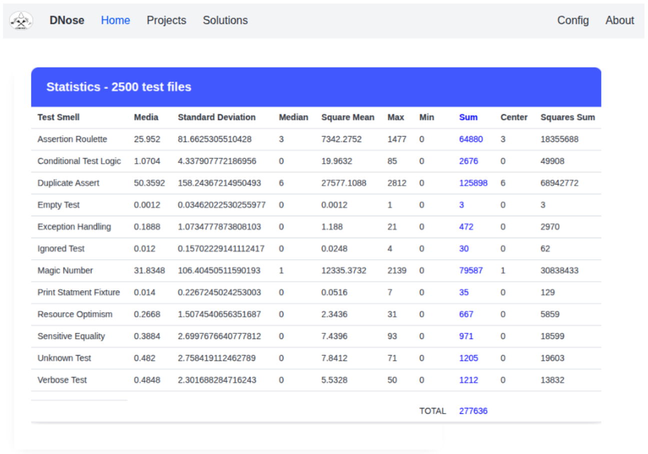
Figure 2: Home screen of the DNose application

Figure 2 shows the DNose home screen. The top navigation menu provides access to modules: *Home*, *Projects*, *Recommendations*, *Settings*, and *About*. Users can process selected repositories or batch-process all listed projects. A lower panel displays previously analyzed projects, alongside downloadable CSV and SQLite files.

The generated artifacts include:

- **CSV 1:** `project_name, test_name, module, path, testsmell, start, end, commit, lineNumber, commitAuthor, author, dateStr, timeStr, summary, score, comparative, words`.
- **CSV 2:** `test_smell, qtd`.
- **Metrics 1:** `project_name, test_name, module, path, metric, start, end, value, commit`.
- **Metrics 2:** Expanded metrics including control-flow and I/O constructs (e.g., `if, for, expect, print, try`) and their test-level variants.
- **Database:** SQLite database with four core tables: `commits, filestests, metrics, testsmells`.

Figure 3 displays statistical summaries computed for each test smell across all selected projects, including metrics such as mean, median, standard deviation, RMS, min/max, and sum.



Statistics - 2500 test files

Test Smell	Media	Standard Deviation	Median	Square Mean	Max	Min	Sum	Center	Squares Sum
Assertion Roulette	25.952	81.6625305510428	3	7342.2752	1477	0	64880	3	18355688
Conditional Test Logic	1.0704	4.337907772186956	0	19.9632	85	0	2676	0	49908
Duplicate Assert	50.3592	158.24367214950493	6	27577.1088	2812	0	125898	6	68942772
Empty Test	0.0012	0.03462022530255977	0	0.0012	1	0	3	0	3
Exception Handling	0.1888	1.0734777873808103	0	1.188	21	0	472	0	2970
Ignored Test	0.012	0.15702229141112417	0	0.0248	4	0	30	0	62
Magic Number	31.8348	106.40450511590193	1	12335.3732	2139	0	79587	1	30838433
Print Statment Fixture	0.014	0.2267245024253003	0	0.0516	7	0	35	0	129
Resource Optimism	0.2668	1.5074540656351687	0	2.3436	31	0	667	0	5859
Sensitive Equality	0.3884	2.6997676640777812	0	7.4396	93	0	971	0	18599
Unknown Test	0.482	2.758419112462789	0	7.8412	71	0	1205	0	19603
Verbose Test	0.4848	2.301688284716243	0	5.5328	50	0	1212	0	13832
						TOTAL	277636		

Figure 3: Screen of the statistical overview of test smell occurrences

3.3 Project Cloning and Repository Management

Figure 4 shows the interface for cloning repositories from GitHub. Users may enter one or multiple URLs, and initiate cloning via a dedicated button. The repositories are downloaded to the `.dnose/projects` directory for further analysis.

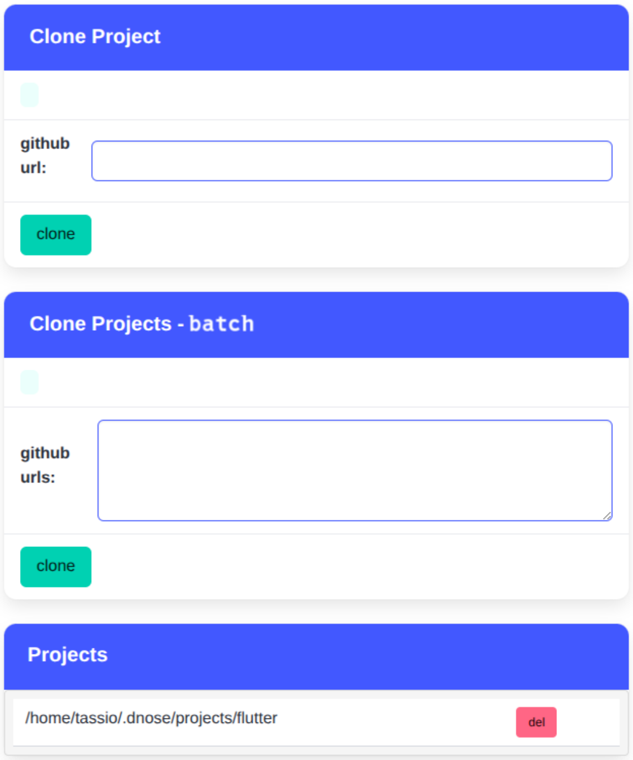
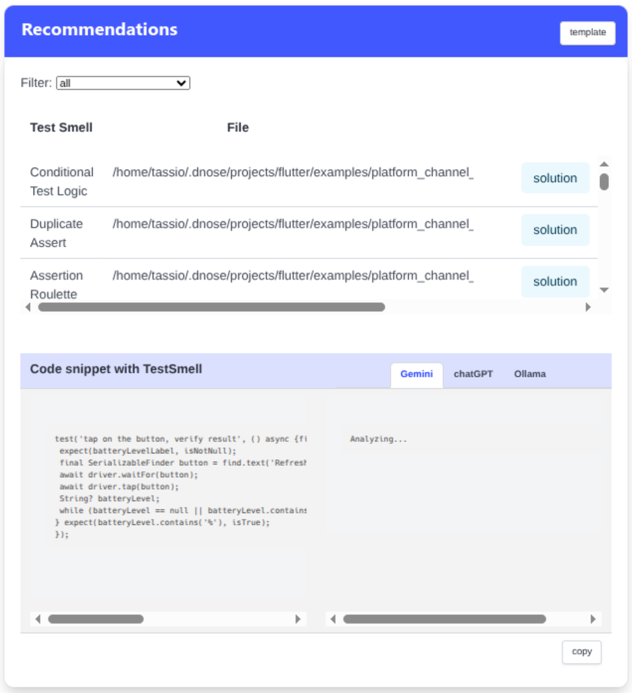


Figure 4: Repository cloning interface

3.4 LLM-based Recommendations for Test Smells

To assist with the remediation of detected test smells, DNose includes a *Recommendations* interface (Figure 5) that integrates with three LLM platforms: Gemini[3], ChatGPT[4], and Ollama[5]—the latter supporting offline inference.

Users can filter test smells by type and generate recommendations by selecting the corresponding code snippets. DNose displays the affected method (left panel) and the generated solution (right panel). The prompts are configurable in the *Settings* interface (Figure 6), and support two placeholders: `{testSmellName}` and `{code_full}`, dynamically replaced at runtime.



Test Smell	File
Conditional Test Logic	/home/tassio/.dnose/projects/flutter/examples/platform_channel_
Duplicate Assert	/home/tassio/.dnose/projects/flutter/examples/platform_channel_
Assertion Roulette	/home/tassio/.dnose/projects/flutter/examples/platform_channel_

```
test('tap on the button, verify result', () async {fi
 expect(batteryLevelLabel, isNotNull);
 final SerializableFinder button = find.text('Refresh
 await driver.waitFor(button);
 await driver.tap(button);
 String? batteryLevel;
 while (batteryLevel == null || batteryLevel.contains
} expect(batteryLevel.contains('%'), isTrue);
});
```

```
Analyzing...
```

Figure 5: Recommendations page: LLM-based recommendations for refactoring

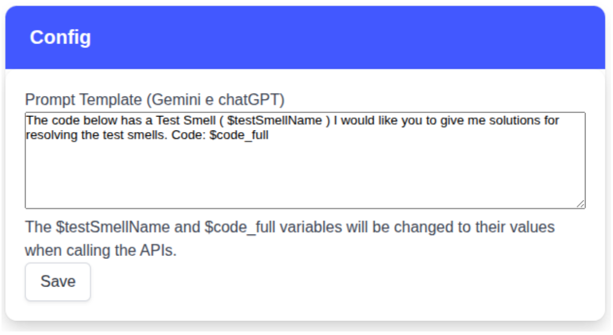
3.5 Configuration and Environment Variables

LLM integration parameters can be specified via environment variables, as shown in Listing 1. These control API access and model selection.

Listing 1: Environment variables for LLM configuration

```
API_KEY_GEMINI=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
API_KEY_CHATGPT=XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
OLLAMA_MODEL=llama3
```

- `API_KEY_GEMINI`: API key for Gemini access.
- `API_KEY_CHATGPT`: API key for ChatGPT access.
- `OLLAMA_MODEL`: Model name used with the Ollama runtime (e.g., `llama3`).



Prompt Template (Gemini e chatGPT)

The code below has a Test Smell ($testSmellName) I would like you to give me solutions for resolving the test smells. Code: $code_full

The $testSmellName and $code_full variables will be changed to their values when calling the APIs.

Figure 6: Settings interface: prompt configuration and API setup

4 Example of Use

The Dart programming language maintains an official package repository, pub.dev[6], which hosts thousands of libraries and projects. For this study, we selected the Flutter project, which is currently the largest and most prominent project written in Dart, and the most widely adopted framework for cross-platform mobile application development [32].

Flutter is an open-source UI toolkit developed by Google, and it boasts a highly active community: the project has over 170,000 stars on GitHub, more than 28,000 forks, and contributions from over 1,500 developers[7]. Beyond a visual framework, Flutter encompasses a comprehensive software stack, including a rendering engine, compilers, build system, native platform integration, and a complete development toolchain.

Its robust architecture and widespread adoption make it a representative and relevant case study. Flutter is used by Google in several of its products and is also employed by prominent companies such as Alibaba, Nubank, eBay, Philips, and The New York Times, among others.

4.1 Results

In this section, we present the results found by the tool for the Flutter project. Initially, DNose provides us with a Table 1 containing statistical data about the detected test smells. For each type of test smell, we have the Mean, Standard Deviation, Median, Square Mean, Minimum, Maximum, and Square Sum.

[3]https://gemini.google.com/
[4]https://chat.com/
[5]https://ollama.com/
[6]https://pub.dev
[7]https://github.com/flutter/flutter

Table 1: Descriptive Statistics for Detected Test Smells in the Flutter Project

Test Smell	Mean	Standard Deviation	Median	Square Mean	Min	Max	Square Sum
Assertion Roulette	25.55	79.71	3	7006.07	0	1502	68145
Conditional Test Logic	1.04	4.24	0	19.09	0	85	2772
Duplicate Assert	48.61	154.19	5	26138.04	0	2814	129644
Empty Test	0.00	0.04	0	0.00	0	1	4
Exception Handling	0.32	2.04	0	4.26	0	64	848
Ignored Test	0.02	0.22	0	0.05	0	8	43
Magic Number	31.00	103.95	1	11766.42	0	2139	82680
Print Statement Fixture	0.02	0.26	0	0.07	0	7	48
Resource Optimism	0.27	1.51	0	2.35	0	31	708
Sensitive Equality	0.44	2.74	0	7.69	0	93	1162
Sleepy Fixture	0.01	0.21	0	0.05	0	10	17
Unknown Test	0.49	2.76	0	7.83	0	71	1297
Verbose Test	0.52	2.35	0	5.80	0	52	1390

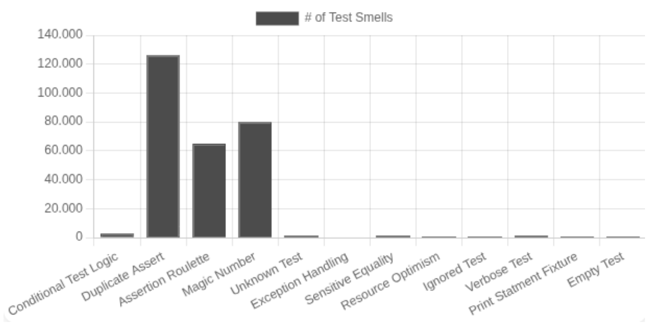


Figure 7: Occurrences of test smells by type

Figure 7 displays the distribution of test smells by type. The three most frequently identified test smells in the Flutter project were: *Duplicate Assert*, *Assertion Roulette*, and *Magic Number*, indicating a pattern of repeated assertions and insufficiently descriptive validations in test code.

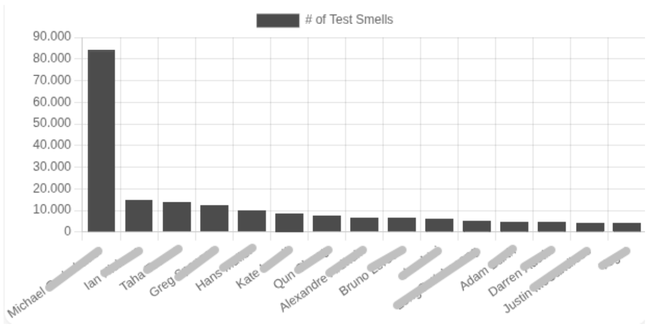


Figure 8: Test smell occurrences by author

Figure 8 presents the number of test smells attributed to each developer. One contributor introduced over 80,000 test smells, suggesting either a high volume of contributions or potentially lower adherence to test quality guidelines.

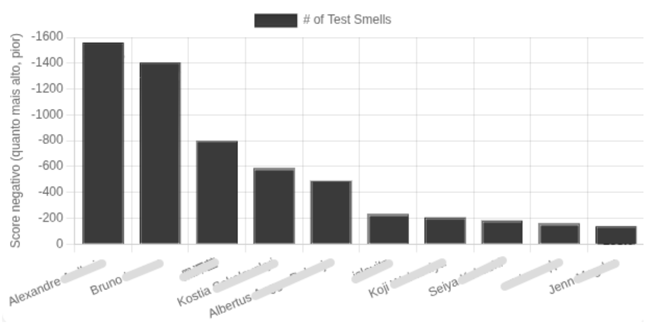


Figure 9: Developers with the highest cumulative negative sentiment

Figure 9 highlights the developers whose commit messages, associated with test smell insertions, reflected the most negative sentiment. This correlation can offer insights into developer frustration, stress, or dissatisfaction during test implementation.

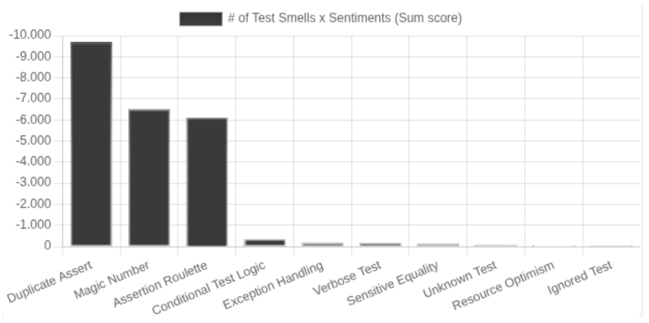
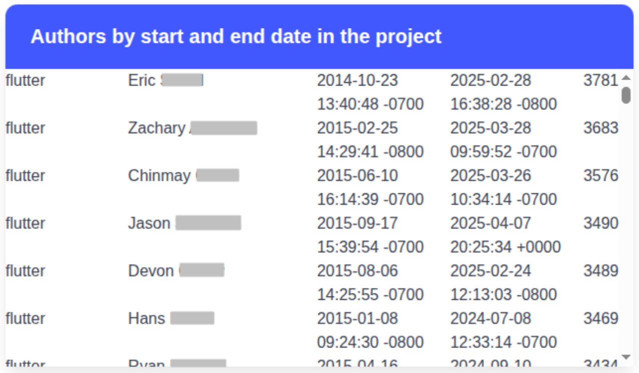


Figure 10: Average sentiment by type of test smell

In Figure 10, we examine the sentiment scores associated with each type of test smell. The smell type with the strongest negative sentiment was *Duplicate Assert*, followed by *Magic Number* and *Assertion Roulette*. These results may reflect developer frustration with redundant validations and ambiguous test logic.

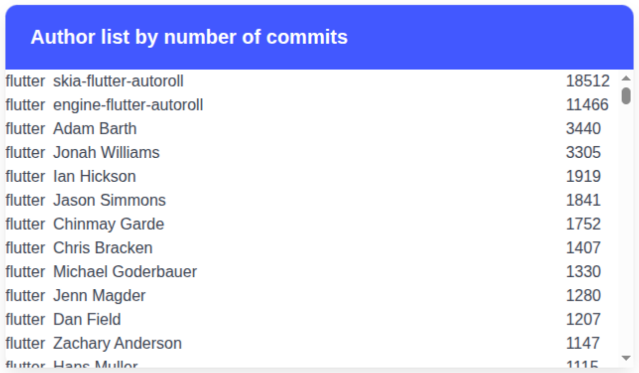


Authors by start and end date in the project

flutter	Eric [redacted]	2014-10-23 13:40:48 -0700	2025-02-28 16:38:28 -0800	3781
flutter	Zachary [redacted]	2015-02-25 14:29:41 -0800	2025-03-28 09:59:52 -0700	3683
flutter	Chinmay [redacted]	2015-06-10 16:14:39 -0700	2025-03-26 10:34:14 -0700	3576
flutter	Jason [redacted]	2015-09-17 15:39:54 -0700	2025-04-07 20:25:34 +0000	3490
flutter	Devon [redacted]	2015-08-06 14:25:55 -0700	2025-02-24 12:13:03 -0800	3489
flutter	Hans [redacted]	2015-01-08 09:24:30 -0800	2024-07-08 12:33:14 -0700	3469
flutter	Ryan [redacted]	2015-04-16	2024-09-10	3434

Table 2: Developer activity period based on first and last commits

Beyond the detection of test smells and sentiment analysis, DNose also provides metadata about developer engagement. Table 2 lists developers along with the start and end dates of their contributions, allowing us to estimate the duration of their activity in the project.



Author list by number of commits

flutter	skia-flutter-autoroll	18512
flutter	engine-flutter-autoroll	11466
flutter	Adam Barth	3440
flutter	Jonah Williams	3305
flutter	Ian Hickson	1919
flutter	Jason Simmons	1841
flutter	Chinmay Garde	1752
flutter	Chris Bracken	1407
flutter	Michael Goderbauer	1330
flutter	Jenn Magder	1280
flutter	Dan Field	1207
flutter	Zachary Anderson	1147
flutter	Hans Muller	1115

Table 3: Number of commits by developer

Table 3 complements the previous table by showing the total number of commits made by each contributor. Together, these tables help contextualize the relationship between developer activity and test quality.

5 Comparison with Similar Tools

Several tools have been proposed for detecting test smells, particularly in the Java programming language. TsDetect [1] is a state-of-the-art tool that identifies twenty-one types of test smells. It operates via the command line and returns only a boolean value indicating the presence or absence of each smell in the test code, offering no graphical interface, which limits its usability.

JNose [34], another tool for Java, was initially developed based on TsDetect. It improves upon its predecessor by detecting twenty-four types of test smells and providing a graphical user interface. JNose also enhances user feedback by displaying the specific lines where smells occur.

In contrast, DNose addresses a previously unserved need by focusing on the Dart language. While it offers a GUI comparable to JNose, DNose goes beyond by integrating features not available in either Java tool. These include statistical summaries, interactive charts, static code metrics (e.g., line count, cyclomatic complexity), sentiment analysis, and insights into developer experience—all within a unified interface.

Beyond Java and Dart, other tools include TEMPY [9] and PyNose [37] for Python, and SNUTS.js [20] for JavaScript. TEMPY and SNUTS.js offer local web-based interfaces, whereas PyNose is integrated as a plugin for the PyCharm IDE. However, these tools are generally limited to basic test smell detection.

In summary, DNose distinguishes itself not only by supporting a different programming language but also by providing a more comprehensive and user-friendly solution, offering advanced features that go beyond detection to support analysis and decision-making in test code maintenance.

6 Conclusion

This paper introduced DNose, a novel tool designed to support the analysis of test code quality by integrating test smell detection, sentiment analysis, static code metrics, and developer activity insights. DNose currently detects 14 types of test smells and performs sentiment analysis using the AFINN-165 wordlist. It also computes key static metrics, such as Lines of Code (LOC) and Cyclomatic Complexity. One of the tool's distinguishing features is its ability to correlate test smells with emotional tone in commit messages and to analyze the relationships between developers and the introduction of test smells.

To demonstrate DNose's capabilities, we applied it to Flutter, the largest project developed in Dart and the most popular cross-platform mobile app framework. We analyzed the distribution of test smells across the project, extracted descriptive statistics (mean, median, and standard deviation), and examined associations between test smells and negative sentiments. Our findings identified which developers were most responsible for introducing test smells and the sentiment trends associated with those insertions. Additionally, we gathered developer-level insights, such as the number of commits and active periods within the project.

As future work, we plan to expand DNose's functionality by: (i) increasing the number of detectable test smell types, (ii) replacing the generic sentiment wordlist with one specifically tailored to the software engineering context, (iii) incorporating suggestions powered by large language models (LLMs) to support automated or semi-automated test smell resolution, (iv) broadening the set of collected code metrics to enhance the tool's analytical capabilities, and (v) conducting a study on the tool's accuracy and precision in detecting test smells.

ARTIFACT AVAILABILITY

The authors declare that the research artifacts supporting the findings of this study are accessible at
https://doi.org/10.5281/zenodo.17059610. The material generated in the example and the video are available on Zenodo:
https://doi.org/10.5281/zenodo.15484918. The DNose site is available at https://dnose-ts.github.io/.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code

001; CNPq grants 140587/2024-1, 315840/2023-4 and 403361/2023-0; and FAPESB grant PIE0002/2022.

REFERENCES

[1] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. Are test smells really harmful? An empirical study. *Empirical Software Engineering* 20, 4 (2015), 1052–1094.

[2] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea De Lucia, and Dave Binkley. 2015. Are test smells really harmful? an empirical study. *Empirical Software Engineering* 20 (2015), 1052–1094.

[3] CISQ. 2019. The Cost of Poor Quality Software in the US 2018 Report. https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/The-Cost-of-Poor-Quality-Software-in-the-US-2018-Report.pdf Last access: May 2025.

[4] CISQ. 2020. The Cost of Poor Quality Software in the US 2018 Report. https://www.it-cisq.org/cisq-files/pdf/CPSQ-2020-report.pdf Last access: May 2025.

[5] Jonas De Bleser, Dario Di Nucci, and Coen De Roover. 2019. SoCRATES: Scala radar for test smells. In *Proceedings of the Tenth ACM SIGPLAN Symposium on Scala* (London, United Kingdom) *(Scala '19)*. Association for Computing Machinery, New York, NY, USA, 22–26. https://doi.org/10.1145/3337932.3338815

[6] Arie Deursen, Leon M.F. Moonen, A. Bergh, and Gerard Kok. 2001. Refactoring Test Code. In *Refactoring Test Code*. CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, The Netherlands.

[7] Rashmi Dhakad and Luigi Benedicenti. 2023. Analyzing Emotional Contagion in Commit Messages of Open-Source Software Repositories. (05 2023), 113–133.

[8] José Pereira dos Reis, Fernando Brito e Abreu, Glauco de Figueiredo Carneiro, and Craig Anslow. 2022. Code Smells Detection and Visualization: A Systematic Literature Review. *Archives of Computational Methods in Engineering* 29, 1 (2022), 47–94. https://doi.org/10.1007/s11831-021-09566-x

[9] Daniel Fernandes, Ivan Machado, and Rita Maciel. 2022. TEMPY: Test Smell Detector for Python. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering* (Virtual Event, Brazil) *(SBES '22)*. Association for Computing Machinery, New York, NY, USA, 214–219. https://doi.org/10.1145/3555228.3555280

[10] Vahid Garousi and Barış Küçük. 2018. Smells in software test code: A survey of knowledge in industry and academia. *Journal of Systems and Software* 138 (2018), 52 – 81.

[11] O Globo. 2024. Raio-X da telefonia no Brasil. https://infograficos.oglobo.globo.com/economia/raio-x-da-telefonia-no-brasil.html. Accessed: 2024-07-15.

[12] Giovanni Grano, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Harald C Gall. 2019. Scented since the beginning: On the diffuseness of test smells in automatically generated test code. *Journal of Systems and Software* 156 (2019), 312–327.

[13] Dayne Guerra Calle, Julien Delplanque, and Stéphane Ducasse. 2019. Exposing Test Analysis Results with DrTests. In *International Workshop on Smalltalk Technologies*. HAL, Cologne, Germany, 1–5. https://hal.inria.fr/hal-02404040

[14] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment analysis of commit comments in GitHub: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories* (Hyderabad, India) *(MSR 2014)*. Association for Computing Machinery, New York, NY, USA, 352–355. https://doi.org/10.1145/2597073.2597118

[15] Nildo Silva Junior, Luana Almeida Martins, Larissa Rocha, Heitor A. X. Costa, and Ivan Machado. 2021. How are test smells treated in the wild? A tale of two empirical studies. *J. Softw. Eng. Res. Dev.* 9 (2021), 9:1–9:16. https://doi.org/10.5753/JSERD.2021.1802

[16] Luana Almeida Martins, Heitor A. X. Costa, and Ivan Machado. 2024. On the diffusion of test smells and their relationship with test code quality of Java projects. *J. Softw. Evol. Process.* 36, 4 (2024). https://doi.org/10.1002/smr.2532

[17] Luana Almeida Martins, Valeria Pontillo, Heitor A. X. Costa, Filomena Ferrucci, Fabio Palomba, and Ivan Machado. 2025. Test code refactoring unveiled: where and how does it affect test code quality and effectiveness? *Empir. Softw. Eng.* 30, 1 (2025), 27.

[18] Gerard Meszaros, Shaun M. Smith, and Jennitta Andrea. 2003. The Test Automation Manifesto. In *Extreme Programming and Agile Methods - XP/Agile Universe 2003*, Frank Maurer and Don Wells (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg.

[19] F. Å. Nielsen. 2011. AFINN. http://www2.compute.dtu.dk/pubdb/pubs/6010-full.html

[20] Jhonatan Oliveira, Luigi Mateus, Tássio Virgínio, and Larissa Rocha. 2024. SNUTS.js: Sniffing Nasty Unit Test Smells in Javascript. In *Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software* (Curitiba/PR). SBC, Porto Alegre, RS, Brasil, 720–726. https://doi.org/10.5753/sbes.2024.3563

[21] Fabio Palomba, Andy Zaidman, and Andrea De Lucia. 2018. Automatic Test Smell Detection Using Information Retrieval Techniques. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Madrid, Spain, 311–322.

[22] Annibale Panichella, Sebastiano Panichella, Gordon Fraser, Anand Ashok Sawant, and Vincent J. Hellendoorn. 2022. Test smells 20 years later: detectability, validity, and reliability. *Empirical Software Engineering* 27, 7 (2022), 170. https://doi.org/10.1007/s10664-022-10207-5

[23] Anthony Peruma, Khalid Almalki, Christian D. Newman, Mohamed Wiem Mkaouer, Ali Ouni, and Fabio Palomba. 2020. tsDetect: an open source test smells detection tool. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 1650–1654. https://doi.org/10.1145/3368089.3417921

[24] Valeria Pontillo, Luana Almeida Martins, Ivan do Carmo Machado, Fabio Palomba, and Filomena Ferrucci. 2025. An empirical investigation into the capabilities of anomaly detection approaches for test smell detection. *J. Syst. Softw.* 222 (2025), 112320. https://doi.org/10.1016/J.JSS.2024.112320

[25] Davide Quaglione, Nicola Matteucci, Donatella Furia, Alessandro Marra, and Cesare Pozzi. 2020. Are mobile and fixed broadband substitutes or complements? New empirical evidence from Italy and implications for the digital divide policies. *Socio-Economic Planning Sciences* 71 (2020), 100823. https://doi.org/10.1016/j.seps.2020.100823

[26] Railana Santana, Luana Martins, Larissa Rocha, Tássio Virgínio, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. RAIDE: a tool for Assertion Roulette and Duplicate Assert identification and refactoring. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 374–379. https://doi.org/10.1145/3422392.3422510

[27] Railana Santana, Luana Martins, Tássio Virgínio, Larissa Soares, Heitor Costa, and Ivan Machado. 2022. Refactoring Assertion Roulette and Duplicate Assert test smells: a controlled experiment. In *Anais do XXV Congresso Ibero-Americano em Engenharia de Software* (Córdoba). SBC, Porto Alegre, RS, Brasil, 263–277. https://doi.org/10.5753/cibse.2022.20977

[28] Shahriar Shirvani Moghaddam. 2024. The Past, Present, and Future of the Internet: A Statistical, Technical, and Functional Comparison of Wired/Wireless Fixed/Mobile Internet. *Electronics* 13, 10 (2024). https://doi.org/10.3390/electronics13101986

[29] Vinayak Sinha, Alina Lazar, and Bonita Sharif. 2016. Analyzing developer sentiment in commit logs. In *Proceedings of the 13th International Conference on Mining Software Repositories* (Austin, Texas) *(MSR '16)*. Association for Computing Machinery, New York, NY, USA, 520–523. https://doi.org/10.1145/2901739.2903501

[30] Elvys Soares, Manoel Aranda III, Davi Romão, and Márcio Ribeiro. 2023. The Open Catalog of Test Smells. Available at https://test-smell-catalog.readthedocs.io/en/latest/index.html.

[31] Davide Spadini, Fabio Palomba, Andy Zaidman, Magiel Bruntink, and Alberto Bacchelli. 2018. On the Relation of Test Smells to Software Code Quality. In *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, Madrid, Spain, 1–12.

[32] Statista Research Department. 2024. Most used frameworks among developers worldwide as of 2024. https://www.statista.com/statistics/793840/worldwide-developer-survey-most-used-frameworks/. Accessed: 2025-05-06.

[33] Tricentis. 2018. Software Fail Watch: 5th Edition. https://www.tricentis.com/resources/software-fail-watch-5th-edition/ Last access: May 2025.

[34] Tássio Virgínio, Luana Martins, Larissa Rocha, Railana Santana, Adriana Cruz, Heitor Costa, and Ivan Machado. 2020. JNose: Java Test Smell Detector. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (Natal, Brazil) *(SBES '20)*. Association for Computing Machinery, New York, NY, USA, 564–569. https://doi.org/10.1145/3422392.3422499

[35] Tássio Virgínio, Luana Almeida Martins, Railana Santana, Adriana Cruz, Larissa Rocha, Heitor A. X. Costa, and Ivan Machado. 2021. On the test smells detection: an empirical study on the JNose Test accuracy. *J. Softw. Eng. Res. Dev.* 9 (2021), 8:1–8:14. https://doi.org/10.5753/JSERD.2021.1893

[36] Tássio Virgínio, Luana Martins, Larissa Rocha Soares, Santana Railana, Heitor Costa, and Ivan Machado. 2020. An empirical study of automatically-generated tests from the perspective of test smells. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering* (Virtual Conference) *(SBES 2020)*. ACM, New York, NY, USA, 5 pages.

[37] Tongjie Wang, Yaroslav Golubev, Oleg Smirnov, Jiawei Li, Timofey Bryksin, and Iftekhar Ahmed. 2021. PyNose: A Test Smell Detector For Python. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 593–605. https://doi.org/10.1109/ASE51524.2021.9678615

[38] Pravin Singh Yadav, Rajwant Singh Rao, Alok Mishra, and Manjari Gupta. 2024. Machine Learning-Based Methods for Code Smell Detection: A Survey. *Applied Sciences* 14, 14 (2024). https://doi.org/10.3390/app14146149

[39] Vahid Garousi Yusifoğlu, Yasaman Amannejad, and Aysu Betin Can. 2015. Software test-code engineering: A systematic mapping. *Information and Software Technology* 58 (2015), 123 – 147.

[40] Fengji Zhang, Zexian Zhang, Jacky Wai Keung, Xiangru Tang, Zhen Yang, Xiao Yu, and Wenhua Hu. 2024. Data preparation for Deep Learning based Code Smell Detection: A systematic literature review. *Journal of Systems and Software* 216 (2024), 112131. https://doi.org/10.1016/j.jss.2024.112131