Improving JavaScript Test Quality with Large Language Models:

Lessons from Test Smell Refactoring

Master's student
Gabriel Amaral
PGCC - UEFS

Advisor
Larissa Rocha
UNEB/PGCC-UEFS

Co-Advisor
Eduardo Figueiredo
UFMG

Tests Smells

What are they?

Inadequate design patterns that arise in automated test suites.

Impact

Efficacy and maintainability of tests

Indicators

- Slow Testing
- Excessive Configurability
- Interdependent Tests
- Redundant Tests

Example - Lazy Test

```
test("User operations", ()=> {
  createUser("Ana");
  updateUser("Ana", { age: 30 });
  deleteUser("Ana");
  expect(getUser("Ana")).toBeNull();
});
```

Motivation

Several studies analyze the impact of **test smells** on the development process.
(Van Deursen, 2001; Bavota, 2016)

Most studies focus on **statically** typed languages (Java, Scala, C++), leaving a **gap** in **dynamic** languages like **Javascript**, which has few recent studies on the subject.
(Jorge, 2021; Oliveira, 2024)

LLMs have demonstrated great potential in code generation/refactoring.
(Yu, 2023; Hou, 2024)

Research Objectives and Questions

Objective: To evaluate the **effectiveness** of **LLMs** in **refactoring test smells** in JavaScript.

- **RQ1**: Do **LLMs** remove **test smells** without affecting tes**t behavior** or **coverage?**
- **RQ2**: Does refactoring using **LLMs introduce new test smells** or **degrade test quality**?
- **RQ3**: How do **LLMs** differ in removing specific types of **test smells** (**strengths** and **limitations**)?
- **RQ4**: What is the **impact of refactoring** using **LLMs** on structural quality (**complexity**, **size**, **maintainability**)?

Research Design

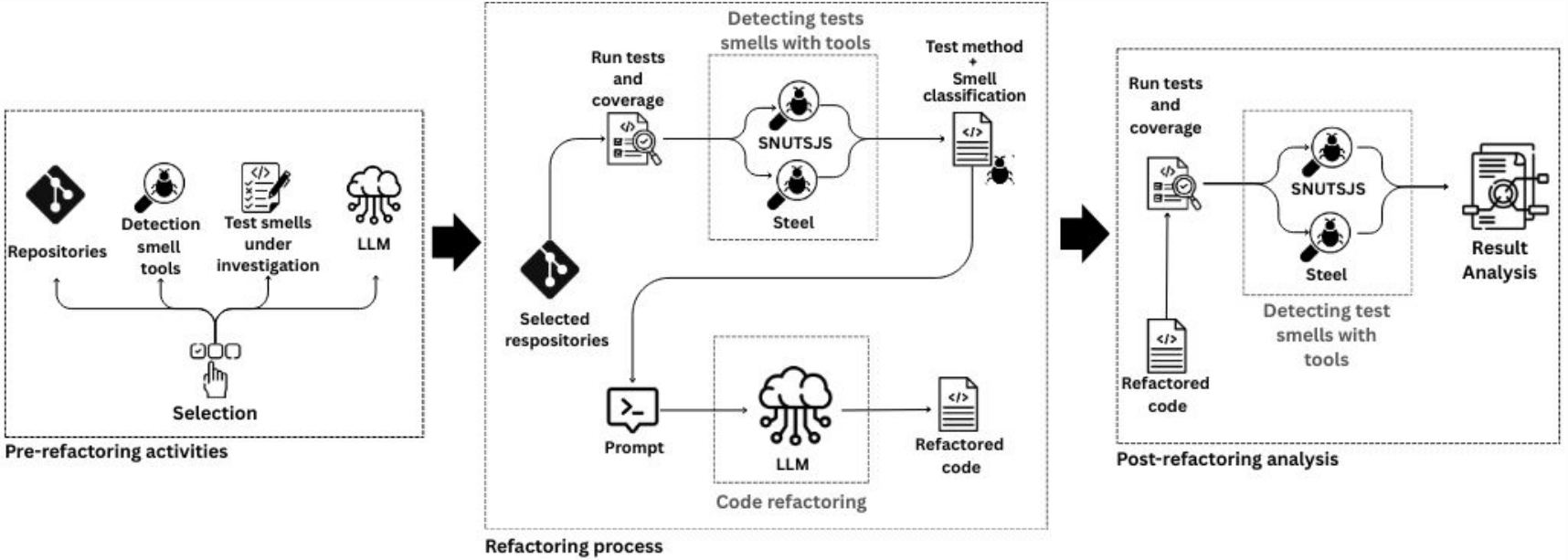


Figure 1: Research **design** flow

Pre-Refactoring

Detection Tools

Steel

(Jorge et al., 2021)

16 types of test smells

SNUTS.JS

(Oliveira et al., 2024)

16 types of test smells

Pre-Refactoring

Selection of Smell Tests

Steel

(Jorge et al., 2021)

Smell	Description
Assertion Roulette	Multiple **assertions** without clear explanation, making it difficult to identify the fault.
Duplicate Assert	Repeated verification of the same condition within the same method, impairing readability.
Magic Number	Use of literal numbers without explanation, reducing readability.
Lazy Test	Superficial testing that checks multiple functionalities without verifying each functionality separately and specifically.
Redundant Print	Unnecessary debug **printouts**, such as **console.log()** etc...

Table 1. Selected smells from the Steel tool.

Pre-Refactoring

Selection of Smell Tests

SNUTS.JS

(Oliveira et al., 2024)

Smell	Description
Conditional Test Logic	The presence of conditional statements (**if-else**, **loops**) within the test method compromises its effectiveness.
Overcommented Test	The test method is overly detailed, making it difficult to read.
Suboptimal Assert	Using inappropriate **assertions** to verify conditions reduces clarity.
Test Without Description	The test case lacks a name or descriptive message, making it unclear.
Sensitive Equality	Using comparisons based on **toString()** or textual representation makes the tests fragile.

Table 2. Selected smells from the SNUTS.JS tool.

Pre-Refactoring

Selection of Smell Tests

Human Evaluation

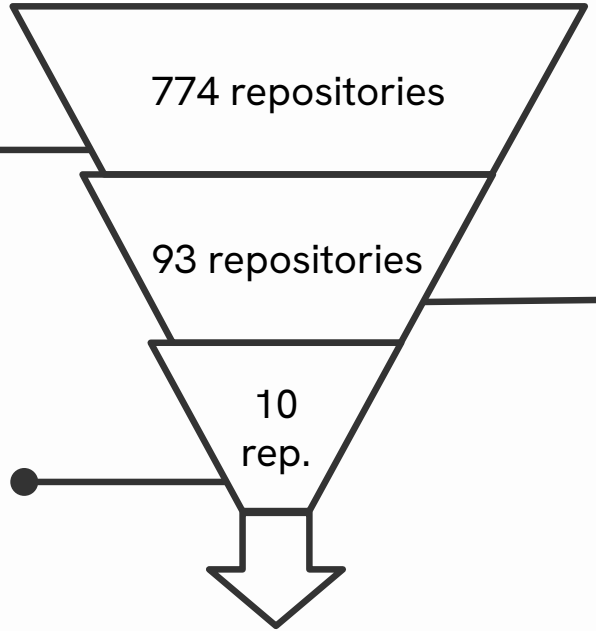
To ensure the **reliability** of **detected test smells**, a **Human Evaluation** step is required. This validation confirms whether each automatically identified smell is indeed a **true smell**, enabling the construction of a **robust** and **trustworthy dataset.**

Pre-Refactoring

Selection of Repositories

SEART
GitHub Search (GHS)

1. Javascript as the main language
2. Activity of recent development
3. Popularity and recognition



4. JavaScript is present in 75% of the source code.
5. Open-source
6. Test suite
7. Jest Framework

Script python

Selected Repositories

Repositories used in this study

Pre-Refactoring

Selection of LLMS

Business Models using **Zero-Shot Prompting**

Copilot
(GPT-4o)

Amazon Code
Whisperer

Claude Code
(Haiku 4.5)

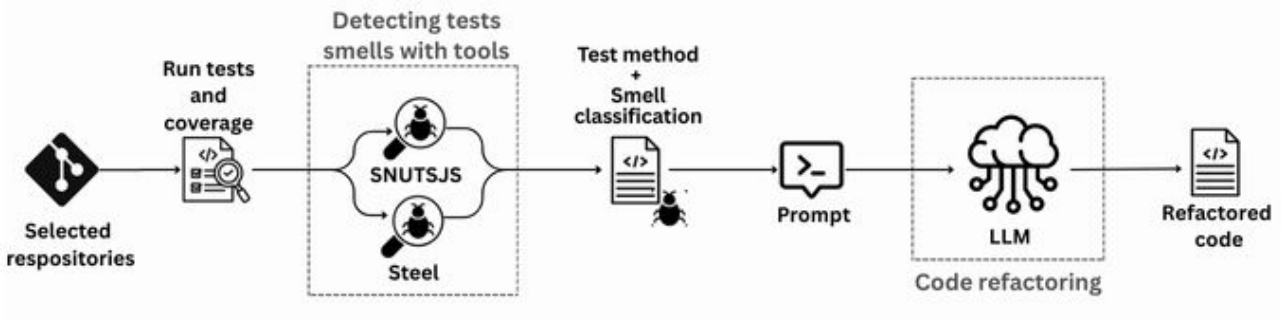
Modelos **Open-Weight** para **Auto-CoT Prompting**

Lhama-70B

Deepseek-R1

StarCoder2-15B

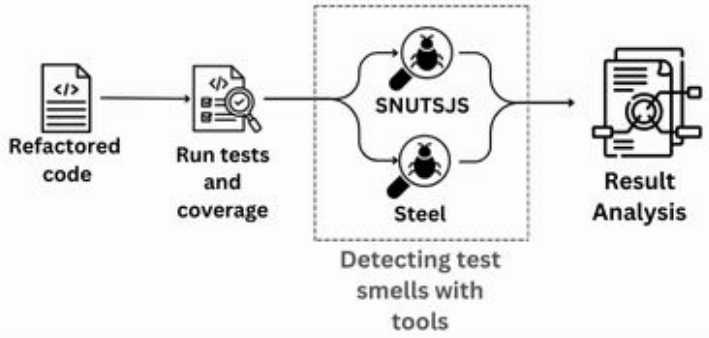
Refactoring



Prompt and Execution

- **Zero-Shot** – direct instructions without examples. Via **VSCode** extensions.
- **Auto-CoT:** The model generates a structured chain of reasoning before refactoring. Via **Hugging Face API**

Post-Refactoring Analysis



- **Functional Validation:** Execution of tests to preserve behavior and code coverage analysis.
- **Smell Removal:** Reapplication of detection tools (SNUTS.JS/Steel).
- **Structural Quality:** Analysis of ASTs for metric extraction:
  - Lines of Source Code (SLOC)
  - Cyclomatic Complexity and Cyclomatic Density
  - Halstead Effort and Bugs
  - Maintainability Index

Preliminary Results

Business Models using **Zero-Shot Prompting**

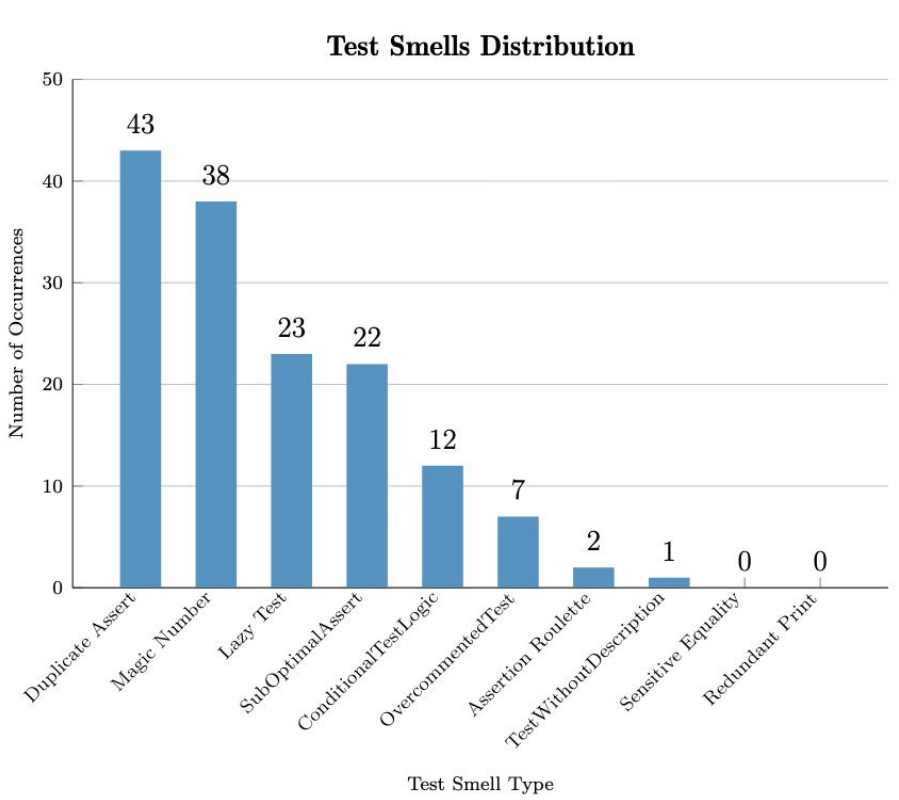
Copilot (GPT-4o)

Amazon Code Whisperer

148 occurrences identified in **10** distinct **categories**.

Key challenges identified in **JavaScript testing:**

- Bad practices in **assertions**
- Lack of **documentation** and **clarity**
- **Inadequate** structural organization



RQ1: Do **LLMs** remove **test smells** without affecting test **behavior** or **coverage**?

Changes observed:

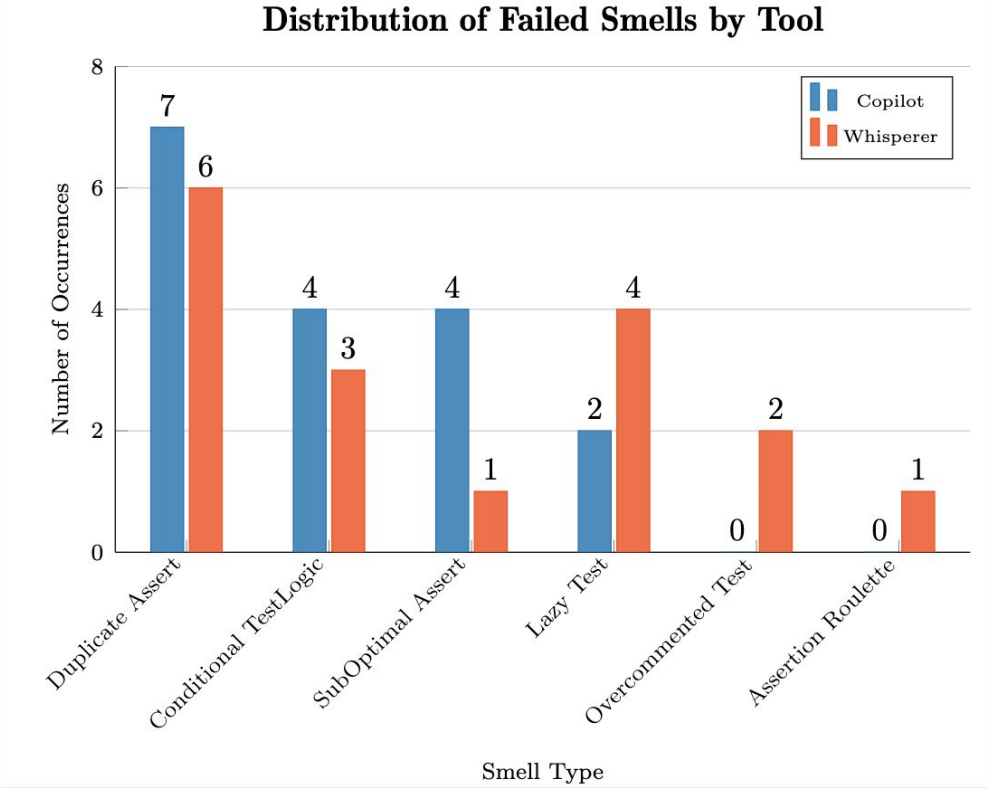
- **Copilot and Whisperer:**
  17 of **148** (11,48%)

Smells with few or no flaws:

- *Test Without Description*
- *Overcommented Test*

*more **cosmetic** nature*

Structural refactoring carries a **higher risk/impact**, while **cosmetic** adjustments are **safer**.



RQ1: Do **LLMs** remove **test smells** without affecting test **behavior** or **coverage**?

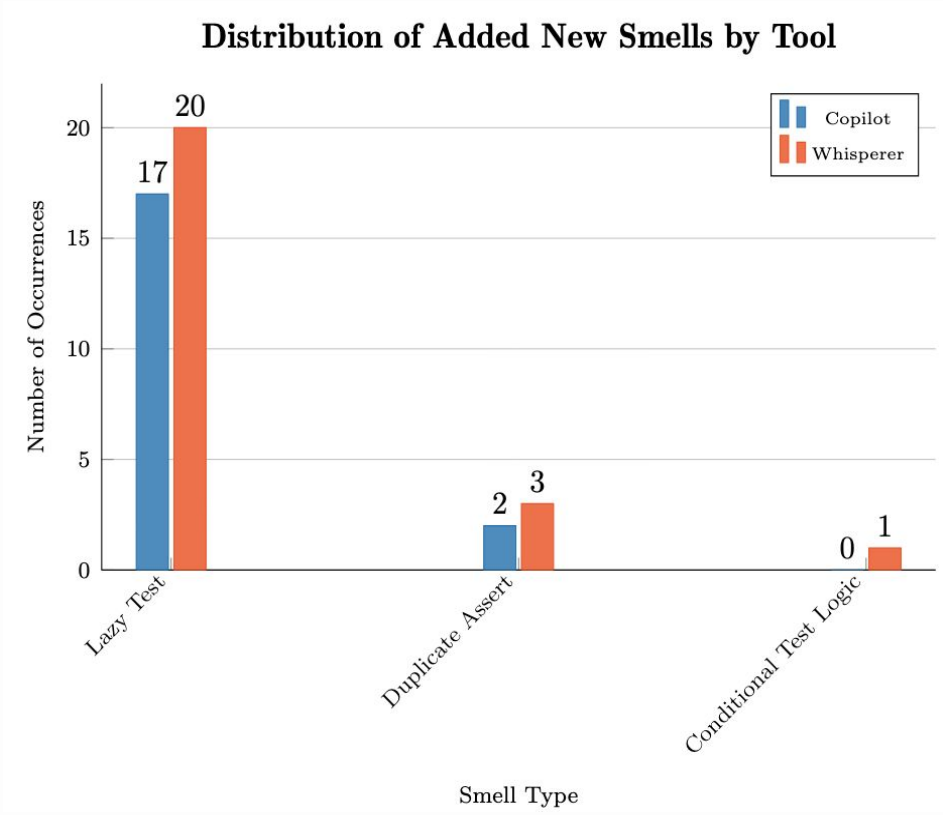
- **Copilot** with **5** cases, **altered test coverage; Whisperer** with **6** cases:
  - *Duplicate Assert*
  - *Lazy Test*
  - *Conditional Test Logic*
- **Modest** variations (almost always <1%)
- Refactoring prioritizes **preserving behavior**.

RQ2: Does refactoring using **LLMs introduce new test smells** or **degrade test quality**?

Refactoring the **Lazy Test** is the biggest **catalyst** for problems.

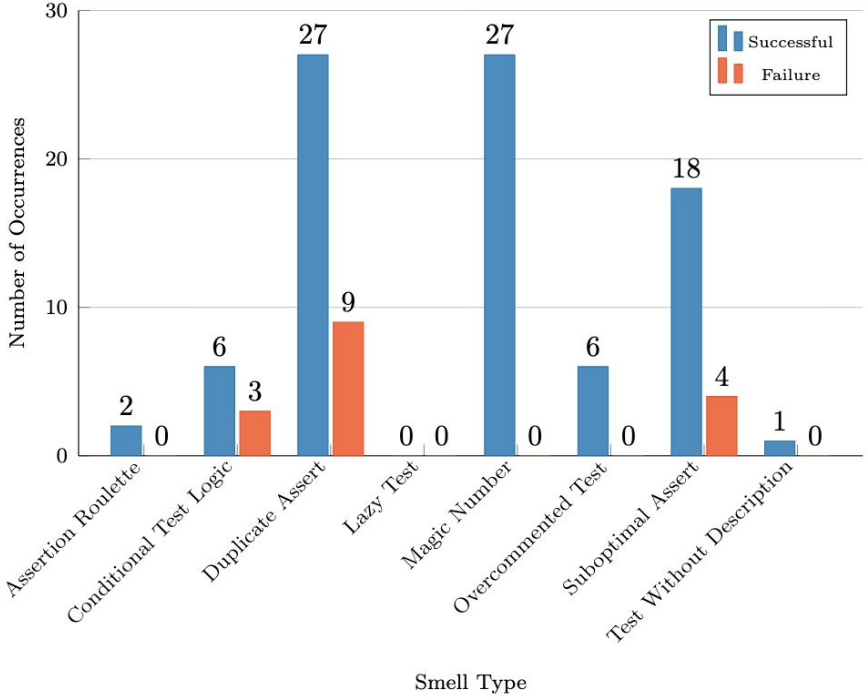
Implications:

- Refactoring **Lazy Tests** requires **extra validation**.
- Both **LLMs** face **similar limitations**.

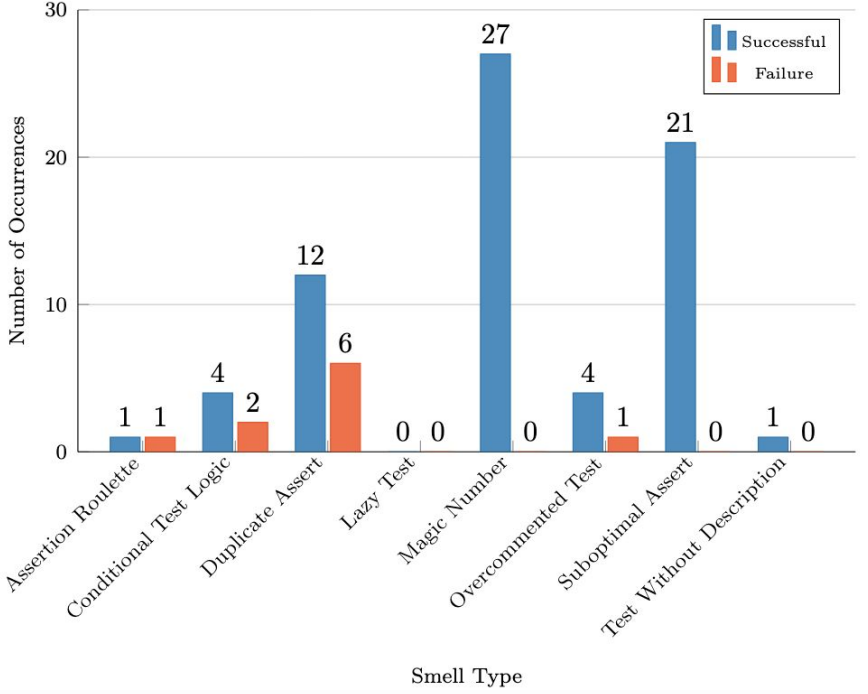


RQ3: How do **LLMs** differ in removing specific types of test smells (**strengths** and **limitations**)?

Distribution of Removed Smells on **Copilot**



Distribution of Removed Smells on **Whisperer**



RQ4: What is the **impact of refactoring** using **LLMs** on structural quality (**complexity**, **size**, **maintainability**)?

- **Code volume (Logical SLOC):**
  - Consistent **increase** in both tools (+13.2%).
  - Whisperer → higher average growth (14,338 vs. 12,662).
- **Cyclomatic complexity:**
  - **Stable** averages (~1.62)
- **Cyclomatic density:**
  - **Reduction** 14,8% (Copilot)
  - **Reduction** 21,4% (Whisperer)
- **Halstead Effort:**
  - **Copilot:** –19,3% (simplification and removal of redundancies).
  - **Whisperer:** practically stable.
- **Halstead Bugs:**
  - **Stable** (Copilot 0,017; Whisperer 0,018).
- **Manutenibilidade:**
  - Both above 95 (high).
  - Whisperer: slight drop and greater variability (IQR ↑).

→ Indicates a **better balance** between **code complexity** and **size**.

Refactoring tends to **increase code size** but **reduce complexity** (cyclomatic density and cognitive effort) – especially in Copilot, while maintaining good maintainability.

Conclusion

- **Copilot (GPT-4o): 58.78%** success rate in removing smells without altering behavior
- **Code Whisperer: 47.30%** success rate
- **15%** of refactorings alter test behavior
- Stable test coverage in **96%** of cases (preservation > expansion)
- New smells introduced in **13-16%** of cases (especially in **Lazy Test**)

Next steps

- Increase the number of smells analyzed.
- Use of more prompt **models** and **strategies (Few-Shot, Auto-CoT)**
- Expert analysis of refactored code.

References

Aljedaani, W., Peruma, A., Aljohani, A., Alotaibi, M., Mkaouer, M. W., Ouni, A., Newman, C. D., Ghallab, A., e Ludi, S. (2021). Test smell detection tools: A systematic mapping study. In Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering, páginas 170–180

Cass, S. (2024). The top programming languages 2024. IEEE Spectrum. Accessed: 2025-04-28.

Greif, S. e Burel, E. (2024). The state of javascript 2024: Testing - jest. https://2024.stateofjs.com/en-US/libraries/testing/
. Online; accessed 25 April 2025. Survey run by Devographics from Nov 13 to Dec 10, 2024 with 14,015 responses. Results published on Dec 16, 2024.

Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo,David Lo, John Grundy, and Haoyu Wang. 2024. Large language models for software engineering: A systematic literature review. ACM Transactions on Software Engineering and Methodology (2024).

Jorge, D., Machado, P., e Andrade, W. (2021). Investigating test smells in javascript test code. In Proceedings of the 6th Brazilian Symposium on Systematic and Automated Software Testing, SAST '21, página 36–45, New York, NY, USA. Association for Computing Machinery.

Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2016. An empirical investigation into the nature of test smells. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. Association for Computing Machinery, 4–15.

Oliveira, J., Mateus, L., Virg´ınio, T., e Rocha, L. (2024). Snuts.js: Sniffing nasty unit test smells in javascript. In Anais do XXXVIII Simpósio Brasileiro de Engenharia de Software, páginas 720–726, Porto Alegre, RS, Brasil. SBC.

Silva, A. C. (2022). Identificação e caracterização de test smells em javascript. Instituto de Ciências Exatas e Informática - Pontifícia Universidade, 138:52–81.

Sommerville, I. (2011). Software Engineering, 9/E. Pearson Education India.

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

Yu, S., Fang, C., Ling, Y., Wu, C., e Chen, Z. (2023). Llm for test script generation and migration: Challenges, capabilities, and opportunities. In 2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS), páginas 206–217.

Thanks!

Do you have any questions?
gabrielamaralsousa@gmail.com

Artifacts available at Zenodo
https://doi.org/10.5281/zenodo.17058737