Test Smell Detection and Refactoring with Agentic LLMs: An Empirical Study of Small Open Models

Rian Melo
Federal University of Campina Grande (UFCG)
Campina Grande, Brazil
Instituto Kunumi
Belo Horizonte, MG, Brazil
rian.melo@ccc.ufcg.edu.br

Rohit Gheyi
Federal University of Campina Grande (UFCG)
Campina Grande, Brazil
rohit@dsc.ufcg.edu.br

Abstract

Test smells are recurring design problems in test code that can hurt readability and complicate maintenance, and although many approaches detect them, few automate their removal. This work investigates small, open language models—Llama-3.2-3B, Gemma-2-9B, DeepSeek-R1-14B, and Phi-4-14B—for the automated detection and refactoring of test smells through agentic workflows with one, two, and four agents. We evaluate 150 instances of five common test smells drawn from 11 real-world Java projects using JUnit 5, with all models executed locally on consumer-grade hardware via Ollama and LangChain. The models detected nearly all instances (up to 100%), and Phi-4-14B reached the highest refactoring accuracy (75.3% pass@5), rivaling proprietary single-agent baselines such as o3 and Gemini-2.5-Pro while running locally. Multi-agent workflows improved results for three of the five smells. A survey with 50 developers preferred the refactored tests in about 66% of the comparisons, and 6 of 10 submitted pull requests were merged into open-source projects. The approach is easily extended to new smells through natural-language definitions and generalizes to other programming languages, such as Python, Go, and JavaScript.

Keywords

Test Smells, Refactoring, Large Language Models, Agentic Workflows, Software Testing

1 Introduction

Test smells are recurring design problems in test code that can hurt readability and make maintenance more error-prone over time, without necessarily causing immediate functional failures [10]. They are widely studied and prevalent in real-world projects [6]. Common examples include undocumented assertions (*Assertion Roulette*), control flow inside tests (*Conditional Test Logic*), repeated assertions (*Duplicate Assert*), manual exception handling (*Exception Handling*), and hard-coded literals (*Magic Number*). Existing tools mostly rely on static analysis or machine learning [7, 8] and are often hard to adapt to new smells or other programming languages [1]. Few automate the actual removal of a smell, which still tends to require error-prone manual edits.

This paper summarizes a study published in *IEEE Software* [5] and carried out within a larger research collaboration. The work reported here is the undergraduate research of its first author, centered on the agentic pipeline and the evaluation of the open models. We investigate whether small, open language models can detect and refactor test smells when organized into agentic workflows with one, two, and four agents, over 150 instances of five common smells from 11 real-world Java projects [2]. We also benchmark them against proprietary models, assess practical relevance through pull requests and a developer survey, and examine generalization to Python, Go, and JavaScript.

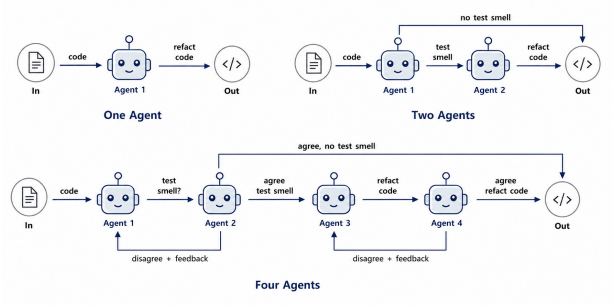

Figure 1: Agentic workflows with one, two, and four agents to detect and remove test smells.

2 Method

We evaluated four small, open models—Llama-3.2-3B, Gemma-2-9B, DeepSeek-R1-14B, and Phi-4-14B—using their default configurations, served locally through Ollama on a MacBook Pro M3 (18 GB RAM) and orchestrated with LangChain. The dataset comprises 150 JUnit 5 test methods (30 per smell, each ≤30 LOC, the range covering 89% of these projects' tests) from 11 real-world open-source Java projects [2], covering five of the most prevalent test smells [6]—Assertion Roulette, Conditional Test Logic, Duplicate Assert, Exception Handling, and Magic Number. Two independent researchers manually validated each instance and each refactoring, checking smell removal and behavior preservation.

We compared single-, two-, and four-agent workflows (Figure 1). The single agent detects and refactors in one prompt. The two-agent setup separates a Detector from a Refactorer, and the four-agent setup adds a Critic that confirms the detection and a Validator that checks the result through an evaluator–optimizer loop (up to three iterations). Each agent uses a role-specific persona, a chain-of-thought prompt [12] with a natural-language smell definition, and a strict output format, so new smells require only a new definition. We report pass@1 and, for the best configuration, pass@5 [3]—a task is solved when at least one of five attempts is correct—and score detection and refactoring separately, so refactoring quality is not masked by detection errors.

3 Results

Detection was high across models at pass@1, with Llama-3.2-3B reaching 100%, Gemma-2-9B 98%, and Phi-4-14B 96%, while DeepSeek-R1-14B trailed at 78%. Adding agents barely affected detection. Refactoring was harder. Phi-4-14B led at 75.3% pass@5, with structural smells such as Magic Number and Duplicate Assert the easiest to refactor and Conditional Test Logic and Exception Handling the hardest. Multi-agent workflows helped on three of the five smells, but a single agent was better for Assertion Roulette. At pass@1, the best single-attempt refactoring per model was 55.3% for Phi-4-14B (two agents), 40.7% for Gemma-2-9B and 22% for Llama-3.2-3B (four agents), and 39.3% for DeepSeek-R1-14B (single agent). Combining their complementary strengths refactored 70.6% of all instances, recovering 23 instances that Phi-4-14B alone missed (Llama-3.2-3B 6, Gemma-2-9B 14, DeepSeek-R1-14B 6). Running the full set under the four-agent setup took about 1h for Llama-3.2-3B and Gemma-2-9B, 5h for Phi-4-14B, and 8h for DeepSeek-R1-14B.

With pass@5, Phi-4-14B (75.3%) rivaled the proprietary single-agent baselines reported in the same study—o3 (74.7%), Gemini-2.5-Pro (72.0%), and Claude-4-Sonnet (59.3%) at pass@1—while running locally at a fraction of the cost. The approach also generalized beyond Java, correctly refactoring (pass@1) 29 of 30 instances in Python, 25 of 30 in JavaScript, and 24 of 30 in Go using a single agent, and detected 28 of 30 additional test smell types from an external catalog. On longer tests (>30 LOC), accuracy dropped to 86% detection and 28% refactoring (pass@1), a current limitation. Beyond test smells, the same pipeline detected all 30 Java code smells evaluated and refactored a subset of them.

A survey with 50 developers (median 2 years of experience) compared original tests against Phi-4-14B refactorings. The refactored version was preferred for Duplicate Assert (78%), Magic Number (76%), and Exception Handling (64%), roughly split for Assertion Roulette, while 62% preferred the original for Conditional Test Logic. Overall, refactored tests were preferred in about 66% of comparisons. Of 10 pull requests with Phi-4-14B refactorings submitted to open-source projects, 6 were merged by human maintainers.

4 Related Work

Most prior work targets test smell *detection* rather than removal. Surveys and mapping studies catalog detection tools and document how prevalent test smells are across real projects [1]. Static detectors such as tsDetect flag a wide range of smells but rely on fixed rules tied to specific languages [7], whereas learning-based methods aim to improve robustness across heterogeneous projects [8]. Support for actually *removing* a smell is comparatively scarce and usually tied to specific smells or frameworks, as in RAIDE [9], while JUnit 5 catalogs offer behavior-preserving transformations that developers still apply by hand [2]. More broadly, surveys map both the opportunities and the risks of applying LLMs to software-engineering tasks, including their sensitivity to prompt design and context [4, 11]. Our work differs by combining detection and end-to-end refactoring in a single collaborative agentic pipeline built on small, open models and benchmarked against proprietary baselines.

5 Conclusion

Small, open language models in agentic workflows can detect and refactor common test smells on consumer-grade hardware. Phi-4-14B reached 75.3% refactoring accuracy (pass@5), rivaling proprietary models such as o3, Gemini-2.5-Pro, and Claude-4-Sonnet that run only on remote infrastructure, and multi-agent workflows helped on three of the five smells. The practical implications are direct—teams can refactor test smells locally, avoiding the cost and data-sharing concerns of proprietary APIs, and extend the pipeline to new smells or languages through natural-language definitions alone, as the cross-language results and the six merged pull requests suggest. The main contributions are a natural-language-extensible agentic pipeline, an empirical comparison of four open models against proprietary baselines, and evidence of practical relevance through merged pull requests, a developer survey, and cross-language generalization. Future work includes covering additional smells and automating behavior-preservation checks.

Artifact Availability

A replication package (dataset, prompts, agent code, and outputs) is available on Zenodo at https://zenodo.org/records/17285750.

Previous Work

This work summarizes a study from a special issue of *IEEE Software*, "Agentic LMs: Hunting Down Test Smells" (vol. 43, no. 1, pp. 32–40, 2026), DOI 10.1109/MS.2025.3621356. The article has 10 citations, and its replication package has 495 views and 221 downloads. The approach was extended with Gemini-2.5-Pro and presented at the ICSE 2026 ACM Student Research Competition (SRC), winning the Gold Award (1st place, undergraduate).

References

[1] Wajdi Aljedaani et al. 2021. Test Smell Detection Tools: A Systematic Mapping Study. In *International Conference on Evaluation and Assessment in Software Engineering*. 170–180. doi:10.1145/3463274.3463335

[2] Elvys Soares et al. 2023. Refactoring Test Smells With JUnit 5: Why Should Developers Keep Up-to-Date? *IEEE Transactions on Software Engineering* 49, 3 (2023), 1152–1170.

[3] Mark Chen et al. 2021. Evaluating Large Language Models Trained on Code. https://arxiv.org/abs/2107.03374

[4] Xinyi Hou et al. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *ACM Transactions on Software Engineering and Methodology* 33, 8 (2024).

[5] Rian Melo, Pedro Simões, Rohit Gheyi, Marcelo d'Amorim, Márcio Ribeiro, Gustavo Soares, Eduardo Almeida, and Elvys Soares. 2026. Agentic LMs: Hunting Down Test Smells. *IEEE Software* 43, 1 (2026), 32–40. doi:10.1109/MS.2025.3621356

[6] Anthony Peruma et al. 2019. On the Distribution of Test Smells in Open Source Android Applications: An Exploratory Study. In *International Conference on Computer Science and Software Engineering*. 193–202.

[7] Anthony Peruma et al. 2020. tsDetect: an open source test smells detection tool. In *Foundations of Software Engineering*. ACM, 1650–1654.

[8] Valeria Pontillo et al. 2024. Machine learning-based test smell detection. *Empirical Software Engineering* 29, 2 (2024), 1–44.

[9] Railana Santana et al. 2020. RAIDE: A Tool for Assertion Roulette and Duplicate Assert Identification and Refactoring. In *34th Brazilian Symposium on Software Engineering (SBES)*. 374–378. doi:10.1145/3422392.3422510

[10] Arie van Deursen et al. 2001. Refactoring test code. In *International conference on extreme programming and flexible processes in software engineering*. 92–95.

[11] Junjie Wang et al. 2024. Software testing with large language models: Survey, landscape, and vision. *IEEE Transactions on Software Engineering* 50 (2024), 911–936.

[12] Jason Wei et al. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*.