Análise e Caracterização de Ferramentas Automatizadas de Refatoração

Matheus Machado de O. Andrade, Samara Martins Ferreira, Cleiton Silva Tavares, Leonardo Cardoso, Laerte Xavier, Lucila Ishitani

Bacharelado em Engenharia de Software – PUC Minas
Belo Horizonte – MG – Brasil

{andradematheusse,mferreira.samara}@gmail.com,
{cleitontavares,leonardocardoso,laertexavier,lucila}@pucminas.br

Abstract. *Automated refactoring is an increasingly common practice in the software development process. However, the effectiveness of using artificial intelligence for this purpose is still unclear. Therefore, this study evaluates the GPT-4.5 and Claude 3.7 Sonnet for automated refactorings. The research focuses on the application of these tools to the jparse system, aiming to measure their efficiency and applicability. The evaluation involved quantitative and qualitative analyses of 816 refactorings, generating eight criteria for evaluating these tools in real-world scenarios. The results showed that the tools perform better when applied to smaller scopes and when the type of refactoring to be performed is specifically guided.*

Resumo. *Refatoração automatizada é uma prática cada vez mais comum no processo de desenvolvimento de software. Porém, ainda não está claro a eficácia do uso de inteligência artificial para esse propósito. Sendo assim, este estudo avalia o GPT-4.5 e o Claude 3.7 Sonnet para refatorações automatizadas. A pesquisa concentra-se na aplicação dessas ferramentas ao sistema jparse, visando medir sua eficiência e aplicabilidades. A avaliação envolveu análises quantitativas e qualitativas de 816 refatorações, gerando oito critérios para avaliação dessas ferramentas em cenários reais. Os resultados mostraram que as ferramentas possuem um melhor comportamento quando aplicadas a escopos menores e direcionamento da refatoração a ser feita.*

1. Introdução

Refatoração é o processo de modificar a estrutura interna de um software sem alterar seu comportamento externo, visando melhorar sua legibilidade, manutenibilidade e *design* interno [Fowler 2018]. Técnicas como extração de métodos e extração de classes são amplamente utilizadas nesse contexto por facilitarem a modularização e a redução da complexidade [Valente 2020]. Essas atividades ocorrem durante a evolução do *software* e, quando feitas manualmente, são complexas, demoradas e suscetíveis a erros [Mealy and Strooper 2006]. O uso de ferramentas automatizadas de refatoração auxilia os desenvolvedores, reduzindo esses problemas do processo manual [Eilertsen and Murphy 2021b].

Com o avanço de modelos de linguagem, surgiram ferramentas que aplicam refatorações com o suporte de inteligência artificial, como o GPT-4.5 e o Claude 3.7 Sonnet. Estudos como os de Almogahed et al. (2023) e Ivers et al. (2022) demonstram o uso de *Large Language Models* (LLMs) em atividades de engenharia de *software*, incluindo refatoração automatizada. Desse modo, a formulação adequada dos *prompts* tem papel central nesse processo. Estratégias como *Persona Pattern*, *Template Pattern* e o uso de *embeddings* são citadas na literatura como formas de estruturar comandos mais claros e efetivos para os modelos [White et al. 2023]

Embora tragam benefícios, estudos anteriores apontam limitações nas ferramentas de refatoração, como problemas de suporte e usabilidade [Mealy and Strooper 2006], e a falta de previsibilidade que gera insegurança nos desenvolvedores [Eilertsen and Murphy 2021b] e dificuldades com grandes bases de código [Ivers et al. 2022]. Contudo, esses trabalhos não fizeram uma comparação sistemática das ferramentas de refatoração com IA, deixando um espaço para investigação. Avaliar essas ferramentas é essencial para garantir suporte adequado ao ambiente de desenvolvimento [Mealy and Strooper 2006]. Compará-las permite orientar seu uso, demonstrar eficiência e avaliar confiabilidade na refatoração [Eilertsen and Murphy 2021b]. Além disso, confrontar suas capacidades auxilia na definição de propósitos e na identificação da necessidade de métodos complementares.

O objetivo geral deste estudo é analisar o comportamento de ferramentas de refatoração automatizada baseadas em inteligência artificial, considerando seu impacto nas métricas estruturais de código e em aspectos qualitativos das refatorações, e propor um conjunto de critérios para avaliar essas refatorações a partir da comparação entre GPT-4.5 e Claude 3.7 Sonnet. Para avaliar os impactos das refatorações, este estudo utiliza um conjunto consolidado de métricas de projeto orientado a objetos amplamente discutidas na literatura, como aquelas propostas por Chidamber e Kemerer (1994), que abrangem aspectos de complexidade, coesão e acoplamento em nível de classe. Tais métricas têm sido empregadas extensivamente em pesquisas empíricas como indicadores quantitativos da qualidade estrutural de sistemas orientados a objetos, contribuindo para a análise objetiva dos efeitos de técnicas de refatoração estruturais, como Extração de Classe e Extração de Método [Chidamber and Kemerer 1994].

Os experimentos envolveram 816 refatorações no sistema jparse, sendo 327 executadas pelo GPT-4.5 e 489 pelo Claude 3.7 Sonnet. Como resultado foram propostos 8 critérios para a análise da eficácia das refatorações, 4 qualitativos e 4 quantitativos.

A Seção 2 discute trabalhos relacionados, destacando avaliações e limitações dessas ferramentas. A Seção 3 detalha a metodologia, incluindo seleção das ferramentas, *dataset* e critérios de avaliação. As seções 4 e 5, respectivamente, apresentam e discutem os resultados obtidos, comparando o desempenho conforme os critérios estabelecidos. Por fim, a Seção 7 traz as conclusões, ressaltando os principais achados e sugestões para trabalhos futuros.

2. Trabalhos Relacionados

Almogahed et al. (2023) investigaram os impactos de refatorações, como adição de parâmetros e extração de classes, na qualidade do *software*, mostrando que o efeito varia conforme o tamanho do sistema e as ferramentas usadas. O estudo destaca a in-

fluência de ferramentas automatizadas como JDeodorant e Eclipse Refactor na eficácia da refatoração. Embora compare refatorações em diferentes ferramentas, esse estudo não oferece recomendações para realizá-las.

Por sua vez, Ivers et al. (2022) entrevistaram 107 desenvolvedores da indústria para investigar o uso de ferramentas de refatoração em larga escala. O estudo revelou que, apesar do interesse em refatorar sistemas monolíticos para melhorar a manutenibilidade e migrar para arquiteturas modernas, as ferramentas atuais não suportam essas refatorações maiores. Diferentemente deste estudo, não exploraram comparações entre soluções eficazes para esse contexto.

Além disso, na abordagem de Eilertsen e Murphy (2021) foi proposto um modelo de refatoração em etapas, permitindo que desenvolvedores apliquem transformações graduais e tenham maior controle, solucionando problemas de usabilidade das ferramentas tradicionais, como alterações atômicas prematuras. Contudo, o estudo limita-se ao paradigma tradicional, sem considerar ferramentas com inteligência artificial.

O estudo de Mealy e Strooper (2006) avaliaram seis ferramentas Java usando o método DESMET para identificar limitações em critérios de refatoração e usabilidade. O estudo mostrou deficiências significativas no suporte ao processo e na usabilidade, evidenciando que nenhuma ferramenta atendeu plenamente às necessidades dos desenvolvedores. Diferentemente deste estudo, Mealy e Strooper avaliaram ferramentas de forma individual, sem o objetivo de compará-las quanto à eficácia

Em síntese, os trabalhos revisados abordam diferentes aspectos das ferramentas de refatoração, como usabilidade, limitações e critérios de avaliação. Este estudo busca complementar essas contribuições ao analisar comparativamente ferramentas mais recentes que incorporam inteligência artificial e técnicas de engenharia de *prompt*, com foco na eficiência no processo de refatoração.

3. Materiais e Métodos

Este experimento investiga o uso das ferramentas de inteligência artificial GPT 4.5 e Claude 3.7 Sonnet na refatoração de código, com foco na extração de métodos e classes, utilizando o *dataset* Qualitas Corpus como base empírica [Tempero et al. 2010]. A Figura 1 apresenta as etapas do fluxo metodológico adotado.

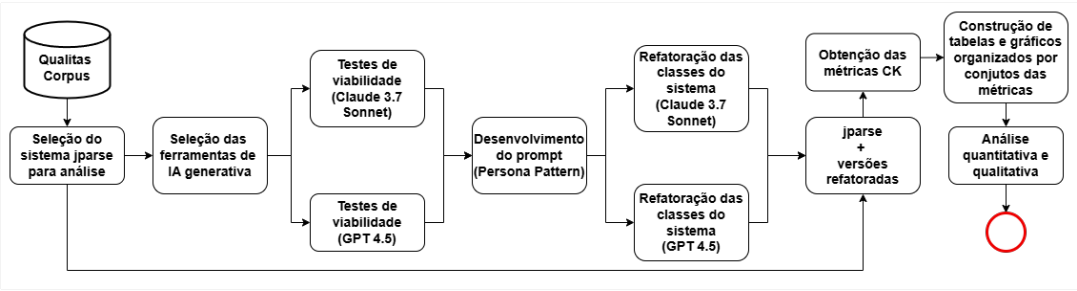


Figura 1. Fluxograma metodológico

Inicialmente foi realizado a escolha do *dataset*. O Qualitas Corpus foi escolhido por ser composto por 111 projetos Java com mais de 18 milhões de linhas de código [Terra et al. 2013], e amplamente reconhecido em pesquisas de engenharia de software

pela presença de *code smells*, tornando-se adequado para análises de refatoração automatizada. Logo em seguida foi realizado a escolha do sistema a ser avaliado. A escolha do sistema *jparse* fundamenta-se em suas métricas elevadas de complexidade e acoplamento, como MLOC (*Method Lines of Code*) de 23.44, WMC (*Weighted Methods per Class*) de 87.84 e VG (*McCabe Cyclomatic Complexity*) de 7.28, além de baixa coesão, com LCOM (*Lack of Cohesion in Methods*) de 0.26, o que evidencia sua propensão a melhorias estruturais por meio da extração de componentes.

A análise foi conduzida com o uso da IDE IntelliJ IDEA, que integra as ferramentas de IA avaliadas. O Gemini foi descartado devido a restrições de uso, mantendo-se o foco no GPT 4.5 e no Claude 3.7 Sonnet. Para aferir os impactos das refatorações, adotaram-se as métricas CK, de Chidamber e Kemerer (1994), obtidas por meio de uma aplicação de análise estática de código. Essas métricas forneceram subsídios objetivos para comparar a eficácia das ferramentas, tanto em termos quantitativos quanto qualitativos, oferecendo subsídios práticos para estudos futuros e para o avanço de técnicas de refatoração assistidas por IA.

A engenharia de *prompt* deste trabalho foi projetada com base em princípios de personalização e controle contextual descritos por White et al. (2023). O *prompt* segue o padrão *Persona Pattern*, atribuindo à IA o papel de especialista em refatoração de código, e incorpora elementos do *Template Pattern* para garantir consistência estrutural nas respostas. Essa abordagem padroniza e restringe as refatorações às técnicas de Extração de Métodos e Extração de Classes, conforme Fowler (2018) e Valente (2023), assegurando replicabilidade e foco ao limitar o escopo das transformações.

Além disso, o modelo de *prompt* explora o conceito de *embeddings*, conforme destacado pela AWS (2024), ao fornecer ao modelo um contexto rico e semanticamente denso por meio de instruções detalhadas. Também foi solicitado que cada resposta apresentasse um resumo das alterações, especificando o total de refatorações realizadas e sua divisão por tipo, visando futuras análises. A Figura 2 apresenta o *prompt* utilizado nesta etapa.

```
Você é uma IA especialista em refatoração de código, com foco exclusivo em Extração de Classe e Extração de Métodos.
Seu objetivo é aprimorar a legibilidade, organização e manutenibilidade do código sem alterar seu comportamento.

Baseie-se nos seguintes princípios e referências:
- Martin Fowler (2018) - Refactoring: Improving the Design of Existing Code
- Marco Tulio - Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade
  (https://engsoftmoderna.info/cap9.html)

Diretrizes:
- Remova todos os comentários da classe a ser refatorada antes de começar a refatoração.
- Analise a classe fornecida e identifique oportunidades para refatorações de Extração de Classes e Extração de Métodos.
  Buscando remover as duplicações de código durante a refatoração.
- Não execute nenhuma refatoração fora desse escopo.
- Explique suas decisões com base nos conceitos dos livros mencionados.
- No código refatorado, adicione um comentário iniciando com "TRECHO REFATORADO" em cada local modificado, para facilitar
  a identificação.
- Os comentários no código devem estar em português.

Modo de Operação:
- Utilize esse mesmo padrão para TODAS as classes que forem refatoradas.
- Não faça nada ainda. Aguarde meu comando para iniciar.

Estrutura da resposta ao refatorar uma classe (passos de 1 a 4):

1) Oportunidades de refatoração encontradas – Liste os trechos problemáticos que justificam as refatorações.

2) Classe refatorada com os devidos comentários – Apresente o código atualizado com comentários em português, incluindo
   // TRECHO REFATORADO nos locais alterados.

3) Justificativa das refatorações – Explique por que cada refatoração foi feita e como ela melhora o design do código.

4) Resumo das alterações, incluindo:
       - Quantidade total de refatorações realizadas.
       - Divisão por tipo (quantas foram Extração de Método e quantas foram Extração de Classe).
```

Figura 2. *Prompt* Desenvolvido

As refatorações foram realizadas individualmente em cada classe do sistema *jparse* (exceto testes), utilizando a integração nativa da IDE com os assistentes de IA Claude 3.7 Sonnet e GPT 4.5. Em ambos os casos, um mesmo *prompt* foi aplicado com consistência, anexando o arquivo da classe correspondente a cada requisição para garantir precisão e uniformidade no processo. As respostas retornadas incluíram o código refatorado e dados sobre os tipos e quantidades de refatorações, que foram armazenados como arquivos '.java' para posterior análise. Essa sistematização permitiu a construção dos dados, que organiza e classifica as refatorações por tipo e ferramenta, facilitando comparações entre os resultados obtidos. Estes dados estão disponíveis no repositório do trabalho[1].

Para avaliar os impactos quantitativos das refatorações, utilizou-se a ferramenta CK Metrics em três momentos: antes das alterações e após a refatoração por cada ferramenta. As métricas selecionadas incluíram CBO, LCOM, TCC, LCC, RFC, NOM, MLOC, VG, e WMC para análise de extração de classes e de métodos — métricas diretamente ligadas à coesão, complexidade e acoplamento. A partir dessas informações, foram conduzidas análises comparativas detalhadas, com visualizações como *boxplots* e *violinplots*, além de testes estatísticos que asseguraram a significância dos resultados, oferecendo uma base robusta para interpretar a eficácia das refatorações geradas por cada IA.

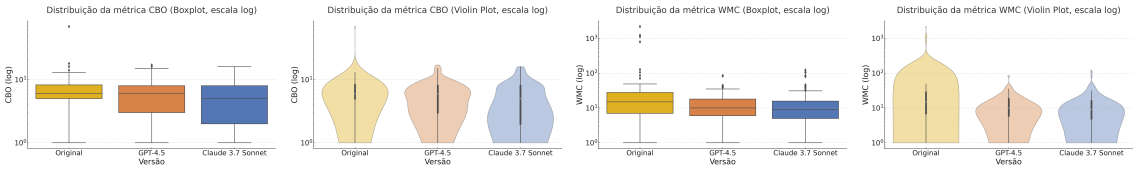
4. Resultados Quantitativos

Esta seção apresenta os resultados quantitativos das refatorações automatizadas realizadas pelas ferramentas GPT 4.5 e Claude 3.7 Sonnet no sistema *jparse*, do Qualitas Corpus. Ao todo, foram executadas 816 refatorações, das quais 327 pelo GPT 4.5 — sendo 277 de extração de métodos $(84,71\%)$ e 50 de extração de classes $(15,29\%)$ — e 489 pelo Claude 3.7 Sonnet, com 408 extrações de métodos $(83,43\%)$ e 81 de classes $(16,57\%)$. Esses dados evidenciam uma atuação mais intensa da ferramenta Claude, com predominância de extrações de métodos em ambas as abordagens.

Com o objetivo de mensurar o impacto estrutural das refatorações realizadas pelas ferramentas GPT 4.5 e Claude 3.7 Sonnet, foi conduzida uma análise quantitativa com base nas métricas Chidamber & Kemerer (CK). Na versão original da métrica *Coupling Between Objects* (CBO), observou-se mediana de 6, média de $7,68$ e valor máximo de 69. Após refatoração, o GPT 4.5 manteve mediana igual, reduzindo a média para $5,61$ e o valor máximo para 17; já o Claude 3.7 Sonnet baixou a mediana a 4, a média a $4,85$ e o valor máximo a 16 (Fig 3(a) e 3(b)). Os *boxplots* evidenciam a redução de acoplamento, enquanto os *violin plots* ressaltam a densidade concentrada em faixas mais baixas, destacando maior densidade de valores na base inferior para Claude 3.7 Sonnet e concentração abaixo da média original no GPT 4.5.

A métrica *Weighted Methods per Class* (WMC) apresentou mediana de 15, média de 49.38 e máximo de 1217, na versão original (Figs 3(c) e 3(d)). O GPT 4.5 reduziu a mediana para 9, a média para $13,73$ e o máximo para 85; o Claude 3.7 Sonnet obteve mediana de 8, média de $14,03$ e máximo de 123. Os gráficos mostram essas alterações por meio de caixas mais compactas e concentrações visíveis de valores em faixas

[1] `https://github.com/CleitonSilvaT/refactoring-vem-2025`

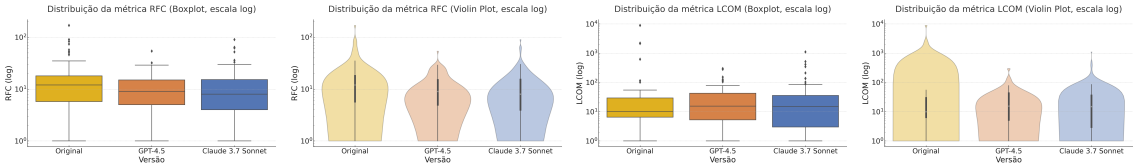


(a) Distribuição em Boxplot - *CBO* **(b)** Distribuição em Violin Plot - *CBO* **(c)** Distribuição em Boxplot - *WMC* **(d)** Distribuição em Violin Plot - *WMC*

Figura 3. Distribuição das métricas CBO e WMC utilizando Boxplot e Violin Plot.

inferiores, representando maior uniformidade e redução de complexidade funcional.

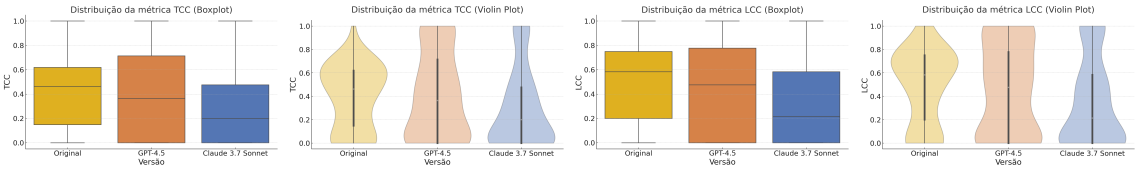
Para *Response For a Class* (RFC), a versão original demonstra alta dispersão, com média de $16,52$, mediana de 10 e valor máximo de 169 (Figs. 4(a) e 4(b)). Ambas as ferramentas reduziram esses valores: GPT 4.5 atingiu média de $9,61$, mediana de 7 e máximo de 54; Claude 3.7 Sonnet obteve média de $9,48$, mediana de $5,5$ e máximo de 90. O gráfico *boxplot* mostra esses valores de forma mais agrupada. Através do *violin plot*, é possível visualizar a concentração de valores em faixas mais baixas, indicando menor acoplamento nas classes refatoradas.



(a) Distribuição em Boxplot - *RFC* **(b)** Distribuição em Violin Plot - *RFC* **(c)** Distribuição em Boxplot - *LCOM* **(d)** Distribuição em Violin Plot - *LCOM*

Figura 4. Distribuição das métricas RFC e LCOM utilizando Boxplot e Violin Plot.

Lack of Cohesion in Methods (LCOM) evidencia comportamento assimétrico na versão original, com média de 193,53, mediana de 1 e valor máximo de 8.905, indicando a existência de classes com baixa coesão (Fig. 4(c) e 4(d)). Após as refatorações, os valores de LCOM apresentaram redução na versão GPT-4.5, a média foi de 20,21, a mediana 1 e o máximo de 300; já na versão Claude 3.7 Sonnet, média de 30,29, mediana 1 e máximo de 1.104. Embora as médias das versões refatoradas sejam menores que a original, os gráficos mostram que a dispersão dos valores continua elevada. Tanto o *boxplot* quanto o *violin plot* indicam que a coesão interna das classes permaneceu heterogênea.

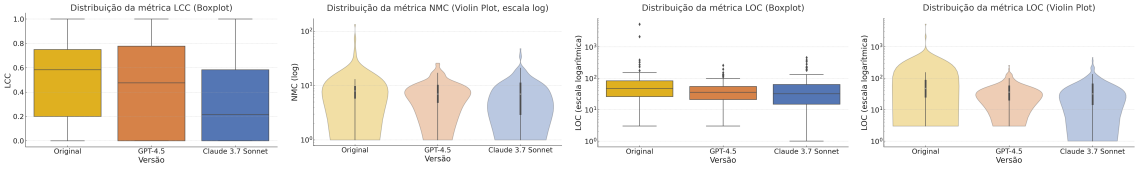


(a) Distribuição em Boxplot - *TCC* **(b)** Distribuição em Violin Plot - *TCC* **(c)** Distribuição em Boxplot - *LCC* **(d)** Distribuição em Violin Plot - *LCC*

Figura 5. Distribuição das métricas TCC e LCC utilizando Boxplot e Violin Plot.

Em *Tight Class Cohesion* (TCC), notou-se diminuição da mediana e concentração de valores mais baixos nas versões refatoradas (Figs. 5(a) e 5(b)), evidenciando redução da coesão. Analogamente, *Loose Class Cohesion* (LCC) exibiu diminuição da mediana e distribuição deslocada para níveis inferiores (Figs. 5(d) e 5(e)), sinalizando menor consistência na ligação entre métodos.

Na métrica Número de Métodos por Classe (NMC), foram constatadas média de $11,30$, mediana de 8,0 e valor máximo de 134 (Fig. 6(a) e 6(b)). As versões refatoradas indicam maior controle no tamanho funcional das classes: GPT 4.5 apresentou média de $7,93$, mediana 7 e máximo de 26; Claude 3.7 Sonnet, média de $7,43$, mediana 6 e máximo de 48. O *boxplot* e o *violin plot* mostram que as classes tiveram redução na dispersão dos dados e se tornaram mais uniformes em termos de número de métodos.



(a) Distribuição em Boxplot - *NMC* **(b)** Distribuição em Violin Plot - *NMC* **(c)** Distribuição em Boxplot - *LOC* **(d)** Distribuição em Violin Plot - *LOC*

Figura 6. Distribuição das métricas NMC e LOC utilizando Boxplot e Violin Plot.

A métrica *Lines of Code* (LOC) das classes apresentou mediana de 47, média de $174,97$ e máximo de 5297 na versão original; o GPT 4.5 reduziu esses números para mediana 36, média $45,64$ e máximo 257; o Claude, para mediana 32, média $56,04$ e máximo 458 (Figs. 6(c) e 6(d)). A menor dispersão observada no *boxplot*, aliada à concentração dos valores centrais no *violin plot*, indica uma maior uniformidade no tamanho das classes após a refatoração.

5. Discussão dos Resultados

Esta seção avalia a eficácia das ferramentas de refatoração com base em oito critérios, organizados em dois grupos: quatro quantitativos, que mensuram objetivamente alterações estruturais no código (complexidade e acoplamento via CBO e RFC, coesão com LCOM, TCC e LCC, tamanho e fragmentação com LOC e NMC, e distribuição entre tipos de refatoração), e quatro qualitativos/metodológicos, voltados à padronização do experimento, como o uso de *prompts* estruturados, aplicação de boas práticas de projeto orientado a objetos, separação adequada de novas classes em arquivos próprios e preservação das interfaces existentes. Esses critérios garantem a consistência, reprodutibilidade e robustez na comparação entre as ferramentas avaliadas.

5.1. Discussão dos Resultados Quantitativos

No que se refere à quantidade de refatorações, observou-se que ambas as ferramentas apresentaram predomínio de operações de extração de métodos em relação à extração de classes, indicando a tendência das ferramentas em atuar na simplificação interna das classes. Além disso, constatou-se que Claude 3.7 Sonnet realizou um número maior de refatorações em relação ao GPT 4.5. A avaliação das métricas CK revelou que as refatorações por IA promoveram melhorias estruturais consistentes. De maneira geral,

as versões refatoradas apresentaram redução dos níveis de acoplamento e complexidade, aumento da coesão interna e diminuição do tamanho médio das classes.

A comparação das ferramentas mostrou que o Claude 3.7 Sonnet se destacou nas métricas de redução de acoplamento (CBO e RFC), sugerindo que priorizou a diminuição de vínculos entre classes e dependências indiretas, promovendo um sistema mais modular e menos acoplado. Houve redução significativa do TCC e LCC indicando redução da coesão interna nas classes após a refatoração. Isso ocorre porque, ao dividir classes grandes em menores e mais especializadas, a coesão interna das classes permaneceu heterogênea, sugerindo que as refatorações automáticas não garantiram melhoria substancial na coesão entre métodos apesar de gerar redução no LCOM.

Quanto à simplificação estrutural, ambas as ferramentas reduziram o número total de métodos por classe e as linhas de código, facilitando a manutenção e a compreensão do sistema. No entanto, o GPT 4.5 apresentou valores médios e máximos mais baixos, sugerindo maior fragmentação e simplificação estrutural. A versão refatorada pelo Claude 3.7 Sonnet também reduziu significativamente o tamanho das classes, mas manteve algumas estruturas maiores, indicando uma abordagem menos incisiva em modularização.

Apesar das diferenças pontuais, os resultados convergentes mostram que ambas as ferramentas melhoraram aspectos essenciais da qualidade do código, alinhando-se às boas práticas de modularização e coesão em sistemas orientados a objetos. Assim, a escolha da ferramenta depende do objetivo: para reduzir dependências e aumentar modularidade global, o Claude 3.7 Sonnet pode ser mais indicado; para fortalecer coesão interna e simplificar métodos, o GPT 4.5 tende a ser mais eficaz.

5.2. Discussão dos Resultados Qualitativos

No contexto do experimento proposto, observou-se que, durante o processo de refatoração, houve uma redução progressiva na qualidade e quantidade das refatorações por linha de código à medida que o tamanho das classes aumentava. Verificou-se que classes com grande número de linhas frequentemente receberam um tratamento incompleto, sendo parcialmente ou totalmente ignoradas pelas ferramentas analisadas. Tanto a versão original do sistema quanto as versões resultantes das refatorações estão disponíveis no repositório do trabalho. Na sequência, são apresentadas e discutidas algumas das refatorações realizadas.

Como exemplo, a classe JavaLexer.java, com LOC original de 2.123, foi parcialmente refatorada, resultando em versões finais com LOC 123 pelo Claude 3.7 Sonnet e 195 pelo GPT-4.5. A classe JavaParser.java, com LOC 529, não foi refatorada pelo GPT-4.5, enquanto o Claude 3.7 Sonnet fez duas extrações de métodos e uma de código, reduzindo o LOC para 37. Isso levanta dúvidas sobre a qualidade dessas modificações, especialmente em relação à coesão interna e clareza do código. Esse comportamento está associado ao fato dessas classes superarem os limites de dados suportados por uma única requisição às ferramentas.

Na análise qualitativa, ambas as ferramentas modificaram interfaces, apesar de isso não ter sido solicitado no *prompt*. O Claude 3.7 Sonnet foi mais propenso a essas alterações, como na classe Method.java, onde criou duas interfaces adicionais não solicitadas (Figura 7). O GPT-4.5 foi mais conservador, realizando essa operação apenas

em JavaTokenTypes.java e reconhecendo corretamente as interfaces como estruturas que deveriam permanecer intactas nas demais situações.

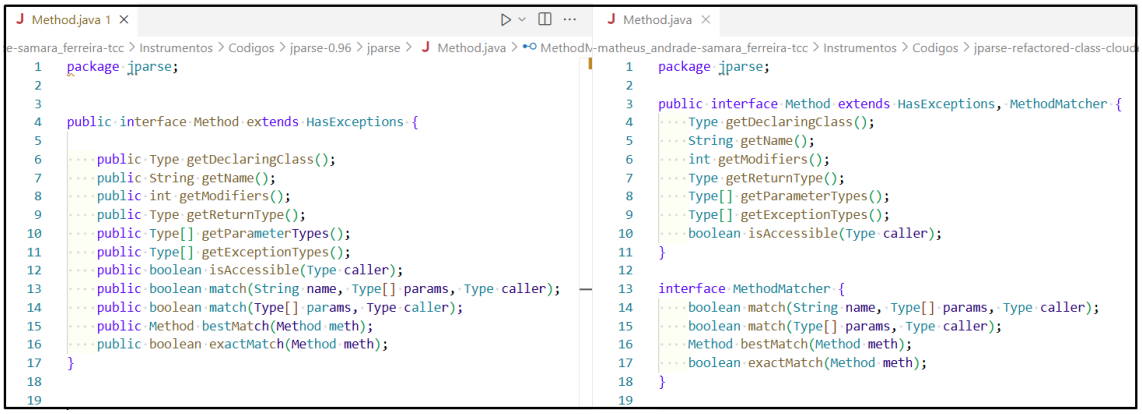


Figura 7. Classe Method com refatoração aplicada por Claude 3.7 Sonnet

Adicionalmente, verificou-se que o Claude 3.7 Sonnet promoveu mudanças nos nomes das classes durante o processo de extração, o que em contextos semelhantes pode comprometer a rastreabilidade. Outro aspecto observado é que, ao contrário do que é considerado boa prática na gestão de código, as novas classes extraídas não foram separadas em arquivos próprios, sendo mantidas dentro do arquivo original da classe refatorada, um comportamento que, em determinadas situações, pode reduzir a clareza estrutural e dificultar a manutenção futura. A classe ModifierAST.java foi selecionada para uma análise detalhada devido à discrepância no número de refatorações realizadas pelas ferramentas de inteligência artificial. Ao comparar o arquivo original com as versões produzidas pelas ferramentas, observa-se que ambas buscam aumentar a legibilidade e reduzir duplicações, porém adotam estratégias distintas. No código original, o construtor concentrava diretamente toda a lógica de inicialização e atribuição de modificadores em um único ponto, sem delegação explícita de responsabilidades.

A refatoração proposta pelo Claude 3.7 Sonnet dividiu o construtor em métodos auxiliares claramente definidos, buscando atribuir responsabilidades específicas e aumentar a clareza do código. Além disso, a criação da classe ModificadorHelper foi realizada em conformidade com o princípio de responsabilidade única, encapsulando operações específicas e reduzindo o acoplamento. Entretanto, essa dispersão em uma classe adicional pode gerar sobrecarga conceitual para situações simples, já que as operações envolvidas são trivialmente expressas em poucas linhas. Embora o método isModifier tenha sido introduzido para centralizar verificações futuras, as checagens existentes continuam utilizando diretamente Modifier.isXxx, limitando parcialmente o potencial de reutilização.

Por outro lado, a versão do GPT-4.5 introduziu a classe GerenciadorModificadores para encapsular e gerenciar o estado dos modificadores, reforçando o encapsulamento e eliminando redundâncias através do método verificaModificador. Essa abordagem está alinhada ao princípio *Don't Repeat Yourself* (DRY), eliminando duplicações de código. Contudo, o uso de um gerenciador para manter o estado pode levar a inconsistências futuras caso o campo mods seja modificado diretamente sem refletir no gerenciador, demandando disciplina adicional ou mecanismos de sincronização.

Na análise qualitativa realizada neste estudo de caso, percebeu-se que as

refatorações propostas pelo Claude 3.7 Sonnet mantiveram em grande parte a estrutura original das classes, resultando em alterações de escopo mais restrito. Em contraste, as transformações sugeridas pelo GPT-4.5 envolveram a criação de novas entidades e a redistribuição de responsabilidades entre elas, configurando um grau de reorganização estrutural mais amplo. Essas diferenças de comportamento indicam perfis distintos de atuação das ferramentas em relação ao tamanho e à complexidade do sistema analisado.

6. Ameaças à validade

Este estudo apresenta algumas ameaças à validade interna [Wohlin et al. 2012], principalmente pelo fato dos experimentos terem sido realizados sequencialmente, o que pode ter introduzido aprendizado indireto na análise das respostas. Além disso, a abordagem utilizada foi exclusivamente estática, sem execução do sistema antes ou após as refatorações, o que impede a verificação de possíveis falhas funcionais introduzidas. Há também limitações quanto à validade externa, uma vez que o projeto analisado foi apenas o *jparse*, dificultando a generalização dos resultados para outros sistemas ou linguagens, dada sua natureza e estrutura específicas.

A validade de construção [Wohlin et al. 2012] também é impactada pela escolha exclusiva das métricas CK, que, embora úteis para avaliar aspectos estruturais, não capturam atributos como legibilidade ou manutenibilidade do código. Além disso, os *code smells* não foram explicitamente informados nos *prompts*, o que fez com que a identificação e execução das refatorações dependessem totalmente da capacidade autônoma das ferramentas de IA. Isso pode ter enviesado os resultados, pois o desempenho das ferramentas também refletiu sua habilidade em detectar problemas estruturais. Por fim, a ausência de execuções repetidas compromete a validade de conclusão, já que variações naturais no comportamento das ferramentas podem ter passado despercebidas. Reforça-se, portanto, a necessidade de estudos futuros com múltiplas execuções e testes estatísticos para maior robustez.

7. Conclusão

Este trabalho apresentou uma avaliação da eficácia das ferramentas de IA GPT-4.5 e Claude 3.7 Sonnet aplicadas à refatoração de software, com base em oito critérios quantitativos e qualitativos. A abordagem permitiu identificar diferenças estruturais relevantes entre as ferramentas, destacando o potencial do conjunto de critérios como referência metodológica. A análise revelou que Claude 3.7 Sonnet realizou mais extrações de métodos e classes, enquanto ambas as ferramentas promoveram reduções significativas em acoplamento, complexidade e tamanho do código, conforme evidenciado pelas métricas CK.

A análise qualitativa indicou que as ferramentas são mais eficazes quando aplicadas a escopos menores e com objetivos de refatoração claramente definidos. Claude 3.7 Sonnet mostrou-se mais adequado para modularização e redução de dependências, ao passo que o GPT-4.5 se destacou na simplificação de métodos e menor incidência de alterações indesejadas. Apesar das limitações impostas pelo uso de um único sistema e execução única por ferramenta, os resultados oferecem uma base promissora para estudos futuros. Como sugestões de trabalhos futuros, recomenda-se a replicação com múltiplas execuções, maior diversidade de projetos e aprimoramentos nos *prompts*, de modo a aprofundar a compreensão sobre o comportamento e a efetividade das IAs em diferentes contextos de refatoração.

Referências

Almogahed, A., Mahdin, H., Zakaria, N. H., Omar, M., Barraood, S. O., and Alawadhi, A. (2023). Empirical investigation of the diverse refactoring effects on software quality: The role of refactoring tools and software size. In *International Conference on Emerging Smart Technologies and Applications (eSmartA)*.

Amazon Web Services (2024). What are embeddings in machine learning? Accessed: 25 Mar. 2025.

Chidamber, S. and Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

Eilertsen, A. M. and Murphy, G. C. (2021a). Stepwise refactoring tools. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 629–633.

Eilertsen, A. M. and Murphy, G. C. (2021b). The usability (or not) of refactoring tools. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 237–248. IEEE.

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

Ivers, J., Nord, R. L., Ozkaya, I., Seifried, C., Timperley, C. S., and Kessentini, M. (2022). Industry's cry for tools that support large-scale refactoring. In *International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pages 163–164.

Mealy, E. and Strooper, P. (2006). Evaluating software refactoring tool support. In *Australian Software Engineering Conference (ASWEC'06)*. IEEE.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 336–345.

Terra, R., Miranda, L. F., Valente, M. T., and Bigonha, R. S. (2013). Qualitas.class corpus: A compiled version of the qualitas corpus. *Software Engineering Notes*, pages 1–4.

Valente, M. T. (2020). *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. Independente.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt. *arXiv preprint arXiv:2302.11382*.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Computer Science. Springer Berlin Heidelberg.