Análise da Ocorrência de Test Smells e Refatorações em Projetos Java Open-Source no GitHub

Ian Nascimento[1], **Lucas Nogueira**[1], **Marcelo Aguilar**[1], **Pablo Benevenuto**[1], **Raul Fernandes**[1], **Victor Friche**[1]

[1]Instituto de Ciências Exatas e Informática – Departamento de Engenharia de Software
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
CEP 30535-000 – Belo Horizonte – MG – Brazil

ian.nascimento@sga.pucminas.br, lnmcarvalho@sga.pucminas.br,
maadalmeida@sga.pucminas.br, pablo.amancio@sga.pucminas.br,
raul.fernandes@sga.pucminas.br, victor.friche@sga.pucminas.br

Abstract. *This study investigates the frequency and types of test smells present in tests of popular open-source Java systems, as well as the refactorings applied to mitigate them. Through automated analysis of 1,000 repositories and inspection of commit diffs, the relative incidence of each smell is quantified, refactoring patterns are identified, and the proportion of effectively addressed smells is assessed. The results reveal that between* 40% *and* 60% *of test classes contain smells, especially Assertion Roulette, Unknown Test, and Exception Catching Throwing. However, only about* 8,9% *of these smells are actually refactored in practice. Refactorings are mostly concentrated in low-impact strategies, such as RemoveRedundantPrint and RenameTestClass. These findings suggest that poor testing practices are widespread and that there is little effort toward correcting them, which may compromise the long-term maintainability of test code.*

Resumo. *Este trabalho investiga a frequência e os tipos de test smells presentes em testes de sistemas Java open-source, bem como as refatorações aplicadas para mitigá-los. Por meio de análise automática de 1 000 repositórios e análise de diffs de commits, quantificam-se a incidência relativa de cada smell, identificam-se padrões de refatoração e avalia-se a proporção de tratamentos efetivos. Os resultados revelam que cerca de 40% a 60% das classes de teste contêm smells, especialmente Assertion Roulette, Unknown Test e Exception Catching Throwing, mas apenas cerca de 8.9% desses smells são refatorados na prática. Observa-se concentração das refatorações em abordagens de baixo impacto (ex.: RemoveRedundantPrint, RenameTestClass). Conclui-se que há grande disseminação de más práticas nos testes e escasso esforço de correção, o que pode comprometer a manutenibilidade a longo prazo.*

1. Introdução

Testes são amplamente utilizados no desenvolvimento de software moderno, integrando-se a *pipelines* de integração contínua e assegurando a qualidade de sistemas [4, 13]. Entretanto, esses testes frequentemente contêm problemas estruturais conhecidos como *test smells*, que dificultam a compreensão, manutenção e

evolução do código de teste [13]. Entre os *test smells* mais recorrentes estão *Assertion Roulette*, *Magic Number Test*, *Eager Test* e *Exception Catching Throwing* [4].

A detecção e mitigação desses problemas têm sido amplamente estudadas. Ferramentas como JNose e TSDetect automatizam a identificação de *test smells* em larga escala [12, 1]. Estudos mostram que a densidade de *test smells* é especialmente alta em classes de teste, principalmente em projetos Java [11], sendo raramente removidos mesmo após sucessivas evoluções do software [12]. Esse padrão também foi observado em projetos Android, onde a recorrência de *test smells* pode comprometer a qualidade dos testes móveis [7].

Além disso, pesquisas evidenciam que a presença desses *smells* está associada a maior propensão a falhas, o que reforça a necessidade de tratá-los adequadamente [8, 5, 11]. Apesar da existência de catálogos de refatorações específicas [3] e estratégias sistemáticas para detecção [6], ainda há lacunas quanto à adoção prática dessas recomendações pelos desenvolvedores. Trabalhos como [1, 11] indicam que muitos *smells* permanecem no código sem correção, mesmo quando claramente identificáveis.

Outros estudos apontam que práticas como a rastreabilidade entre testes e unidades testadas auxiliam na manutenção e evolução segura do código [9]. Ainda assim, observa-se uma carência de abordagens automatizadas eficazes no apoio à remoção de *smells*. De forma complementar, a análise de características de testabilidade e antipadrões estruturais pode fornecer insights adicionais sobre a manutenção de testes [2].

Diante disso, uma pergunta central orienta esta pesquisa: até que ponto os *test smells* são efetivamente tratados na prática?

Neste artigo, conduzimos uma análise automatizada em larga escala, envolvendo 1.000 repositórios Java *open-source*, utilizando a ferramenta JNose para detectar *test smells* e, por meio da análise dos *diffs* de commits, identificar as principais refatorações realizadas [12, 11].

Como principais resultados, identificamos que mais de 40% das classes de teste analisadas apresentaram algum tipo de *test smell*, sendo que apenas cerca de 8,9% dessas ocorrências foram efetivamente tratadas por meio de refatorações. Além disso, observamos que as refatorações realizadas concentram-se, majoritariamente, em mudanças de baixo impacto, como a adição de mensagens em asserções e a renomeação de classes de testes.

Estrutura do artigo: a Seção 3 apresenta os principais trabalhos relacionados à detecção de *test smells* e estratégias de refatoração. A Seção 4 descreve a metodologia adotada para a coleta e análise dos dados. A Seção 5 expõe e discute os resultados obtidos. Por fim, a Seção 6 apresenta as considerações finais e sugestões para pesquisas futuras.

As questões de pesquisa (RQs) deste estudo são:

- **RQ1:** Quais são os padrões de ocorrência dos diferentes tipos de *Test Smells* em projetos Java open-source?

- **RQ2:** Como os *Test Smells* são refatorados?
- **RQ3:** De que forma os *Test Smells* identificados são tratados ao longo do tempo nos projetos analisados?

2. Motivação

Testes desempenham um papel fundamental na garantia da qualidade e da confiabilidade de sistemas de software modernos. No entanto, a simples presença de testes não assegura sua efetividade. Quando mal estruturados, esses testes podem conter *test smells*, ou seja, pode conter sintomas de problemas recorrentes que comprometem a clareza, manutenibilidade e eficácia dos testes [13, 4].

O *test smell* conhecido como *Assertion Roulette*, por exemplo, ocorre quando múltiplas asserções são utilizadas sem mensagens explicativas, tornando difícil identificar qual falhou em caso de erro [4]. Esse tipo de problema não apenas dificulta o diagnóstico, mas também reduz o valor do teste como ferramenta de apoio à evolução segura do sistema.

Listing 1. Código com test smell *Assertion Roulette*

```
@Test
public void testCalculateTotal() {
    assertEquals(100, calculateTotal(50, 50));
    assertEquals(200, calculateTotal(150, 50));
}
```

Para eliminar o *Test Smell* Assertion Roulette, a literatura sugere três estratégias de refatoração: (i) adicionar uma mensagem explicativa às asserções [13], (ii) dividir o teste em métodos separados que validam condições individuais [4], ou (iii) agrupar múltiplas asserções com o método `assertAll`, disponível no JUnit 5, para que todas sejam executadas e relatadas conjuntamente [10].

A refatoração mais observada neste estudo foi a adição de mensagens explicativas nas asserções, identificada como `AddAssertionMessage`. Essa prática facilita o diagnóstico de falhas ao tornar explícito o motivo esperado da verificação. O Listing 2 ilustra um exemplo dessa refatoração.

Listing 2. Refatoração adicionando mensagem de asserção para eliminar o *Assertion Roulette*

```
@Test
public void testCalculateTotal() {
    assertEquals("Soma de 50 e 50 deve ser 100", 100,
    calculateTotal(50, 50));
    assertEquals("Soma de 150 e 50 deve ser 200", 200,
    calculateTotal(150, 50));
}
```

Alguns estudos argumentam que o *test smell Assertion Roulette* está se tornando menos relevante, especialmente com a adoção da refatoração que adiciona mensagens explicativas, considerando que os frameworks de teste modernos já indicam explicitamente qual asserção falhou e por quê [10]. No entanto, observamos

que, em projetos reais, os desenvolvedores ainda tendem a corrigir esse *smell* por meio da adição de mensagens descritivas nas asserções, em vez de aplicar outras refatorações sugeridas na literatura, como a separação de testes em métodos distintos ou o uso de `assertAll`.

Diante disso, entender como as refatorações sugeridas pela literatura são aplicadas na prática ajuda a direcionar melhorias em ferramentas de análise e a desenvolver estratégias mais eficazes para apoiar os desenvolvedores na manutenção de testes de qualidade.

3. Trabalhos Relacionados

A pesquisa sobre qualidade de código de teste tem se intensificado nos últimos anos na área de Engenharia de Software, com foco na identificação de *test smells* e na aplicação de refatorações para melhorar a legibilidade e a manutenibilidade dos testes [4, 13].

O conceito de *test smells* foi introduzido por Van Deursen et al. [13], que os definiram como sintomas de falhas estruturais em testes, prejudicando sua compreensão e manutenção. Essa definição serve como base conceitual para a nossa investigação, ao orientar a identificação e caracterização dos *smells* encontrados.

Meszaros [4] ampliou esse conceito ao apresentar um catálogo detalhado de manifestações comuns de *test smells*, além de propor diversas sugestões de refatoração. Seu trabalho fundamenta nossa escolha das categorias de *smells* analisadas e das possíveis estratégias de correção.

Martins et al. [3] apresentaram um catálogo de estratégias específicas de refatoração para *test smells*, mapeando transformações recomendadas como *AddAssertionMessage*, *SplitEagerTest* e *ReplaceMagicNumberWithConstant*. Esse catálogo foi utilizado como referência para classificar as refatorações identificadas em nosso estudo.

Estudos empíricos, como os conduzidos por Tufano et al. [12], analisaram a evolução de *smells* em código de produção, demonstrando que muitos são introduzidos nas versões iniciais e raramente removidos. Essa evidência motiva nossa investigação, ao indicar que fenômeno semelhante pode ocorrer em código de teste.

Bavota et al. [1] e Spadini et al. [11] constataram que a densidade de *test smells* é particularmente elevada em classes de teste, especialmente em projetos Java, e que a frequência de correção por refatoração é baixa. Esses achados reforçam a relevância do nosso trabalho, que busca compreender não apenas a presença, mas também o tratamento dessas más práticas ao longo do tempo.

Por fim, o presente trabalho se diferencia por realizar uma análise automatizada e em larga escala, envolvendo 1.000 repositórios Java *open-source*, ampliando a compreensão sobre a incidência de *test smells* e as refatorações efetivamente aplicadas, aspectos até então pouco explorados em estudos anteriores.

4. Metodologia Proposta

A Figura 1 ilustra a metodologia adotada neste estudo, composta por quatro etapas principais: Mineração de Dados (Seção 4.1), Armazenamento dos Dados (Seção 4.2),

Processamento (Seção 4.3) e Análise Estatística (Seção 4.4). Essa abordagem permitiu investigar a ocorrência de *test smells* e as refatorações aplicadas em projetos reais de forma escalável e sistemática.

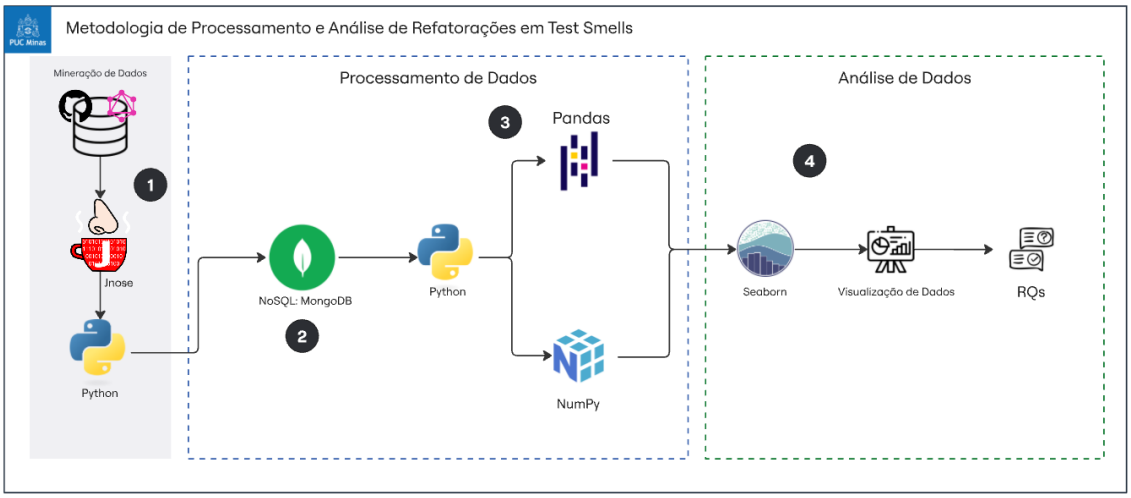

Figure 1. Etapas da metodologia de processamento e análise de refatorações em *test smells*

4.1. Mineração de Dados

Primeiramente, foram selecionados os 1.000 projetos Java mais populares do GitHub, ordenados com base no número de estrelas. O número de estrelas foi adotado como um indicador amplamente aceito e de fácil acesso para estimar a popularidade de projetos na plataforma. Como a API do GitHub impõe um limite de 1.000 requisições por busca, a amostra foi restrita a esse conjunto inicial de repositórios.

Em seguida, aplicou-se um filtro para identificar apenas os repositórios que fazem uso do framework de testes JUnit, uma vez que o foco deste estudo está na detecção e análise de *test smells* e refatorações em código de teste. Essa filtragem considerou a presença de elementos característicos do JUnit na estrutura dos projetos ou em seus arquivos de configuração de *build*, como dependências em `pom.xml`, `build.gradle` ou anotações `@Test` em classes de teste.

Como resultado, foram identificados 339 repositórios que atendem a esses critérios e que compõem a amostra final analisada.

4.2. Armazenamento dos Dados

Os dados brutos extraídos pelo JNose foram armazenados em um banco de dados NoSQL, utilizando o MongoDB. Essa estrutura permitiu maior flexibilidade para lidar com documentos JSON gerados automaticamente e facilitou o acesso a diferentes coleções de informações (ex.: repositórios, classes de teste, smells identificados).

4.3. Processamento dos Dados

Com os dados disponíveis no banco, scripts adicionais em Python foram utilizados para realizar a limpeza, transformação e análise inicial dos dados. Para isso, utilizaram-se as bibliotecas `pandas` e `NumPy`, permitindo o agrupamento por tipo

de *smell*, projeto, frequência, e ainda o cruzamento com refatorações identificadas. Os resultados estruturados foram salvos em arquivos intermediários para posterior visualização.

4.4. Análise Estatística e Visualização

Por fim, os dados processados foram utilizados para gerar gráficos e relatórios estatísticos com o auxílio da biblioteca Seaborn. As visualizações incluíram gráficos de densidade, correlação, frequência relativa e média de *smells* por classe de teste, entre outros. Com base nessas análises, foram respondidas as Questões de Pesquisa (RQs) definidas no escopo do estudo.

4.5. Detecção de Refatorações

Para identificar automaticamente as refatorações aplicadas sobre os *test smells*, elaboramos um catálogo de padrões baseado em expressões regulares. Esses padrões permitem detectar transformações recorrentes no código de teste associadas a boas práticas recomendadas na literatura.

A Tabela 1 apresenta os principais padrões implementados, bem como a descrição de cada refatoração e o *test smell* associado que se busca mitigar.

Table 1. Resumo da correlação entre refatorações detectadas e seus respectivos *test smells*

Refatoração Detectada	Descrição	Test Smell Associado
ReplaceTryCatchWithAssertThrows	Substitui blocos try−catch por chamadas assertThrows para verificar exceções de forma idiomática.	Exception Catching Throwing
ReplaceAssertTrueWithAssertFalse	Substitui assertTrue (!...) por assertFalse (...) para melhorar a clareza e evitar dupla negação.	Assertion Roulette
ReplaceAssertFalseWithAssertTrue	Substitui assertFalse (!...) por assertTrue (...) pelo mesmo motivo de clareza.	Assertion Roulette
AddAssertionMessage	Adiciona mensagens explicativas nas asserções para facilitar o diagnóstico em caso de falha.	Assertion Roulette
RemoveIgnoredTest	Remove anotações @Ignore ou @Disabled para reativar testes que estavam desabilitados.	Ignored Test
RemoveRedundantPrint	Remove comandos System.out.println() redundantes, que podem poluir os logs de execução de testes.	Print Statement
RenameTestClass	Renomeia classes de teste, mantendo a convenção ∗Test.java, melhorando a organização e clareza.	Unknown Test
RemoveEmptyTestMethod	Remove métodos de teste vazios anotados com @Test, que não possuem nenhuma lógica implementada.	Empty Test
ReplaceSleepWithAwaitility	Substitui Thread.sleep() por APIs da biblioteca Awaitility, melhorando a confiabilidade de testes assíncronos.	Sleepy Test

Esses padrões foram aplicados sistematicamente na análise dos *diffs* de commits extraídos dos repositórios, permitindo identificar as principais refatorações realizadas para tratar os *test smells* detectados previamente com a ferramenta JNose.

5. Resultados

A partir da análise dos dados extraídos de repositórios públicos populares do GitHub, foram obtidos os seguintes resultados em relação às questões de pesquisa propostas:

5.1. RQ1: Quais são os padrões de ocorrência dos diferentes tipos de *Test Smells* em projetos Java ?

A Figura 2 apresenta a distribuição do percentual de classes de teste afetadas por *test smells* em cada repositório da amostra. Observa-se que, na maioria dos casos, entre 40 % e 80 % das classes estão afetadas, com um pico de densidade próximo a 60 %. Isso sugere que a presença de *test smells* é recorrente e significativa na base de testes dos projetos analisados. Ainda que existam repositórios com baixa ou altíssima cobertura por *smells* (próximos de 0 % ou 100 %), esses casos são menos frequentes. Esses resultados reforçam a ideia de que práticas inadequadas em testes estão disseminadas, mesmo em projetos populares do GitHub.

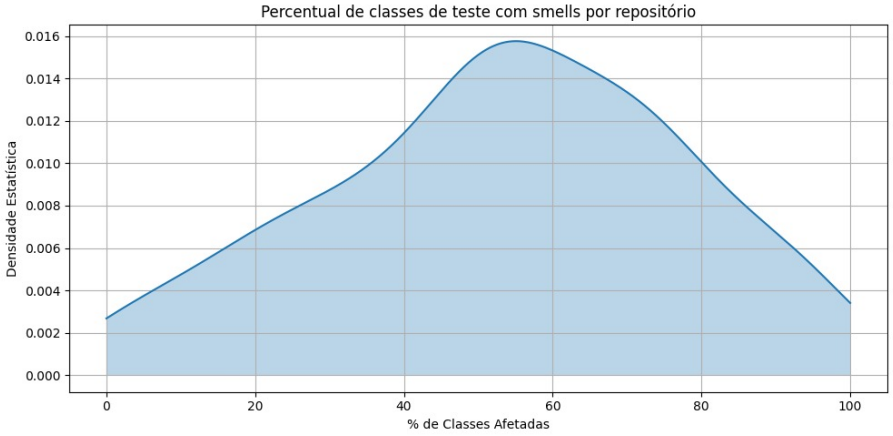

Figure 2. Percentual de testes afetados por *Test Smells*

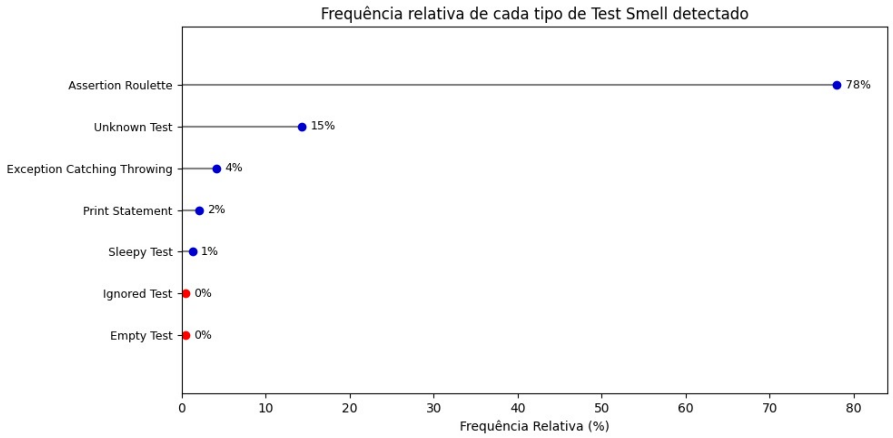

Figure 3. Frequência relativa de *Test Smells* identificados

A Figura 3 apresenta a frequência relativa dos diferentes tipos de *test smells* identificados ao longo dos repositórios analisados. O resultado evidencia uma concentração significativa no smell *Assertion Roulette*, que representa 78 % de todas as ocorrências detectadas. Em seguida, aparece o *Unknown Test*, com 15 %, e o *Exception Catching Throwing*, com 4 %.

Outros smells como *Print Statement* (2 %) e *Sleepy Test* (1 %) foram detectados com menor frequência. Já os smells *Ignored Test* e *Empty Test* não foram identificados, sugerindo que ou são menos recorrentes na prática ou que a ferramenta utilizada apresentou limitações na sua detecção automática.

Esses resultados sugerem que certos smells, como o *Assertion Roulette*, são amplamente disseminados em testes de software, possivelmente por se manifestarem em situações comuns de testes com múltiplas asserções não explicativas. Isso reforça a importância de iniciativas voltadas à melhoria da clareza e manutenção dos testes.

A Figura 4 apresenta a distribuição da quantidade média de *test smells* por classe de código de teste. Observa-se que a maior parte das classes concentra até 10 *smells*, com um pico de densidade em torno de 2 ocorrências por classe.

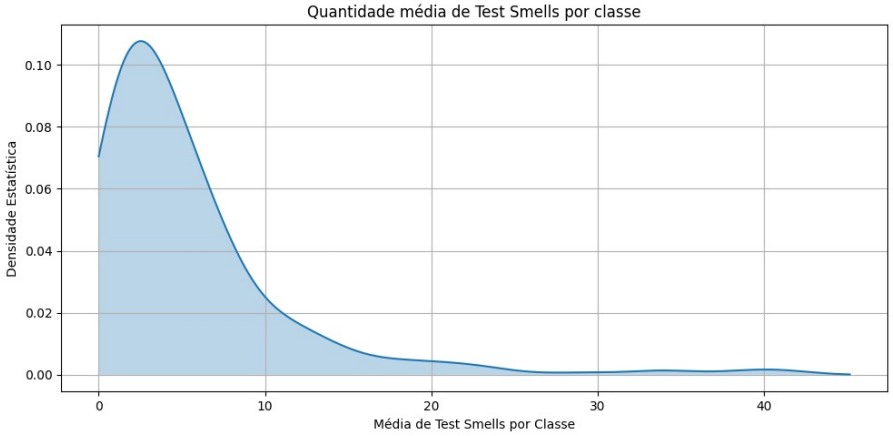

Figure 4. Quantidade média de *Test Smells* por classe

Apesar da média geral de *test smells* por classe ser relativamente baixa quando analisada por repositório, os dados revelam uma grande variação entre os projetos. Em alguns casos, a média ultrapassa 30 *smells* por classe de teste, indicando que determinados repositórios concentram uma quantidade significativa de problemas estruturais.

Esse comportamento sugere que, embora muitos projetos apresentem testes razoavelmente limpos, há casos extremos em que os *test smells* estão altamente concentrados, o que pode comprometer a legibilidade, a confiabilidade e a manutenibilidade dos testes nesse conjunto específico. Portanto, é importante observar a média de *smells* por classe, pois ela ajuda a entender como está a qualidade dos testes em cada projeto.

5.2. RQ2: Como os *Test Smells* são refatorados na prática?

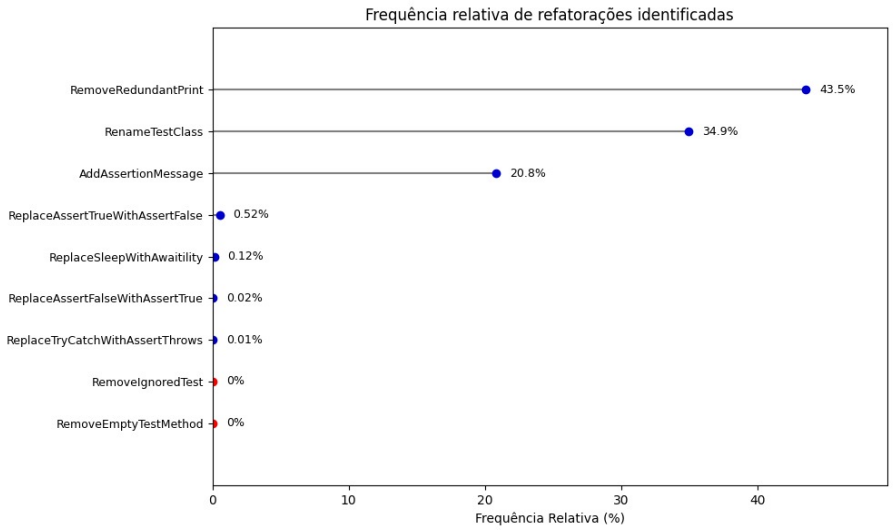

Figure 5. Frequência de cada tipo de refatoração identificado

A Figura 5 apresenta a distribuição relativa dos tipos de refatorações detec-

tadas no histórico de commits dos repositórios analisados. Nota-se que as três refatorações mais comuns *RemoveRedundantPrint* (43,5 %), *RenameTestClass* (34,9 %) e *AddAssertionMessage* (20,8 %) representam quase a totalidade das transformações aplicadas.

As demais refatorações aparecem com frequência significativamente inferior: *ReplaceAssertTrueWithAssertFalse* (0,52 %), *ReplaceSleepWithAwaitility* (0,12 %), *ReplaceAssertFalseWithAssertTrue* (0,02 %) e *ReplaceTryCatchWithAssertThrows* (0,01 %). Já as refatorações *RemoveIgnoredTest* e *RemoveEmptyTestMethod* não foram observadas em nenhuma das revisões analisadas.

Esses dados indicam uma preferência por refatorações de baixo custo e impacto mais superficial, voltadas à organização e clareza do código de teste. Refatorações mais estruturais como a substituição de blocos `try-catch` ou a reestruturação de asserções parecem ser aplicadas com pouca frequência na prática, possivelmente por exigirem maior conhecimento técnico ou demandarem mais esforço de revisão.

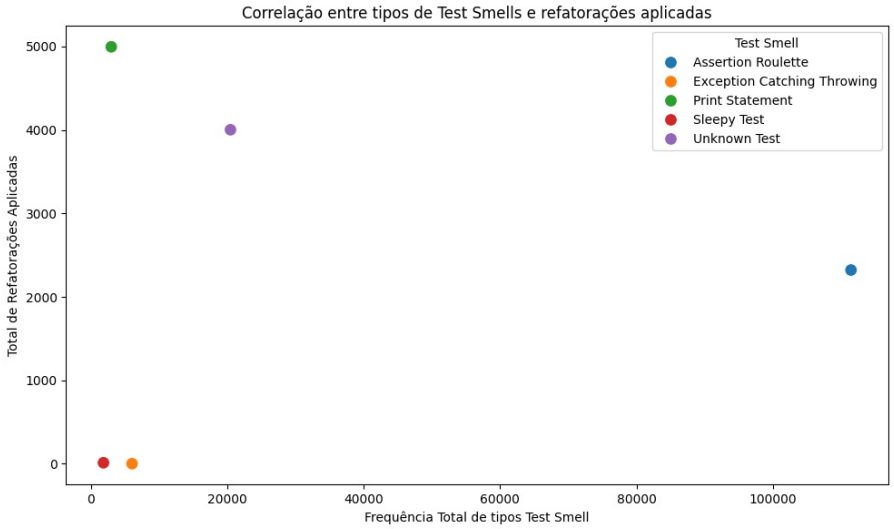

Figure 6. Correlação entre tipos de *Test Smells* e refatorações aplicadas

A Figura 6 apresenta a relação entre a quantidade total de ocorrências de cada tipo de *test smell* (eixo X) e o total de refatorações associadas a esses *smells* (eixo Y). Embora o *Assertion Roulette* seja o *smell* mais frequente na amostra, observa-se que ele não é o mais refatorado ocupando uma posição inferior em relação a *Print Statement* e *Unknown Test*.

Os dados indicam que os *smells* menos frequentes podem receber mais atenção prática dos desenvolvedores, enquanto os mais comuns como o próprio *Assertion Roulette* permanecem em grande parte não tratados. Isso pode refletir a percepção de que certos problemas são menos críticos ou difíceis de serem detectados e corrigidos no contexto do projeto.

Dessa forma, a correlação fraca entre a quantidade de *smells* e as refatorações realizadas sugere uma lacuna entre o que é identificado automaticamente pelas fer-

ramentas e o que é efetivamente tratado pelos desenvolvedores, reforçando a necessidade de estratégias mais eficazes de conscientização e suporte à manutenção de testes.

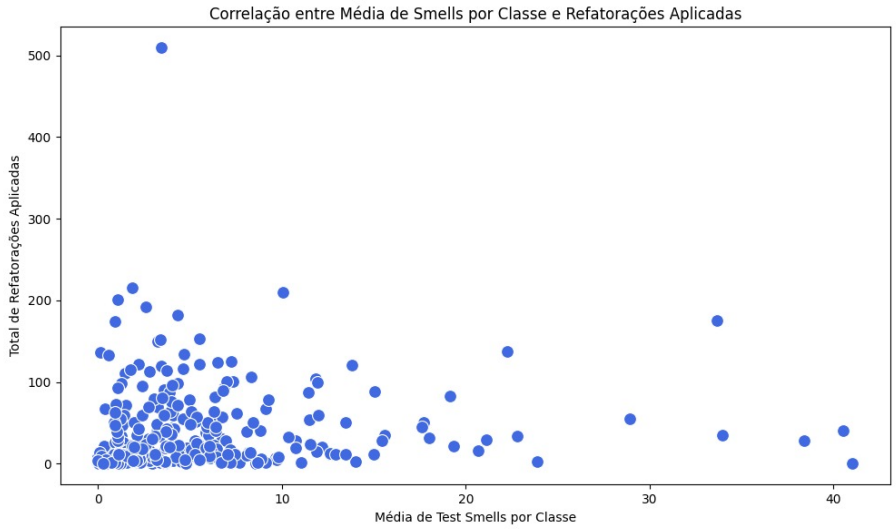

Figure 7. Correlação entre quantidade média de *Test Smells* e refatorações aplicadas

A Figura 7 mostra a dispersão entre a média de *test smells* por classe (eixo X) e a quantidade total de refatorações aplicadas em cada repositório (eixo Y). Esperava-se que repositórios com maior concentração média de *smells* aplicassem mais refatorações, no entanto, os dados revelam uma correlação fraca ou inexistente entre essas variáveis.

Observa-se que mesmo projetos com baixa média de *smells* podem aplicar centenas de refatorações, enquanto outros com valores elevados de smells por classe exibem pouca ou nenhuma ação corretiva. Isso sugere que a densidade de problemas não é necessariamente um fator decisivo para que os desenvolvedores apliquem refatorações sistemáticas.

Esses achados reforçam que fatores externos como priorização de backlog, conhecimento técnico da equipe ou restrições de tempo podem influenciar mais na aplicação de refatorações do que a própria presença ou intensidade dos *smells*.

5.3. RQ3: De que forma os *Test Smells* identificados são tratados ao longo do tempo nos projetos analisados?

A Figura 8 mostra a quantidade total de *test smells* encontrados nos projetos analisados e quantos deles foram realmente refatorados. Ao todo, foram detectadas 144.447 ocorrências de *smells*, das quais apenas 12.922 foram tratadas com alguma refatoração identificável. Isso representa cerca de 8,9% do total. Os outros 131.525 *smells* (ou seja, 91,1%) permaneceram no código sem nenhuma modificação.

Esse resultado mostra que, mesmo com a presença de problemas conhecidos nos testes, a maioria deles não é corrigida na prática. A baixa taxa de refatoração sugere que os desenvolvedores, em muitos casos, não priorizam ou não percebem a necessidade de tratar esses problemas. Com o tempo, isso pode afetar a qualidade

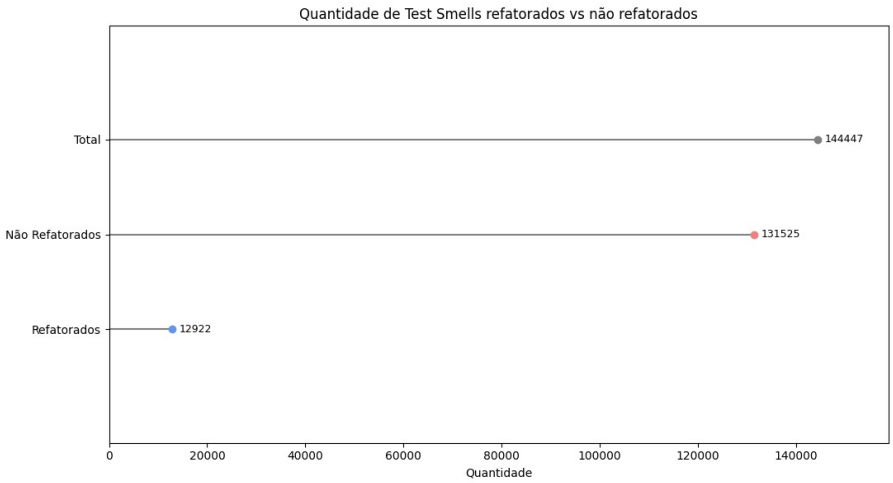

Figure 8. Quantidade de *Test Smells* refatorados identificados e Quantidade de *Test Smells* que persistem sem refatoração

dos testes, tornando-os mais difíceis de entender e manter. Esse cenário reforça a importância de incentivar a aplicação de boas práticas de manutenção de testes nos projetos.

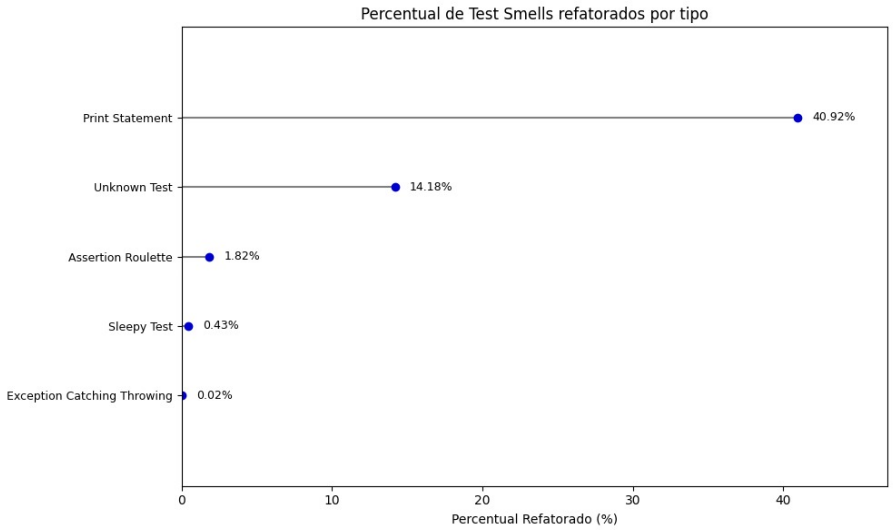

Figure 9. Quantidade de *Test Smells* não refatorados

A Figura 9 mostra o percentual de *test smells* refatorados em relação ao total de ocorrências de cada tipo. Observa-se uma grande disparidade entre os diferentes *smells*. O *Print Statement* é o tipo com maior taxa de tratamento, com 40,92% das ocorrências corrigidas. Em seguida, o *Unknown Test* aparece com 14,18% de refatorações.

Por outro lado, os *smells* mais comuns, como o *Assertion Roulette*, apresentam taxas de refatoração muito baixas — apenas 1,82% neste caso. O *Exception Catching Throwing*, por exemplo, foi praticamente ignorado, com apenas 0,02% de ocorrências tratadas.

Esses dados sugerem que os desenvolvedores tendem a aplicar refatorações apenas em casos simples e pontuais, negligenciando *smells* mais estruturais ou que demandam maior esforço para correção.

6. Conclusão

A qualidade dos testes é essencial para garantir a manutenibilidade e confiabilidade de sistemas de software modernos. No entanto, a presença de *test smells* compromete essa qualidade, prejudicando a legibilidade, dificultando a evolução dos testes e aumentando o risco de falhas. Este estudo apresentou uma análise empírica em larga escala sobre a ocorrência e o tratamento de *test smells* em projetos Java *open-source*, baseada em 1.000 repositórios populares hospedados no GitHub.

Utilizando a ferramenta JNose para detecção automática de *test smells* e análise de *diffs* de commits, respondemos a três questões de pesquisa: (i) padrões de ocorrência e tipos predominantes de *smells*, (ii) refatorações mais comuns aplicadas na prática, e (iii) proporção de *test smells* efetivamente tratados ao longo do tempo.

Entre os principais achados, observamos que mais de 40% das classes de teste analisadas apresentam ao menos um *test smell*, sendo o *Assertion Roulette* o tipo mais recorrente, com 78% das ocorrências relativas. Apesar da alta incidência, apenas cerca de 9% dos *test smells* detectados foram tratados por meio de refatorações, e os tipos de refatoração aplicados concentram-se principalmente em mudanças simples, como remoção de `prints` e renomeações de classes adicionando a palavra *"Test"*.

Além disso, evidenciamos que há uma baixa correlação entre a quantidade de *test smells* presentes nas classes e o número de refatorações realizadas, o que sugere que a presença desses problemas nem sempre é tratada de forma sistemática pelos desenvolvedores. Ainda, mesmo os *test smells* amplamente discutidos na literatura, como o próprio *Assertion Roulette*, possuem baixíssimo percentual de correção na prática, o que reforça o desalinhamento entre as recomendações acadêmicas e o comportamento observado em projetos reais.

Este trabalho contribui para o avanço da compreensão sobre como os *test smells* são tratados (ou ignorados) na prática e oferece evidências que podem orientar tanto a comunidade acadêmica quanto os profissionais na priorização de técnicas de manutenção e boas práticas de testes.

Como trabalhos futuros, sugerimos expandir a análise para incluir outras linguagens de programação além de Java, investigar mais profundamente os motivos pelos quais os desenvolvedores deixam de aplicar refatorações e avaliar o impacto dessas decisões na qualidade dos testes ao longo do tempo. Além disso, propomos o aprimoramento das heurísticas de detecção por meio da incorporação de análise sintática mais aprofundada, permitindo identificar situações mais complexas como a remoção de testes comentados no código e a presença de *Eager Tests* com múltiplas responsabilidades, os quais poderiam ser tratados por meio da divisão de testes grandes em métodos menores e mais específicos.

References

[1] G. Bavota, B. Russo, R. Oliveto, M. Di Penta, and A. De Lucia. Are test smells really harmful? an empirical study. *Empirical Software Engineering*, 20(4):1052–1094, 2015.

[2] M. A. Maiga, F. Khomh, G. Antoniol, and Y.-G. Guéhéneuc. Support to characterize and evaluate testability anti-patterns. In *Proceedings of the 16th European Conference on Software Maintenance and Reengineering*, pages 187–196, 2012.

[3] J. Martins, P. Silva, and M. Almeida. A comprehensive catalog of test smell refactorings, 2024. Preprint.

[4] G. Meszaros. *xUnit Test Patterns: Refactoring Test Code*. Addison-Wesley, 2007.

[5] G. Oliva, F. Petrillo, A. Hora, and M. T. Valente. Are test smells really harmful? a controlled experiment. In *Proceedings of the International Conference on Software Engineering (ICSE)*, 2022.

[6] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, A. D. Lucia, and D. Poshyvanyk. On the diffuseness and the impact of code smells: A large scale empirical investigation. In *IEEE Transactions on Software Engineering*, volume 43, pages 1067–1082, 2016.

[7] A. Peruma, N. Nelson, and C. Jensen. Test smells in android: How often do they occur? *Empirical Software Engineering*, 25(4):2928–2960, 2020.

[8] A. Qusef, A. Hamou-Lhadj, and A. Lbath. A study of the fault-proneness of test smells. In *Proceedings of the 10th International Conference on Quality Software (QSIC)*, pages 303–308, 2011.

[9] B. V. Rompaey and S. Demeyer. Establishing traceability links between unit test cases and units under test. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM)*, pages 81–90, 2007.

[10] G. Soares, G. Pinto, and M. T. Valente. Refactoring test smells with junit 5: Why should developers keep up-to-date? *Software Quality Journal*, 32:641–679, 2024.

[11] D. Spadini, M. Aniche, and A. Bacchelli. When testing hurts: An empirical study on test smells. In *Proceedings of the International Conference on Mining Software Repositories*, pages 142–153, 2018.

[12] M. Tufano, G. Bavota, M. D. Penta, R. Oliveto, and D. Poshyvanyk. An empirical study on developer-related factors characterizing fix-inducing commits. *Journal of Software: Evolution and Process*, 29(1), 2016.

[13] A. van Deursen, L. Moonen, A. van den Bergh, and G. Kok. Refactoring test code. In *Extreme Programming Perspectives*, pages 183–195, 2001.