Investigando a Percepção de Desenvolvedores de Software sobre a Adoção de LLMs na Refatoração de *Code Smells*

Javel Freitas[1], Guilherme Pereira[1], Lara Lima[1], Caio Sousa[1], Edivar Filho[1], José Cezar de Souza Filho[2], Carla Bezerra[1]

[1]Universidade Federal do Ceará (UFC) – Campus Quixadá

[2]Univ. Polytechnique Hauts-de-France – LAMIH, UMR CNRS 8201

{javelqueiroz, guilhermepereira, laragabrielly,

caio.rianbr, edivarcruz}@alu.ufc.br,

josecezar.juniordesouzafilho@uphf.fr, carlailane@ufc.br

Abstract. *This paper investigates developers' perceptions of refactoring code smells using Large Language Models (LLMs). Through a study with 48 Java developers, we analyzed (i) their opinions on the use of these tools, (ii) which factors affect the prioritization of the refactoring order, and (iii) their perceptions about the quality of the code generated for refactorings. Our results indicate the benefits of using LLMs in refactoring and in the software development processes, including improvement in quality attributes, productivity, learning, and support during development. However, possible short- and long-term maladaptive tendencies were also highlighted, such as using low-quality code, counterproductive decision-making, and dependence on LLMs for development activities.*

Resumo. *Este artigo investiga a percepção de desenvolvedores sobre a refatoração de code smells por meio de Large Language Models (LLMs). Através de um estudo com 48 desenvolvedores Java, analisamos (i) a opinião sobre o uso dessas ferramentas, (ii) quais fatores afetam a priorização da ordem de refatoração e (iii) a percepção sobre a qualidade do código gerado para as refatorações. Nossos resultados indicam os benefícios do uso de LLMs tanto na refatoração quanto no processo de desenvolvimento de software, incluindo melhoria em atributos de qualidade, produtividade, aprendizado e apoio durante o desenvolvimento. Porém, também foram evidenciadas possíveis tendências desadaptativas a curto e longo prazo, como o uso de códigos de baixa qualidade, a tomada de decisões contraproducentes e a dependência nos LLMs para as atividades de desenvolvimento.*

1. Introdução

A refatoração de código é uma prática essencial para a melhoria da qualidade de software, especialmente quando lidamos com *code smells*, indicativos de fragilidades no design que são capazes de dificultar a compreensão e evolução do código, como métodos longos, classes que assumem muitas responsabilidades, ou as que fazem mais uso de funções de outras classes que da própria [Fowler 2018]. *Code smells* podem não apresentar falhas funcionais diretas, no entanto, ainda podem impactar negativamente diversos atributos de qualidade de um software [Kruchten et al. 2012, Ahmed et al. 2015], tais como

o acoplamento, coesão, complexidade e herança do código [Martins et al. 2021]. Para mitigar esses efeitos, técnicas de refatoração podem contribuir para a remoção de *code smells*, desde que a refatoração seja adequadamente realizada, com aplicação criteriosa e no contexto apropriado [Lacerda et al. 2020].

Recentemente, os Grandes Modelos de Linguagem (*LLMs*, do inglês *Large Language Models*) têm revolucionado o desenvolvimento de software, oferecendo suporte automatizado para a geração, análise e otimização de código. Tais modelos também podem ser úteis para a melhoria da qualidade de código e refatoração de *code smells* [Chang et al. 2024]. No entanto, a efetividade dessas ferramentas depende de como os desenvolvedores interagem com elas, particularmente, no uso de *prompts* que orientam o modelo na direção desejada [AlOmar et al. 2024].

Diversos estudos têm abordado a capacidade dos *LLMs* em relação à refatoração de código, experimentando variações de *prompt* para o aumento de eficiência na refatoração [Zhang et al. 2024], explorando a capacidade de boas sugestões de refatoração após treinamento com refatorações de qualidade [Pomian et al. 2024], e também casos em que o *LLM* infere o uso de práticas de refatoração conhecidas mesmo sem menção direta [AlOmar et al. 2024]. Dado que este é um tópico de pesquisa emergente, novos estudos são necessários para avançar o entendimento sobre o uso de *LLMs* em diferentes atividades de desenvolvimento de software [Fan et al. 2023], incluindo investigações acerca da percepção dos desenvolvedores sobre a adoção de tais ferramentas para apoiar atividades de refatoração de código.

Este artigo tem como objetivo investigar a percepção de desenvolvedores de software sobre o uso de *LLMs* na refatoração de *code smells*. O estudo investiga, principalmente, a percepção sobre o uso de *LLMs* como suporte às atividades de desenvolvimento, os fatores que influenciam a priorização da refatoração de *code smells* e a percepção da qualidade do código refatorado. Para isso, busca-se responder: (i) a utilidade e limitações das *LLMs* no suporte ao desenvolvimento de software; (ii) os fatores que influenciam a decisão de priorização da refatoração de *code smells*; e, (iii) a percepção dos desenvolvedores sobre a qualidade do código após a refatoração assistida por *LLMs*. Para responder a essas questões, realizamos um estudo experimental com 48 desenvolvedores, aplicando refatorações em uma série de *code smells* presentes em um projeto Java adaptado de instâncias em projetos *open source*. Para refatorar foram utilizadas três *LLMs*: ChatGPT[1], GitHub Copilot[2] e Google Gemini[3]. Os desenvolvedores executaram cenários para refatoração de *code smells* no projeto e suas percepções sobre o processo de refatoração foram coletadas por meio de questionários.

2. Fundamentação Teórica

2.1. *Code Smells*

Os *code smells* são trechos do código-fonte que apresentam indícios de má qualidade estrutural, tornando o software mais difícil de compreender e manter. Eles sinalizam áreas que podem ser refatoradas para melhorar a qualidade do projeto

[1]https://openai.com/
[2]https://github.com/features/copilot
[3]https://gemini.google.com/

[Danphitsanuphan e Suwantada 2012]. Eles foram catalogados, sendo o de Martin Fowler o catálogo mais referenciado, descrevendo 22 tipos distintos de problemas recorrentes na estrutura do código-fonte e técnicas de refatoração que buscam resolvê-los [Fowler 2018]. Tais problemas estruturais variam de tamanho e complexidade. Por exemplo, o *code smell Feature Envy* trata da situação em que uma classe utiliza mais os métodos de outra classe que os seus próprios para realizar uma função.

Todos esses problemas no código necessitam de uma refatoração contínua, a qual é fundamental para manter um código limpo, legível e adaptável às mudanças. Segundo [Kerievsky 2005], essa prática permite evoluir o *design* do sistema de forma incremental, preservando sua saúde estrutural e facilitando a adição de novas funcionalidades, especialmente em contextos de desenvolvimento ágil. Esse processo sistemático torna o código mais legível, modular e eficiente, contribuindo para uma arquitetura mais robusta. Como resultado, a manutenção do software se torna mais simples e os custos de futuras modificações são reduzidos [dos Santos et al. 2019].

2.2. *LLMs*

LLMs são modelos de linguagem computacionais capazes de entender e gerar linguagem humana [Chang et al. 2024]. A maioria deles tem como base a arquitetura *Transformer* [Vaswani et al. 2017, Pan et al. 2024] e podem auxiliar ao gerar resumos, traduzir línguas, identificar padrões [Chang et al. 2024], gerar código e também refatorá-lo [Zhang et al. 2024]. Uma das possíveis formas de interação é por meio de *prompt*, que é um conjunto de instruções que ao serem fornecidas para um *LLM* tem a capacidade de customizar, melhorar ou refinar suas capacidades [Liu et al. 2023].

LLMs são amplamente utilizados no desenvolvimento de software [Li et al. 2023]. O *ChatGPT* é um modelo conversacional capaz de interpretar comandos em linguagem natural para sugerir, corrigir e otimizar código [OpenAI 2024]. O *GitHub Copilot* auxilia na codificação, sugerindo trechos de código e descrições de mudanças, com suporte a modelos como *Claude 3.5 Sonnet* e *GPT-4o* [GitHub 2024]. O *Google Gemini* é um modelo multimodal que integra e processa diferentes tipos de informação, incluindo texto, código, áudio, imagem e vídeo [Google 2023].

3. Trabalhos Relacionados

[Menolli et al. 2024] investigaram o uso do *ChatGPT* no ensino de refatoração, realizando um estudo com 23 estudantes. Os resultados indicam que a ferramenta facilitou o aprendizado. Diferente deste, nosso estudo analisa a percepção dos desenvolvedores sobre *LLMs* na refatoração, focando na qualidade do código gerado, enquanto o estudo dos autores aborda o ensino de refatoração.

[AlOmar et al. 2024] investigaram como desenvolvedores usaram o *ChatGPT* para refatoração, identificando 43 padrões de documentação de refatoração, categorizados em atributos de qualidade. O estudo mostrou que o *ChatGPT* foca mais nos atributos de qualidade interna. Em contraste, nosso estudo analisa a percepção dos desenvolvedores sobre a qualidade do código após refatoração por *LLMs* (*ChatGPT*, *Copilot* e *Gemini*) em código Java.

[Sergeyuk et al. 2024] reavaliaram modelos de legibilidade de código Java com foco no desenvolvedor, examinando como a legibilidade do código gerado por Inteligên-

cia Artificial é percebida por humanos. Enquanto o estudo investiga a legibilidade do código, o nosso estudo foca na percepção dos desenvolvedores sobre a qualidade do código após a refatoração assistida por *LLMs* em código Java.

[Cordeiro et al. 2024] analisaram como o StarCoder2[4] pode realizar refatorações automáticas em código Java *open source*, focando na redução de *code smells* e comparando seu desempenho com desenvolvedores humanos. A principal distinção em relação ao nosso estudo é que enquanto os autores avaliam a eficácia do *LLM* na remoção de *code smells*, nosso estudo investiga a percepção dos desenvolvedores sobre a qualidade do código após a refatoração assistida por *LLMs* considerando múltiplos modelos.

4. Metodologia

4.1. Objetivo e Questões de Pesquisa

Este artigo apresenta um estudo sobre a percepção dos desenvolvedores ao utilizar *LLMs* para refatoração de *code smells*. A pesquisa foi conduzida com desenvolvedores trabalhando em duplas, refatorando *code smells* utilizando um *LLM* e ferramentas de análise e testes de unidade para apoiar a refatoração. Além disso, foram aplicados questionários para obter suas percepções. As questões de pesquisa (QP_s) são detalhadas a seguir.

QP_1: Qual é a opinião dos desenvolvedores sobre a utilidade e limitações dos *LLMs* no suporte ao desenvolvimento de software? Esta questão investiga a percepção dos desenvolvedores, antes de refatorar os *code smells*, a respeito do uso de *LLMs* como ferramentas de apoio às atividades de desenvolvimento. Ao responder a **QP_1**, é possível entender como o uso dos *LLMs* afetou positiva ou negativamente o processo de desenvolvimento.

QP_2: Quais fatores influenciam a decisão dos desenvolvedores na priorização da refatoração de *code smells*? Nesta QP, busca-se entender as motivações por trás da escolha da ordem de refatorações seguida. Ao responder a **QP_2**, é possível identificar quais aspectos os desenvolvedores priorizam antes de considerar uma ordem de refatoração.

QP_3: Como os desenvolvedores avaliam a qualidade do código após a refatoração assistida por *LLMs*? O objetivo desta questão é analisar a percepção dos desenvolvedores sobre a alteração da qualidade de código após utilizarem *LLMs* como ferramenta de apoio à refatoração. Ao responder a **QP_3**, é possível identificar a percepção dos desenvolvedores sobre o código gerado e a contribuição do *LLM*.

4.2. Etapas do Estudo

A Figura 1 ilustra as etapas necessárias para a realização do estudo.

Etapa 1: Seleção dos *code smells*. Foi selecionado o *dataset* MLCQ [Madeyski e Lewowski 2020], que reúne *code smells* de projetos Java industriais, e utilizamos as ferramentas PMD[5] e IntelliJDeodorant[6] para identificar e adaptar os *code smells*, priorizando os de menor gravidade. Foi criado um projeto[7] incluindo cinco instâncias para cada *code smell*: *Data Class*, *Large Class*, *Long Method* e *Long Parameter*

[4]https://github.com/bigcode-project/starcoder2

[5]https://pmd.github.io/

[6]https://plugins.jetbrains.com/plugin/14016-intellijdeodorant

[7]https://anonymous.4open.science/r/WASHES-2025

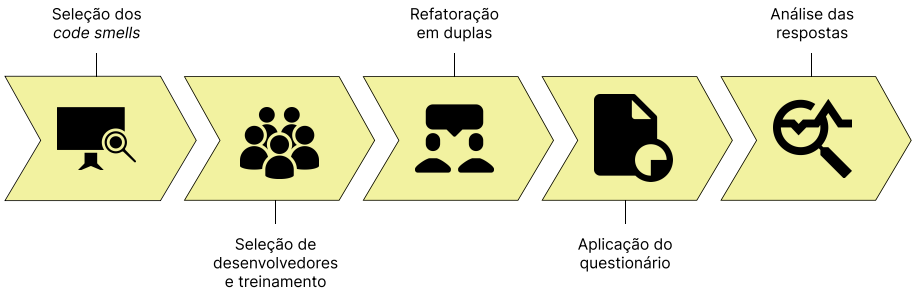

Figura 1. Etapas do estudo

List. O projeto possui 59 classes, distribuídas em 57 arquivos, possuindo 501 funções em 4.458 linhas. Além disso, foram desenvolvidos 100 testes de unidade com *JUnit* (228 *asserts*) para validar as refatorações (em que o comportamento observável não pode ser alterado).

Etapa 2: Seleção de desenvolvedores e treinamento. Foram selecionados 48 desenvolvedores Java, em que a maioria desenvolve a menos de cinco anos e usa *LLMs* no cotidiano. Foi ministrado um treinamento em *code smells*, refatoração e no uso de *LLMs* (*Google Gemini*, *OpenAI ChatGPT* e *GitHub Copilot*). O treinamento, teórico e prático, incluiu tutoriais sobre as ferramentas. Após um questionário de caracterização, cada dupla foi designada a um *LLM* para auxiliar nas refatorações.

Etapa 3: Refatoração em duplas. Os desenvolvedores foram agrupados em duplas para realizar as refatorações. Cada dupla foi designada a um *LLM* específica e usou ferramentas para análise estática de *code smells*. O trabalho foi feito em um repositório do *GitHub*, com cada dupla criando um *fork* e realizando refatorações, seguindo um processo detalhado para garantir a qualidade do código: (i) Fazer um *fork* do repositório; (ii) Executar as ferramentas de análise estática para identificar os *code smells*; (iii) Refatorar o código utilizando o *LLM* com o *prompt* específico para cada tipo de *code smell* (destacado abaixo); (iv) Verificar funcionalidade e executar testes para garantir que o comportamento não foi alterado; e, (v) Registrar as alterações feitas com *commits*, seguindo o padrão de mensagem específico.

> ***Prompt**: Fix the [**codeSmellName**] code smell using the [**refactoringTechniqueName**] refactoring technique.*

Etapa 4: Aplicar questionário para analisar o uso efetivo dos *LLMs* na refatoração. Foi aplicado um questionário para investigar a percepção dos desenvolvedores sobre a dificuldade de refatoração e percepção do uso de *LLMs*. O questionário consiste em perguntas abertas e fechadas a respeito das atividades exercidas no estudo. Ambos os questionários possuem um termo de consentimento informando os procedimentos do estudo e o cuidado com a privacidade dos desenvolvedores. Os dados dos questionários pré- e pós-estudo acerca da percepção de refatoração dos *code smells* estão disponíveis no *Zenodo*[8].

Etapa 5: Análise das respostas. As respostas dos questionários foram analisadas utilizando técnicas de *Grounded Theory* [Corbin e Strauss 2014], uma metodologia de

[8]https://doi.org/10.5281/zenodo.15079585

pesquisa qualitativa, com foco na codificação das respostas dos participantes por meio de *open coding* e *axial coding*. As percepções dos desenvolvedores foram então agrupadas em categorias com base nas suas respostas.

5. Resultados

Esta seção tem como objetivo responder às três questões de pesquisa do presente estudo por meio dos resultados obtidos a partir da análise da percepção dos desenvolvedores sobre o uso de *LLMs* na refatoração de *code smells*. Acompanhadas das categorias obtidas estão as citações dos participantes (P_s).

5.1. Opinião dos Desenvolvedores sobre a Utilidade e Limitações dos *LLMs* no Suporte ao Desenvolvimento de Software ($\mathbf{QP_1}$)

Ao investigar os dados coletados para a $\mathbf{QP}_1$, foram identificadas duas categorias sobre como os desenvolvedores percebem o uso de *LLMs* no desenvolvimento de software: (i) aspectos positivos e (ii) aspectos negativos. Estas são apresentadas na Figura 2.

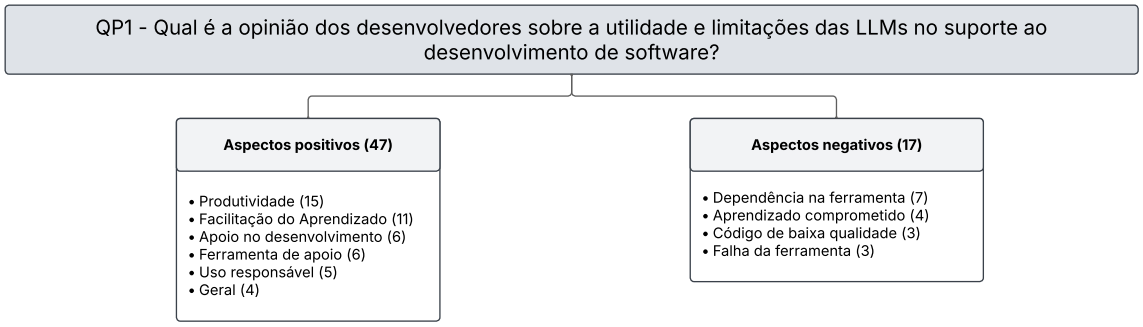

Figura 2. Opinião dos desenvolvedores sobre a utilidade e limitações dos *LLMs* no suporte ao desenvolvimento de software

Os resultados indicam que a maioria dos desenvolvedores (47 citações) percebe os *LLMs* como benéficas para o desenvolvimento de software. A maior parte das citações destacou a melhora na "produtividade" (15) e na "facilitação do aprendizado" (11).

P_5: "No desenvolvimento, LLMs têm sido benéficas ao acelerar a prototipagem, automatizar tarefas repetitivas, resolver problemas e compartilhar conhecimento técnico".
P_{45}: "Utilizo para tirar dúvidas sem oferecer amostra de código, para mostrar exemplos de uso, explicar saída de erros complexas, nomenclatura de funções e variáveis".

Além disso, o uso dessas ferramentas também foi considerado positivo para o apoio no desenvolvimento (6), como ferramenta de apoio (6), desde que seja usada de forma responsável (5), em que pode ser geralmente benéfica (4). Entretanto, parte dos desenvolvedores apontaram aspectos negativos (17) importantes, que podem limitar a experiência. A principal questão levantada está relacionada à dependência excessiva que essas ferramentas podem gerar (7 citações).

P_{47}: "Pode causar dependência criativa entre os programadores que as utilizam com frequência e sem limites".

Além disso, algumas respostas indicam que os desenvolvedores estão preocupados em ter seu aprendizado comprometido (4), que seu código sofra uma redução na qualidade (3) por conta da geração dos *LLMs* ou em eventuais falhas das mesmas (3).

Implicações da $\mathbf{QP}_1$. Foram identificados benefícios no uso de *LLMs* por parte dos desenvolvedores, sendo este uso para gerar código, apoiar no desenvolvimento, reduzir tarefas repetitivas e facilitar o entendimento. Também foi destacado o uso responsável e controlado das ferramentas para que sejam evitadas consequências prejudiciais a curto prazo, como piora na qualidade do código e experiência negativa devido a falhas, assim como depender da ferramenta ou ter o aprendizado comprometido a longo prazo.

5.2. Fatores que Influenciam a Decisão dos Desenvolvedores na Priorização da Refatoração de *Code Smells* ($\mathbf{QP_2}$)

A análise dos dados coletados para a $\mathbf{QP}_2$ identificou fatores que as duplas levaram em consideração para a priorização da refatoração de *code smells*: (i) organização interna, (ii) influência externa e (iii) motivação não explícita. Esses fatores são apresentados e detalhados na Figura 3.

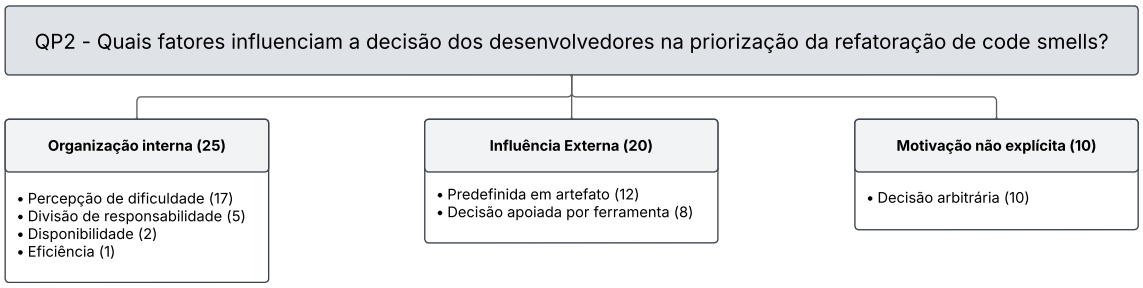

Figura 3. Fatores que influenciam a decisão dos desenvolvedores na priorização da refatoração de *code smells*

A categoria "organização interna" foi a mais citada pelos participantes (25 citações). Ela leva em consideração a tomada de decisão com base na percepção individual e na experiência da dupla. A subcategoria "percepção de dificuldade" (17) foi um dos critérios mais influentes, em que os participantes afirmaram preferir começar pelas refatorações mais simples ou evitar as complexas.

P_{34}: "Percepção de dificuldade, começando pelo aparentemente mais simples".

Além disso, a "divisão de responsabilidade" (5) entre os membros da dupla teve relevância, em que foram levadas em consideração a preferência entre as duplas. A disponibilidade dos desenvolvedores (2) e a eficiência (1 citação) também foram fatores mencionados, indicando que a escolha da ordem das refatorações pode considerar a carga de trabalho e a busca por um fluxo de desenvolvimento mais otimizado.

P_{12}: "Cada integrante escolheu dois tipos de smells para refatorar e realizou de acordo com sua disponibilidade".

Outra categoria levantada foi a "influência externa" (20), simbolizando o uso de meios externos à organização da dupla para definir a ordem das refatorações. Utilizar uma ordem "predefinida em artefato" (12), como documentação, ou tomar a "decisão apoiada por ferramenta" (8), como no resultado da análise do *PMD,* foram alternativas que as duplas utilizaram. As respostas dos participantes revelam que a priorização da refatoração pode ser guiada por informações externas que ajudaram a estruturar o processo.

P_{27}: "Achamos melhor seguir a sequência listada no README do repositório".
P_{25}: "Seguimos a ordem do documento JSON gerado pelo PMD".

Embora a sequência de refatoração tenha seguido diferentes critérios, 10 citações indicaram que a escolha da ordem de refatoração não teve motivações explícitas e foi feita de forma arbitrária. Isto indica que algumas refatorações podem ser realizadas sem uma motivação clara, sendo apenas baseadas na intuição dos participantes.

P_{16}: "A ordem dos demais não teve uma lógica, foi aleatória".

Implicações da $\mathbf{QP}_2$. A priorização da refatoração de *code smells* não segue um padrão único, podendo ser influenciada por fatores internos da dupla, externos ou pela intuição dos desenvolvedores. É possível que o uso de ferramentas automatizadas e documentação contribua para uma abordagem mais sistemática, enquanto a experiência da dupla e a percepção de dificuldade possam afetar diretamente a execução das refatorações. No entanto, a dependência excessiva dos *LLMs* e a adoção de decisões arbitrárias podem comprometer a qualidade do código, reforçando a necessidade de um planejamento cuidadoso e uma abordagem equilibrada entre automação e discernimento humano. Geralmente os desenvolvedores fazem pedidos genéricos, enquanto o *LLM* inclui intenção ao aplicar uma refatoração [AlOmar et al. 2024]. Conhecer as diferentes formas de priorizar a ordem de refatorações pode apoiar os desenvolvedores a incluir intenção ao conversar com os *LLMs*.

5.3. Como os Desenvolvedores Avaliam a Qualidade do Código Após a Refatoração Assistida por *LLM* ($\mathbf{QP_3}$)

A análise qualitativa dos questionários sintetizou dados que respondem à $\mathbf{QP}_3$. A Figura 4 ilustra as categorias levantadas: (i) melhoria em atributos de qualidade; (ii) melhoria na qualidade; (iii) melhoria no processo de desenvolvimento; e (iv) impacto negativo.

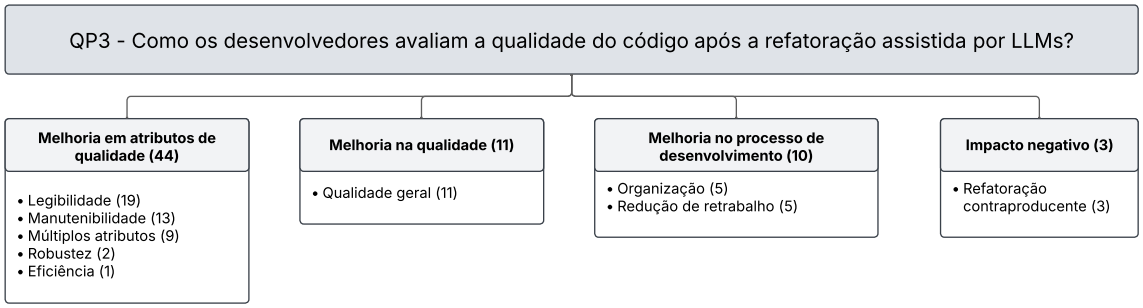

Figura 4. Como os desenvolvedores avaliam a qualidade do código após a refatoração assistida por *LLM*

A categoria "melhoria em atributos de qualidade" foi a mais citada (44 citações). A maior parte dos desenvolvedores comentou sobre o aprimoramento na "Legibilidade" (19 citações) após a refatoração assistida por *LLMs*, destacando a maior facilidade para ler e entender o código.

P_8: "O maior impacto foi em legibilidade, o código foi ficando mais claro".

Outra subcategoria amplamente citada foi a "Manutenibilidade". Contendo 13 menções, esta aborda sobre a maior facilidade para manter o projeto e realizar modificações, possibilitando menos complicações ao longo prazo.

P_3: "Reduz o tempo necessário para futuras alterações e diminui a probabilidade de introduzir novos bugs".

Houve menções sobre "múltiplos atributos" (9) de qualidade, em que os participantes perceberam melhorias em mais de um atributo em relação ao código gerado. Poucas citações foram levantadas sobre "Robustez" (2) e "Eficiência" (1).

P_{49}: "Após a remoção de code smells a melhoria da legibilidade e testabilidade do código foram visíveis".

Alguns participantes notaram melhorias de qualidade, mas não explicitaram um atributo específico. Para isso, a categoria "melhoria na qualidade" (11) foi introduzida. Todas as citações desta pertencem à subcategoria "qualidade geral", representando a percepção positiva geral sobre a qualidade do código gerado.

P_{43}: "Melhora na qualidade do código".

Fatores relativos à "melhoria no processo de desenvolvimento" (10) também foram citados, divididos igualmente entre as subcategorias "organização" e "redução de retrabalho" (5 citações para cada). Os participantes avaliaram como positiva a influência do uso de *LLMs* para reduzir tarefas repetitivas e apoiar na organização do código.

P_{14}: "A remoção impacta positivamente na qualidade do software, visto que esta prática evita problemas mais profundos no código, além de diminuir as chances de uma possível refatoração ou retrabalho futuro".

Por outro lado, foram observadas opiniões negativas sobre o uso das ferramentas para refatoração. A categoria "impacto negativo" (3) obteve três citações sobre "refatoração contraproducente", em que os participantes mencionaram que não refatorar com *LLMs* seria mais benéfico, devido à redução na qualidade do código e dificuldade de uso.

P_{21}: "Pelo trabalho que deu, achei que era melhor ter ficado com mal cheiro mesmo".

A análise dos dados coletados para a $\mathbf{QP}_3$ indicou que, em geral, os desenvolvedores consideram a refatoração assistida por *LLMs* benéfica, especialmente em termos de legibilidade, manutenibilidade e robustez. As melhorias incluem uma organização mais eficiente do código e a redução do esforço em modificações futuras. No entanto, algumas opiniões divergentes surgiram com alguns desenvolvedores considerando o esforço de refatoração não justificável pelos benefícios obtidos. Isso sugere que, embora a refatoração traga impactos positivos, a percepção sobre seu custo-benefício pode variar conforme a complexidade do processo.

Implicações da $\mathbf{QP}_3$. Grande parte dos desenvolvedores perceberam melhorias significativas na qualidade do código após a refatoração, especialmente no quesito de legibilidade e manutenibilidade. Entretanto, alguns desenvolvedores consideraram a refatoração um processo exaustivo, cujo esforço não justificava os benefícios obtidos. Isso sugere que o uso dos *LLMs* deve ser complementado por documentações e diretrizes mais eficientes e adaptáveis ao contexto de cada projeto, garantindo qualidade e otimização.

6. Limitações

São fatores que podem limitar a replicabilidade deste estudo: (i) o uso de apenas um *dataset* para adaptação de um projeto, havendo a possibilidade de experimentar com outros *datasets* e projetos; (ii) a variabilidade nas respostas dos *LLMs*, principalmente, após o primeiro *prompt*; (iii) falsos positivos e negativos das ferramentas de análise a respeito da detecção de *code smells*, que podem ser mitigados pelo uso de outras ferramentas em conjunto; e, (iv) o uso de três *LLMs* específicas (*ChatGPT*, *Copilot*, *Gemini*), podendo limitar a generalização para outros contextos.

7. Considerações Finais

Neste artigo, foram investigadas as percepções de desenvolvedores sobre refatoração de *code smells* por meio de *LLMs*. Para isso, foi adaptado um projeto com 20 instâncias de *code smells* que foi refatorado por duplas utilizando os *LLMs*: *ChatGPT*, *Copilot* e *Gemini*. Os resultados obtidos mostram que os desenvolvedores percebem o uso de *LLMs* como positivo para produtividade e como suporte durante o desenvolvimento de software, desde que usados com cuidado para evitar depender da ferramenta ou utilizar códigos gerados com baixa qualidade. A ordem de refatoração seguida pelas duplas teve influência de meios externos (*e.g.*, artefatos do projeto), internos (*e.g.*, percepção de dificuldade), e da intuição das duplas. Em relação ao código gerado, os desenvolvedores avaliaram que as *LLMs* melhoraram a qualidade geral, atributos de qualidade específicos, organização do código e redução de retrabalho. Porém, eles também consideraram que em certas situações deve-se ponderar se a refatoração é necessária, ou se teria efeitos contraproducentes.

Como trabalhos futuros, pretende-se: (i) replicar o estudo para outros *code smells* e projetos; (ii) avaliar as percepções dos desenvolvedores considerando refatorações de maior complexidade; (iii) comparar a percepção entre os membros das duplas; e, (iv) identificar estratégias para mitigar os impactos negativos na percepção.

Referências

Ahmed, I., Mannan, U. A., Gopinath, R., and Jensen, C. (2015). An empirical study of design degradation: How software projects get worse over time. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10.

AlOmar, E. A., Venkatakrishnan, A., Mkaouer, M. W., Newman, C., and Ouni, A. (2024). How to refactor this code? an exploratory study on developer-chatgpt refactoring conversations. In *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, page 202–206, New York, NY, USA. Association for Computing Machinery.

Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P. S., Yang, Q., and Xie, X. (2024). A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3).

Corbin, J. and Strauss, A. (2014). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, USA, 4th. edition.

Cordeiro, J., Noei, S., and Zou, Y. (2024). An empirical study on the code refactoring capability of large language models.

Danphitsanuphan, P. and Suwantada, T. (2012). Code smell detecting tool and code smell-structure bug relationship. In *2012 Spring congress on engineering and technology*, pages 1–5. IEEE.

dos Santos, H. M., Durelli, V. H., Souza, M., Figueiredo, E., da Silva, L. T., and Durelli, R. S. (2019). Cleangame: Gamifying the identification of code smells. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 437–446.

Fan, A., Gokkaya, B., Harman, M., Lyubarskiy, M., Sengupta, S., Yoo, S., and Zhang, J. M. (2023). Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE)*, pages 31–53.

Fowler, M. (2018). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

GitHub (2024). What is github copilot? Disponível em: https://docs.github.com/en/copilot/about-github-copilot/what-is-github-copilot. Acesso em: 11 mar. 2025.

Google (2023). Introducing gemini: our largest and most capable ai model. Disponível em: https://blog.google/technology/ai/google-gemini-ai/. Acesso em: 11 mar. 2025.

Kerievsky, J. (2005). *Refactoring to Patterns*. Addison-Wesley, Boston.

Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software*, 29(6):18–21.

Lacerda, G., Petrillo, F., Pimenta, M., and Guéhéneuc, Y. G. (2020). Code smells and refactoring: A tertiary systematic review of challenges and observations. *Journal of Systems and Software*, 167:110610.

Li, Z., Wang, C., Liu, Z., Wang, H., Chen, D., Wang, S., and Gao, C. (2023). Cctest: Testing and repairing code completion systems. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*. IEEE/ACM.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2023). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9).

Madeyski, L. and Lewowski, T. (2020). Mlcq: Industry-relevant code smell data set. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering*, EASE '20, page 342–347, New York, NY, USA. Association for Computing Machinery.

Martins, J., Bezerra, C., Uchôa, A., and Garcia, A. (2021). How do code smell co-occurrences removal impact internal quality attributes? a developers' perspective. In *Proceedings of the XXXV Brazilian Symposium on Software Engineering*, SBES '21, page 54–63, New York, NY, USA. Association for Computing Machinery.

Menolli, A., Strik, B., and Rodrigues, L. (2024). Teaching refactoring to improve code quality with chatgpt: An experience report in undergraduate lessons. In *Proceedings of*

the XXIII Brazilian Symposium on Software Quality, SBQS '24, page 563–574, New York, NY, USA. Association for Computing Machinery.

OpenAI (2024). Openai api documentation. Disponível em: https://platform.openai.com/docs/. Acesso em: 11 mar. 2025.

Pan, S., Luo, L., Wang, Y., Chen, C., Wang, J., and Wu, X. (2024). Unifying large language models and knowledge graphs: A roadmap. *IEEE Transactions on Knowledge and Data Engineering*, 36(7):3580–3599.

Pomian, D., Bellur, A., Dilhara, M., Kurbatova, Z., Bogomolov, E., Bryksin, T., and Dig, D. (2024). Next-generation refactoring: Combining llm insights and ide capabilities for extract method. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 275–287.

Sergeyuk, A., Lvova, O., Titov, S., Serova, A., Bagirov, F., Kirillova, E., and Bryksin, T. (2024). Reassessing java code readability models with a human-centered approach. In *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, ICPC '24, page 225–235, New York, NY, USA. Association for Computing Machinery.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Zhang, B., Liang, P., Feng, Q., Fu, Y., and Li, Z. (2024). Copilot-in-the-loop: Fixing code smells in copilot-generated python code using copilot. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, ASE '24, page 2230–2234, New York, NY, USA. Association for Computing Machinery.