9º ENCONTRO DE CIÊNCIA E TECNOLOGIA DO IFSUL - CÂMPUS BAGÉ

Análise de *Design Smells* em Projetos Java de Código Aberto

FARIAS, O, G.[1], PFEIFER, T. V.[2], BETEMPS, C. M.[3]

[1] Universidade Federal do Pampa (UNIPAMPA) – Bagé – RS –Brasil – otaviogarcia.aluno@unipampa.edu.br

[2] Universidade Federal do Pampa (UNIPAMPA) – Bagé – RS–Brasil – thiagopfeifer.aluno@unipampa.edu.br

[3] Universidade Federal do Pampa (UNIPAMPA) – Bagé – RS – Brasil – carlosbetemps@unipampa.edu.br

RESUMO

Este estudo analisou a ocorrência de *design smells* em projetos Java, anomalias que prejudicam a evolução, a manutenção e a qualidade do software. Examinaram-se 39 repositórios de código aberto utilizando a ferramenta de análise estática DesigniteJava. Entre os *smells* identificados, destaca-se o *Unutilized Abstraction* (uso de abstrações desnecessárias). A análise revelou que a degradação da qualidade interna não resulta diretamente do tamanho do software, mas de decisões arquiteturais inadequadas. Conclui-se que o uso de ferramentas de análise estática é essencial para a detecção sistemática de anomalias, auxiliando na refatoração preventiva e na preservação dos princípios SOLID.

Palavras-chave: Designite, Design Smell, Bad Smell, Java.

1 INTRODUÇÃO

A qualidade do código-fonte é essencial para a manutenção e evolução de sistemas de software. Nesse contexto, destacam-se os *bad smells*, que indicam problemas recorrentes de design ou implementação, afetando legibilidade, reutilização e extensibilidade. Segundo Fowler (1999), *bad smells* são indícios de estruturas que podem ser melhoradas, funcionando como sintomas de um design deficiente.

A literatura distingue *code smells* e *design smells*. Os primeiros são problemas em trechos específicos do código (e.g., *Long Method*), enquanto os segundos envolvem questões estruturais mais amplas, como *Large Class*, que concentra responsabilidades excessivas. Identificar esses problemas é fundamental para a saúde do software. Estratégias de detecção baseadas em métricas têm sido propostas (MARINESCU, 2004), permitindo identificar falhas e evitar o acúmulo de débito técnico (SURYANARAYANA, SAMARTHYAM e SHARMA, 2014). Com o crescimento dos sistemas, a análise manual torna-se inviável, impulsionando o uso de abordagens automatizadas. Sabir e Rasool (2025) propõem uma técnica baseada no uso de OWL (*Web Ontology Language*), utilizando engenharia reversa para converter código em UML e, após, em ontologias,

permitindo a detecção por meio de regras formais. Apesar de eficaz, o processo é complexo e demanda múltiplas ferramentas. Outra abordagem, proposta por Rasool e Arshad (2017), usa a ferramenta *Designite* para análise de projetos *C#*. Embora o estudo tenha sido conduzido com poucos casos, a ferramenta demonstra potencial de aplicação mais amplo e suporte para outras linguagens (e.g., Java).

Diante disso, este estudo tem como objetivo analisar a ocorrência e os impactos de *design smells* em projetos Java de código aberto, considerando também sua relação com princípios associados ao ODS 9, especialmente no que se refere à promoção de softwares mais robustos, sustentáveis e inovadores. Para a detecção automática, foi utilizada a ferramenta *DesigniteJava* (SHARMA; MISHRA; TIWARI, 2016) (DESIGNITE, 2026), que identifica padrões de má implementação e fornece métricas estruturais.

2 METODOLOGIA (MATERIAL E MÉTODOS)

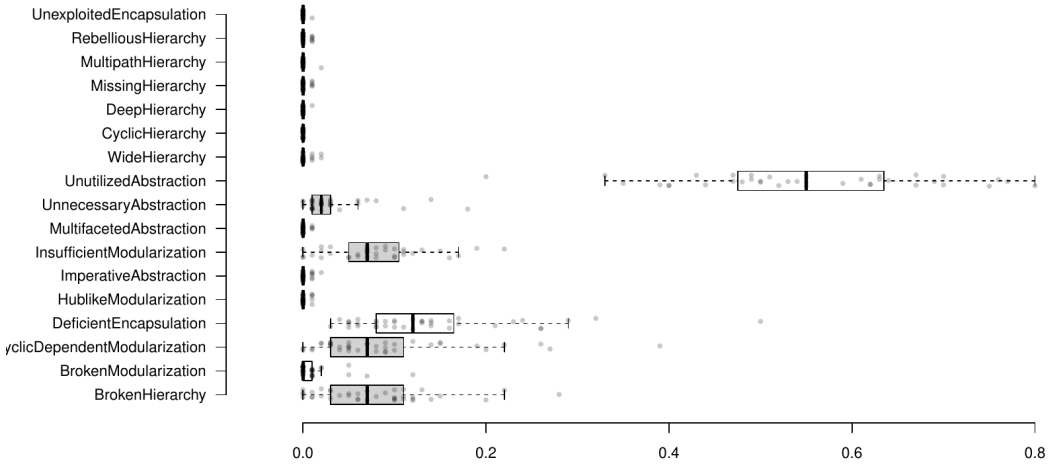
O processo metodológico deste estudo foi estruturado em três etapas principais: (*i*) seleção dos repositórios, (*ii*) execução da análise estática e (*iii*) tratamento e análise dos dados coletados. A seleção dos repositórios foi baseada em projetos que apresentam relevância no ecossistema Java. Foram escolhidos 39 projetos de organizações consolidadas, como Apache e Oracle, disponíveis na plataforma GitHub. O critério adotado consistiu na seleção dos repositórios dessas instituições com maior popularidade na plataforma, por potencialmente representarem aqueles mais utilizados pela comunidade e com maiores chances de apresentarem um desenvolvimento ativo e consistente. Esses projetos se destacam devido à sua utilização e popularidade e, por serem mantidos por grandes organizações, tendem a possuir uma construção sólida e a seguir padrões bem definidos em seu desenvolvimento. Contudo, devido à evolução e ao potencial aumento de complexidade nas implementações, esses sistemas podem sofrer com o aparecimento de *design smells*, tornando-os relevantes para a análise proposta.

Na segunda etapa, utilizou-se a ferramenta *DesigniteJava* para a detecção automática de anomalias. Conforme Sharma, Mishra e Tiwari (2016), essa ferramenta é útil na avaliação da qualidade do design por meio da identificação de *smells* de abstração, encapsulamento, modularidade e hierarquia. A execução consistiu em submeter cada repositório à análise estática na busca por violações de princípios fundamentais de design

orientado a objetos, como o Princípio de Substituição de Liskov (LSP), um dos princípios SOLID (RAMACHANDRAPPA, 2024), manifestadas em *smells* como o *Broken Hierarchy*.

Por fim, a análise dos dados coletados envolveu a extração de métricas dos relatórios gerados pela ferramenta, incluindo o total de anomalias de cada projeto e o número de ocorrências de cada tipo de *design smell*. Para comparar projetos de diferentes tamanhos, os dados foram normalizados em métricas percentuais por meio da relação entre a ocorrência de um *smell* específico e o total de problemas identificados.

Figura 1. Distribuição das categorias de design smells nos repositórios analisados (eixo X: frequência normalizada dos smells; eixo Y: tipo de smell; boxplots mostram a variação e a concentração dos dados, com pontos indicando ocorrências individuais).



Fonte: Autores, 2026.

3 RESULTADOS E DISCUSSÃO

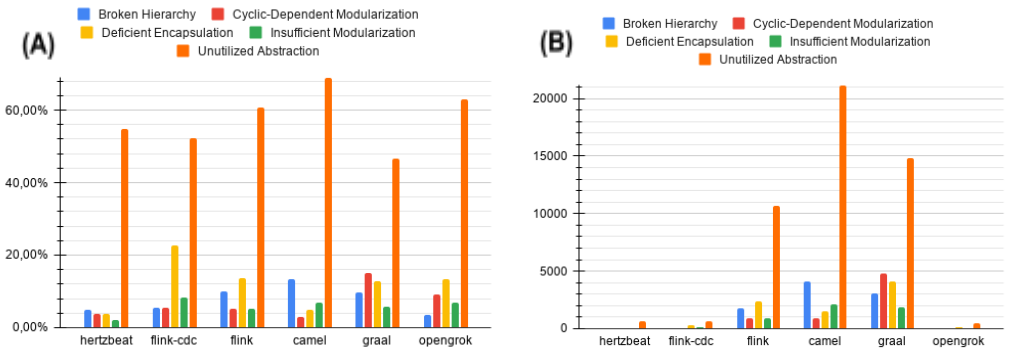
A análise dos 39 repositórios via *DesigniteJava* revelou a predominância de cinco categorias principais de *design smells*. Conforme a Figura 1, as categorias predominantes são *Unutilized Abstraction* (predominância média de ≈54% dentre os *design smells* dos repositórios), *Deficient Encapsulation* (≈11%), *Insufficient Modularization* (≈7%), *Broken Hierarchy* (≈7%) e *Cyclic-Dependent Modularization* (≈7%). Além disso, estas categorias estão presentes em ≈90%, ou mais, dos repositórios considerados. O *smell Unutilized Abstraction*, presente em todos repositórios, introduz complexidade estrutural desnecessária e, ao criar abstrações não utilizadas, pode ferir o Princípio da Segregação de Interface (ISP), pois infla a arquitetura com elementos que não cumprem um papel funcional efetivo, gerando uma interface maior que o necessário. A incidência de 100% do

Deficient Encapsulation eleva o acoplamento, ao permitir acessibilidade excessiva, e fere os princípios SOLID do Aberto/Fechado (OCP) e da Inversão de Dependência (DIP).

Ainda que os referidos *smells* estejam presentes em todos os projetos analisados, destacam-se outros três casos com alta ocorrência, o que torna relevante a investigação de padrões e similaridades que contribuam para a recorrência dessas anomalias e o aumento do débito técnico, comprometendo a evolução e a manutenção dos sistemas. No que tange à modularização, o *Cyclic-Dependent Modularization* aparece em 94,87% dos projetos e revela-se um desafio para a reusabilidade ao criar acoplamentos rígidos entre módulos. Complementarmente, o *Broken Hierarchy* foi identificado em 89,74% dos projetos, sendo uma anomalia que rompe o contrato semântico entre subclasses e superclasses, infringindo diretamente o Princípio de Substituição de Liskov (LSP) e introduzindo riscos imprevisíveis quanto ao emprego do polimorfismo. Ademais, o *smell Insufficient Modularization*, presente em 94,87% dos repositórios, se manifesta como um entrave à coesão do sistema, pois há a concentração de múltiplas responsabilidades em um único módulo, ferindo diretamente o Princípio da Responsabilidade Única (SRP).

Um dos achados mais significativos do estudo, ilustrado pela comparação entre os dados normalizados pelo número de *smells*, da Figura 2A, e os volumes absolutos, contidos na Figura 2B, é a ausência de uma correlação direta entre o tamanho do projeto (número de classes) e o percentual de anomalias. Como observado na Figura 2, projetos em diferentes escalas apresentam densidades similares de *smells*, indicando que a degradação da qualidade interna não é consequência direta do crescimento do software, mas sim um reflexo de decisões de design. A Figura 2B destaca os três repositórios com maior número de *smells*, o que pode elevar demais os custos de manutenção e evolução.

Figura 2. (A) Dados normalizados versus **(B)** Volumes absolutos de ocorrências.



Fonte: Autores, 2026.

4 CONCLUSÃO

Este estudo evidenciou que os *design smells* são desafios persistentes que violam princípios fundamentais e comprometem a evolução do software. Constatou-se que a degradação estrutural deriva de decisões inadequadas de design, e não do crescimento orgânico ou do tamanho do projeto. Adicionalmente, os resultados sugerem que a monitorização contínua e a aplicação rigorosa de padrões de projeto são essenciais para evitar que a complexidade técnica inviabilize manutenções futuras. Assim, reforça-se a importância de ferramentas de análise estática como o DesigniteJava na identificação sistemática de pontos de refatoração. O trabalho alinha-se ao ODS 9 (Indústria, Inovação e Infraestrutura) da ONU, uma vez que promover a qualidade e a longevidade do código-fonte é um pilar fundamental para a construção de infraestruturas tecnológicas resilientes, sustentáveis e inovadoras.

REFERÊNCIAS

DESIGNITE. **DesigniteJava**. 2026. Disponível em: https://www.designite-tools.com/products-dj. Acesso em: 20 mar. 2026.

FOWLER, Martin. ***Refactoring: Improving the Design of Existing Code***. Boston: Addison-Wesley, 1999.

MARINESCU, Radu. **Detection strategies: Metrics-based rules for detecting design flaws.** In: *Proceedings of the 20th IEEE Inter. Conference on Software Maintenance (ICSM)*. Chicago, IL, USA: IEEE, 2004. pp. 350–359. doi: 10.1109/ICSM.2004.1357820.

RAMACHANDRAPPA, Naveen C. **SOLID Design Principles in Software Engineering**. *International Journal of Computer Trends and Technology (IJCTT)*, vol. 72, no. 9, pp. 18-23, 2024, doi: 10.14445/22312803/IJCTT-V72I9P104.

RASOOL, G., ARSHAD, Z. **A Lightweight Approach for Detection of Code Smells**. *Arabian Journal for Science and Eng.,* v. 42, pp. 483–506, 2017. doi: 10.1007/s13369-016-2238-8.

SABIR, S.; RASOOL, G. **A Lightweight Approach for Detection of Software Design Smells**. In: *Proceedings of the 6th Inter. Conf. on Advancements in Computational Sciences* (ICACS), Lahore, Pakistan, 2025, pp. 1-4. doi: 10.1109/ICACS64902.2025.10937837.

SHARMA, Tushar; MISHRA, Pratibha; TIWARI, Rohit. Designite: **A Software Design Quality Assessment Tool**. In: *Proceedings of the 1st Inter. Workshop on Bringing Architectural Design Thinking into Developers' Daily Activities* (BRIDGE '16). New York, NY, USA: ACM, 2016. pp. 1–4, doi: 10.1145/2896935.2896938.

SURYANARAYANA, Girish; SAMARTHYAM, Ganesh; SHARMA, Tushar. ***Refactoring for software design smells: managing technical debt***. Waltham: Morgan Kaufmann, 2014.