UMA ANÁLISE ABRANGENTE DA QUALIDADE ESTRUTURAL DE SOFTWARES JAVA GERADOS POR INTELIGÊNCIA ARTIFICIAL[1]

Samyris Alves Rodrigues[2]
Kecia Aline Marques Ferreira[3]
Rogério Martins Gomes[4]

Submetido em: 17/06/2026 | Aprovado em: 17/06/2026

RESUMO

A Inteligência Artificial Generativa (IAG) vem transformando a Engenharia de Software ao automatizar tarefas de programação por meio de Modelos de Linguagem de Grande Escala (LLMs). Apesar dos ganhos de produtividade proporcionados por essas tecnologias, persistem questionamentos quanto à qualidade dos artefatos produzidos. Nesse contexto, este trabalho investiga a qualidade estrutural de softwares Java gerados por sete ferramentas de IAG: ChatGPT, Gemini, Gemini Advanced, DeepSeek, GitHub Copilot, AWS CodeWhisperer e Claude. O estudo avaliou oito sistemas Java em três períodos de execução independentes, utilizando 46 métricas de software e classificadas com base em valores de referência da literatura. A avaliação estrutural considerou apenas os softwares gerados de forma completa e funcional. Adicionalmente, foi proposto um indicador denominado *qualidade efetiva*, que combina a taxa de completude funcional e a qualidade estrutural dos sistemas gerados, permitindo uma comparação mais abrangente entre as ferramentas avaliadas. Os resultados indicam que, quando considerados apenas os sistemas completos, as ferramentas apresentam qualidade estrutural semelhante, sem diferenças expressivas entre modelos generalistas e especializados, nem entre versões gratuitas e pagas. As métricas de acoplamento e herança apresentaram predominância de classificações favoráveis, enquanto a coesão destacou-se como a principal limitação recorrente. A incorporação da completude funcional por meio da *qualidade efetiva* evidenciou diferenças mais claras entre as ferramentas, destacando AWS CodeWhisperer e DeepSeek pelos melhores resultados globais. Observou-se, ainda, elevada estabilidade temporal dos padrões identificados ao longo das execuções experimentais. Conclui-se que a avaliação de ferramentas de geração automática de código deve considerar simultaneamente aspectos estruturais e funcionais, uma vez que a qualidade arquitetural isolada não reflete integralmente a capacidade de geração de software.

Palavras-chave: Inteligência Artificial Generativa (IAG). Modelos de Linguagem de Grande Escala (LLMs). Qualidade de software. Métricas orientadas a objetos.

[1] Artigo científico elaborado para o trabalho de conclusão do curso de Engenharia de Computação do campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG.)

[2] Graduanda em Engenharia de Computação no campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). msamyris@gmail.com

[3] Orientadora do trabalho. Docente do Departamento de Computação do campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). kecia@cefetmg.br

[4] Co-orientador do trabalho. Docente do Departamento de Computação do campus Nova Gameleira do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG). rogerio@cefetmg.br

ABSTRACT

Generative Artificial Intelligence (GenAI) has been transforming Software Engineering by automating programming tasks through Large Language Models (LLMs). Despite the productivity gains enabled by these technologies, concerns remain about the quality of the software artifacts generated. In this context, this study investigates the structural quality of Java software generated by seven GenAI tools: ChatGPT, Gemini, Gemini Advanced, DeepSeek, GitHub Copilot, AWS CodeWhisperer, and Claude. Eight Java systems were evaluated across three independent execution periods using 46 software metrics and classified according to reference values established in the literature. The structural assessment considered only fully functional software systems. In addition, an indicator called *effective quality* was proposed to combine functional completeness and structural quality into a single comparative measure. The results indicate that, when only fully functional systems are considered, the evaluated tools produce software with similar structural quality, with no substantial differences between general-purpose and code-oriented tools or between free and paid versions. Coupling and inheritance metrics were predominantly classified as favorable, whereas cohesion emerged as the main recurring limitation. When functional completeness was incorporated through the *effective quality* indicator, clearer differences among the tools became evident, with AWS CodeWhisperer and DeepSeek achieving the best overall results. High temporal stability was also observed across the experimental executions. Overall, the findings suggest that the evaluation of AI-based code generation tools should jointly consider structural and functional aspects, since architectural quality alone does not fully reflect software generation capability.

Keywords: Generative Artificial Intelligence (GenAI). Large Language Models (LLMs). Software quality. Object-Oriented Software Metrics.

1 INTRODUÇÃO

A incorporação de ferramentas de Inteligência Artificial Generativa (IAG) no desenvolvimento de software representa uma mudança de paradigma na Engenharia de Software contemporânea. Modelos de Linguagem de Grande Escala (*Large Language Models* — LLMs) expandiram a capacidade de automação de atividades ao longo do ciclo de vida do software, isto é, desde a elicitação de requisitos até as tarefas de implementação e manutenção (Nguyen-Duc *et al.*, 2023). Esse avanço tem promovido ganhos relevantes de produtividade, permitindo que os desenvolvedores concentrem esforços em atividades de maior complexidade conceitual. Apesar da crescente adoção dessas ferramentas, persistem incertezas importantes quanto à qualidade estrutural dos códigos produzidos pelas LLMs.

Estudos recentes indicam que, embora as ferramentas de IAG frequentemente produzam código funcionalmente correto, os artefatos gerados podem apresentar fragilidades arquiteturais relevantes, incluindo acoplamento excessivo, baixa coesão e aumento da complexidade estrutural. Yetiştiren *et al.* (2023), por exemplo, avaliaram ferramentas como ChatGPT, GitHub Copilot e Amazon CodeWhisperer utilizando técnicas de análise estática, reportando a presença de falhas, *code smells* e vulnerabilidades em parte dos códigos gerados. De forma semelhante, Sabra *et al.* (2025) discutem questões relacionadas à qualidade estrutural e à segurança de códigos produzidos por Inteligência Artificial Generativa. Além disso, Wang *et al.* (2025) demonstraram que o resultado de LLMs tende

a deteriorar-se em tarefas mais complexas, principalmente quando os requisitos exigem maior extensão textual ou lógica de programação mais elaborada. Pereira (2025) observou que softwares Java gerados por IAG podem apresentar falhas estruturais e problemas de compilação, mesmo quando aparentemente atendem aos requisitos funcionais solicitados.

Embora relevantes, grande parte dos estudos recentes sobre geração automática de código por Inteligência Artificial Generativa baseia-se em ferramentas de análise voltadas predominantemente à manutenção prática, como o SonarQube, cuja ênfase recai principalmente sobre segurança, vulnerabilidades e detecção de bugs. Nesse contexto, avaliações mais profundas da qualidade estrutural do software permanecem subexploradas. No caso dos softwares orientados a objetos, é importante avaliar a qualidade dos softwares gerados quanto aos aspectos de modularidade, coesão e acoplamento, dentre outros, pois são elementos essenciais da qualidade estrutural desses softwares. Além disso, ainda são escassos estudos comparativos que investiguem simultaneamente variáveis como o modelo econômico da ferramenta (versões gratuitas *versus* pagas) e o grau de especialização dos modelos, diferenciando LLMs generalistas de plataformas dedicadas à geração de código. Essas limitações evidenciam a necessidade de investigações mais aprofundadas sobre a qualidade estrutural de softwares gerados automaticamente por LLMs.

Diante desse panorama, o presente trabalho tem como objetivo avaliar a qualidade estrutural de softwares gerados por ferramentas de Inteligência Artificial Generativa. A avaliação da qualidade estrutural é realizada por meio de métricas de software e de seus valores de referência previamente definidos na literatura. A linguagem de programação considerada neste trabalho é Java por ser uma linguagem amplamente utilizada e pelo fato de a literatura possuir ferramenta consolidada de coleta de métricas específicas da orientação a objetos e valores de referência para essas métricas em software desenvolvido em Java. Foram analisadas 46 métricas de software coletadas pela ferramenta CK Tool (Aniche, 2015) e utilizados os valores de referência definidos por Gonçalves (2022). A partir de um levantamento das ferramentas de IAG para geração de código, foram selecionadas sete ferramentas para este estudo: ChatGPT, Gemini, Gemini Advanced, DeepSeek, Claude.ai, AWS CodeWhisperer e GitHub Copilot. As quatro primeiras são ferramentas de propósito geral e as três últimas são específicas para a geração de código. As ferramentas Gemini Advanced e GitHub Copilot são de acesso pago, enquanto as demais são gratuitas.

Especificamente, este trabalho investiga as seguintes questões de pesquisa:

RQ1 Qual o nível de qualidade estrutural dos softwares gerados pelas ferramentas?

RQ2 Ferramentas especializadas em geração de código produzem softwares estruturalmente superiores aos produzidos por LLMs generalistas?

RQ3 As ferramentas apresentam estabilidade temporal em sua Qualidade Efetiva ao longo das diferentes execuções experimentais?

Este trabalho traz as seguintes contribuições principais: 1) realiza um levantamento de ferramentas de IAG para geração de software; 2) emprega métricas e seus valores de referência para analisar a qualidade estrutural dos softwares gerados por IAGs; 3) compara a qualidade estrutural do código gerado por versões gratuitas e pagas das ferramentas; 4) analisa o comportamento de plataformas especializadas em geração de software em relação a LLMs de propósito geral; 5) propõe um indicador denominado Qualidade Efetiva, que integra qualidade estrutural e completude funcional para permitir uma avaliação

comparativa mais abrangente das ferramentas.. O estudo considera aspectos ainda pouco explorados na literatura, especialmente o modelo econômico das ferramentas e o grau de especialização dos modelos. Assim, busca-se oferecer subsídios empíricos para uma adoção mais criteriosa de ferramentas de Inteligência Artificial Generativa na Engenharia de Software.

Além da análise individual da completude funcional e da qualidade estrutural, este trabalho propõe um indicador denominado Qualidade Efetiva, que integra ambos os aspectos em uma única medida. O objetivo desse indicador é evitar interpretações potencialmente enviesadas decorrentes da análise exclusiva de métricas estruturais obtidas apenas para os sistemas gerados com sucesso, permitindo uma avaliação mais abrangente do resultado das ferramentas.

O restante deste trabalho está organizado da seguinte forma: Seção 2 discute os trabalhos relacionados; Seção 3 apresenta a fundamentação teórica; Seção 4 descreve a metodologia adotada; Seção 5 apresenta e discute os resultados obtidos; Seção 6 discute as principais ameaças à validade do estudo; por fim, Seção 7 traz as considerações finais e trabalhos futuros.

2 TRABALHOS RELACIONADOS

A investigação científica sobre a aplicação de Inteligência Artificial Generativa (IAG) na Engenharia de Software tem avançado de forma acelerada, embora ainda existam lacunas significativas quanto à qualidade estrutural dos artefatos produzidos. Nguyen-Duc *et al.* (2023) delinearam uma agenda abrangente de pesquisa que destaca desafios relacionados à confiabilidade, segurança e dependabilidade dos modelos generativos, enfatizando a necessidade de avaliações empíricas rigorosas sobre a qualidade do código produzido.

Estudos empíricos recentes têm revelado limitações concretas nos códigos gerados por LLMs. Yetiştiren *et al.* (2023), analisaram o resultado de ferramentas como ChatGPT, GitHub Copilot e Amazon CodeWhisperer utilizando o conjunto de dados *HumanEval*. Os autores observaram que, embora os modelos apresentem níveis relevantes de corretude funcional, os artefatos produzidos frequentemente contêm bugs, *code smells* e vulnerabilidades detectadas por ferramentas de análise estática. O estudo evidencia que o resultado funcional satisfatório não implica, necessariamente, boa qualidade estrutural.

Além das análises focadas em corretude funcional, Nguyen e Nadi (2022) discutiram os impactos do uso crescente de ferramentas de geração automática de código sobre práticas de manutenção e evolução de software. Os autores argumentaram que, embora modelos generativos acelerem a implementação inicial, a ausência de controle arquitetural e de consistência estrutural pode ampliar a dívida técnica ao longo do ciclo de vida do sistema. Tal perspectiva reforça a necessidade de avaliações que considerem não apenas a funcionalidade imediata do código produzido, mas também sua qualidade interna e sustentabilidade evolutiva.

Sabra *et al.* (2025), por sua vez, conduziram uma análise em larga escala envolvendo milhares de artefatos gerados por LLMs em diferentes linguagens de programação. Utilizando ferramentas de análise estática, os autores identificaram que entre 90% e 93% dos problemas encontrados estavam associados a deficiências de design e *code smells*, indicando fragilidades estruturais persistentes. Além disso, o estudo aponta que a recorrência desses problemas pode aumentar significativamente a dívida técnica e comprometer atividades futuras de manutenção, evolução e reutilização do software.

De forma complementar, Wang *et al.* (2025) investigaram erros sintáticos, semânticos e lógicos típicos cometidos por LLMs e observaram que a complexidade do enunciado exerce influência direta sobre o resultado dos modelos. Os autores verificaram que tarefas mais extensas e descrições mais complexas resultam em elevada incidência de *garbage code*, revelando limitações intrínsecas às capacidades generativas dos modelos atuais.

Pereira (2025) avaliou a qualidade estrutural de softwares Java gerados por IAG utilizando 46 métricas de software orientado a objetos e seus valores de referência definidos por Gonçalves (2022). O trabalho considerou quatro ferramentas — ChatGPT 4.0, DeepSeek V3, Gemini e GitHub Copilot — e analisou oito sistemas Java de diferentes domínios e níveis de complexidade. Os principais resultados indicaram que as ferramentas GitHub Copilot, ChatGPT e DeepSeek apresentaram os melhores resultados médios de qualidade estrutural, gerando códigos com nível comparável ao de softwares desenvolvidos por profissionais nos aspectos de qualidade de acoplamento, herança e ocultação de informação. A ferramenta Gemini obteve os piores resultados. A coesão revelou-se uma limitação recorrente em todas as ferramentas avaliadas. Além disso, o DeepSeek destacou-se com o maior percentual de sistemas funcionalmente completos (66,6%), enquanto a Gemini apresentou maior incidência de códigos incompletos ou que não compilaram. Entretanto, o estudo não considerou ferramentas como AWS CodeWhisperer e Claude.ai, não incluiu versões pagas das ferramentas como Gemini Advanced, não investigou o impacto do modelo econômico (gratuito *versus* pago) sobre a qualidade do software gerado e não realizou execuções temporais distintas para avaliar a estabilidade dos modelos. O presente trabalho utiliza os mesmos *prompts*, métricas, valores de referência e ferramenta de medição de Pereira (2025), diferenciando-se ao incluir três ferramentas adicionais (AWS CodeWhisperer, Claude.ai e Gemini Advanced), ao realizar três execuções temporais independentes para cada combinação de ferramenta e sistema, e ao investigar variáveis como o tipo de acesso (gratuito ou pago) e a especialização dos modelos.

Apesar dessas contribuições, ainda são escassos os estudos que comparem sistematicamente, no ecossistema Java, ferramentas generalistas e especializadas, bem como versões gratuitas e pagas. Esta pesquisa busca preencher essa lacuna por meio de uma avaliação baseada em métricas estruturais coletadas pela CK Tool, considerando diferentes categorias de ferramentas de Inteligência Artificial Generativa e a estabilidade dos resultados ao longo do tempo.

3 FUNDAMENTAÇÃO TEÓRICA

Esta seção apresenta os conceitos fundamentais empregados neste trabalho.

3.1 Qualidade de Software

A qualidade de software constitui um dos pilares fundamentais da Engenharia de Software, estando diretamente associada à confiabilidade, manutenibilidade, reutilização e evolução dos sistemas ao longo de seu ciclo de vida. Bartié (2013) destaca que a qualidade de software não se restringe apenas ao funcionamento correto do sistema, mas envolve também aspectos relacionados à organização interna do código, facilidade de manutenção, capacidade de evolução e redução de falhas futuras. Nesse contexto, atributos como modularidade, legibilidade e manutenibilidade tornam-se essenciais para garantir sustentabilidade arquitetural e reduzir custos associados à evolução do software.

Com o crescimento da complexidade dos sistemas modernos, a literatura aponta que problemas estruturais acumulados ao longo do desenvolvimento podem resultar em degradação arquitetural progressiva e aumento da dívida técnica. Assim, avaliar a qualidade estrutural do software torna-se fundamental para compreender não apenas o comportamento funcional imediato do sistema, mas também sua capacidade de manutenção, reutilização e evolução ao longo do tempo (Masuda *et al.*, 2018).

Em uma perspectiva complementar, Pressman e Maxim (2020) compreendem a qualidade de software como o grau em que um sistema atende aos requisitos funcionais e não funcionais especificados, além de aderir a padrões e boas práticas de desenvolvimento. Em conjunto, essas definições evidenciam que a qualidade de software envolve tanto atributos externos, percebidos pelos usuários, quanto atributos internos, relacionados à estrutura, organização e manutenibilidade do código.

O modelo de qualidade de software estabelecido pela International Organization for Standardization (2023) define características que norteiam a mensuração, especificação e avaliação de sistemas, tais como: adequação funcional, eficiência de desempenho, compatibilidade, capacidade de interação, confiabilidade, segurança, manutenibilidade, modularidade, flexibilidade e segurança física. Esses atributos fornecem um referencial sistemático para avaliar tanto os produtos e dados quanto o impacto do sistema sobre seus usuários.

3.2 Qualidade Estrutural em Sistemas Orientados a Objetos

No paradigma orientado a objetos, a qualidade estrutural está diretamente relacionada à forma como classes, métodos e dependências são organizados internamente. Características como coesão, acoplamento, modularidade e complexidade exercem influência significativa sobre atributos de qualidade, incluindo testabilidade, reutilização, manutenção e evolução arquitetural.

Segundo Pressman e Maxim (2020), a modularidade permite subdividir sistemas complexos em partes independentes, cada uma com responsabilidades específicas, sendo a coesão e o acoplamento seus principais direcionadores:

- **Coesão**: grau de relação entre os elementos internos de um módulo ou classe. É desejável alta coesão, de modo que os componentes trabalhem juntos para cumprir uma única responsabilidade.
- **Acoplamento**: grau de dependência entre diferentes módulos. É desejável baixo acoplamento, pois aumenta a flexibilidade e a independência do sistema, evitando impactos em cascata e facilitando a manutenção.

Outro aspecto relevante é a herança, que permite o reuso de atributos e comportamentos sem comprometer a coesão ou aumentar o acoplamento, desde que aplicada com critério.

Segundo Aniche (2015), sistemas com alto acoplamento e baixa coesão tendem a apresentar maior propagação de erros, dificuldades de manutenção e menor flexibilidade evolutiva. De forma semelhante, Masuda *et al.* (2018) ressaltam que a modularidade e a separação adequada de responsabilidades são fatores essenciais para reduzir complexidade estrutural e facilitar atividades futuras de manutenção e evolução do software.

Além disso, a ausência de controle arquitetural consistente pode ampliar a dívida técnica e comprometer atributos fundamentais, como modularidade, manutenibilidade e sustentabilidade evolutiva, especialmente em cenários de geração automática de código. Tais aspectos tornam-se particularmente relevantes no contexto atual de ferramentas baseadas em IAG, nas quais a produção automatizada de software pode introduzir fragilidades estruturais difíceis de identificar apenas por inspeção funcional.

3.3 Métricas CK

Uma das abordagens mais consolidadas para avaliação estrutural em sistemas orientados a objetos é o conjunto de métricas CK (*Chidamber and Kemerer Metrics Suite*), proposto originalmente por Chidamber e Kemerer (1994). Essas métricas foram desenvolvidas com o objetivo de quantificar características arquiteturais relevantes em projetos orientados a objetos, sendo amplamente utilizadas tanto na academia quanto em estudos empíricos de qualidade de software.

O conjunto é composto por seis métricas principais:

- **CBO** (*Coupling Between Objects*): mede o grau de dependência entre classes;
- **LCOM** (*Lack of Cohesion in Methods*): avalia o nível de coesão entre métodos de uma classe;
- **WMC** (*Weighted Methods per Class*): representa a complexidade associada aos métodos de uma classe;
- **DIT** (*Depth of Inheritance Tree*): mede a profundidade da hierarquia de herança.
- **NOC** (*Number of Children*): mede o número de subclasses diretas de uma determinada classe;
- **RFC** (*Response for a Class*): mede o número de métodos que podem ser invocados em resposta a uma mensagem recebida por um objeto da classe;

Segundo Chidamber e Kemerer (1994), essas métricas permitem identificar potenciais problemas arquiteturais relacionados à modularidade, complexidade e facilidade de manutenção do sistema. Em aplicações orientadas a objetos, elevados níveis de acoplamento e baixa coesão frequentemente indicam maior dificuldade de evolução e maior suscetibilidade à propagação de falhas.

Além das métricas CK originais, a literatura registra métricas complementares que avaliam dimensões como tamanho, complexidade e ocultação de informação. Embora tradicionalmente classificada como um princípio de projeto, neste trabalho a ocultação de informação é tratada como dimensão de qualidade estrutural mensurável por meio das métricas de visibilidade descritas na Tabela 3. A descrição das principais métricas utilizadas neste trabalho encontra-se na Tabela 3, apresentada no Anexo A.

No contexto deste trabalho, as métricas de software extraídas pela CK Tool são particularmente relevantes por possibilitarem uma avaliação mais aprofundada da estrutura interna dos softwares gerados por ferramentas de IAG, permitindo identificar fragilidades arquiteturais que nem sempre são evidenciadas por métricas tradicionais de manutenção.

3.4 Ferramentas de Coleta de Métricas de Software

Além das métricas estruturais clássicas, ferramentas modernas de análise estática vêm sendo amplamente empregadas na indústria para avaliação da qualidade de código. Plataformas como SonarQube e SonarCloud permitem detectar bugs, vulnerabilidades de segurança, *code smells* e dívida técnica, oferecendo suporte à manutenção prática e à integração contínua. Segundo Gonçalves (2022), ferramentas de análise estática desempenham um papel importante na identificação automatizada de problemas de implementação e padrões inadequados de desenvolvimento. Entretanto, embora sejam eficazes na detecção de falhas operacionais e problemas de manutenção, essas ferramentas nem sempre capturam adequadamente aspectos mais profundos relacionados à arquitetura orientada a objetos.

Neste trabalho, a ferramenta utilizada para a coleta de métricas é a CK Tool, desenvolvida por Aniche (2015) e disponível como projeto de código aberto. CK Tool é uma ferramenta de análise estática voltada à extração de métricas de software orientado a objetos em projetos Java. Ela coleta métricas em nível de classe, incluindo as métricas originalmente propostas por Chidamber e Kemerer (1994), além de métricas adicionais relacionadas a tamanho, complexidade, herança e ocultação de informação. A ferramenta opera por meio da análise do código-fonte Java, sem necessidade de compilação prévia, o que permite avaliar inclusive artefatos que apresentem dependências externas não resolvidas.

A partir das métricas coletadas pela CK Tool, Gonçalves (2022) derivou estatisticamente valores de referência para 46 métricas de software orientado a objetos em Java. Esses valores de referência foram organizados em três faixas — Bom/Frequente, Regular/Ocasional e Ruim/Raro — com base na distribuição empírica dos valores observados em um conjunto representativo de projetos Java. Dessa forma, os valores de referência constituem um parâmetro de comparação da estrutura de um software em relação ao comportamento típico de softwares desenvolvidos por profissionais. Esse conjunto de 46 métricas e seus respectivos valores de referência é adotado como base para a avaliação da qualidade estrutural neste trabalho, conforme descrito na Tabela 4, apresentadas no Anexo A.

Dessa forma, a utilização da CK Tool como instrumento de coleta, associada aos valores de referência derivados por Gonçalves (2022), possibilita uma avaliação objetiva e sistematizada da qualidade estrutural dos softwares gerados por ferramentas de IAG.

3.5 Qualidade Estrutural em Código Gerado por Inteligência Artificial Generativa

O surgimento de agentes de codificação aumentou a produtividade dos desenvolvedores. Em contrapartida, esses assumem menos o papel de "escritores de código" e mais o papel de "orquestradores" ou revisores (Watanabe *et al.*, 2026). Embora as tarefas sejam concluídas mais rapidamente, há fatores que tornam o processo mais lento: apesar de os códigos gerados serem funcionalmente corretos, muitas vezes é necessário reescrever funções, gerenciar contexto e lidar com código não escrito pelo próprio desenvolvedor (Becker *et al.*, 2025).

Nesse contexto, avaliações estruturais assumem papel relevante em cenários de geração automática de código, uma vez que modelos generativos podem produzir artefatos sintaticamente válidos, mas arquiteturalmente inconsistentes. Assim, métricas clássicas de orientação a objetos e ferramentas de análise estática tornam-se instrumentos importantes para investigar atributos internos de qualidade relacionados à organização estrutural do

software.

Dessa forma, a utilização combinada de métricas de software orientado a objetos e ferramentas de análise estática permite uma avaliação mais abrangente dos artefatos produzidos por Inteligência Artificial Generativa, integrando aspectos arquiteturais e estruturais da qualidade de software.

4 METODOLOGIA

Esta seção descreve a metodologia adotada para a condução deste trabalho.

4.1 Caracterização da Pesquisa

Este trabalho caracteriza-se como uma pesquisa empírica de natureza aplicada, com abordagem quantitativa e caráter experimental, voltada à avaliação da qualidade estrutural de softwares Java gerados por ferramentas de Inteligência Artificial Generativa.

O estudo adota um desenho comparativo, investigando diferenças entre modelos de linguagem de propósito geral e assistentes especializados em geração de código. Além disso, considera variáveis como tipo de ferramenta, modelo de acesso, sistema avaliado, completude funcional e estabilidade dos resultados ao longo de três execuções experimentais.

O fluxo geral da metodologia adotada neste trabalho é apresentado na Figura 1. O diagrama sintetiza as etapas do protocolo experimental, organizadas em sete blocos: (1) caracterização da pesquisa; (2) seleção e caracterização das ferramentas avaliadas, incluindo a etapa preliminar (2.1) de triagem das ferramentas; (3) definição dos oito sistemas Java avaliados; (4) desenho experimental, com a especificação das variáveis independentes e dependentes; (5) execução e coleta dos artefatos, em três execuções temporais independentes; (6) avaliação dos artefatos, subdividida em validação funcional (6.1) e avaliação estrutural com a CK Tool (6.2); e (7) análise integrada dos dados.

4.2 Seleção e Caracterização das Ferramentas Avaliadas

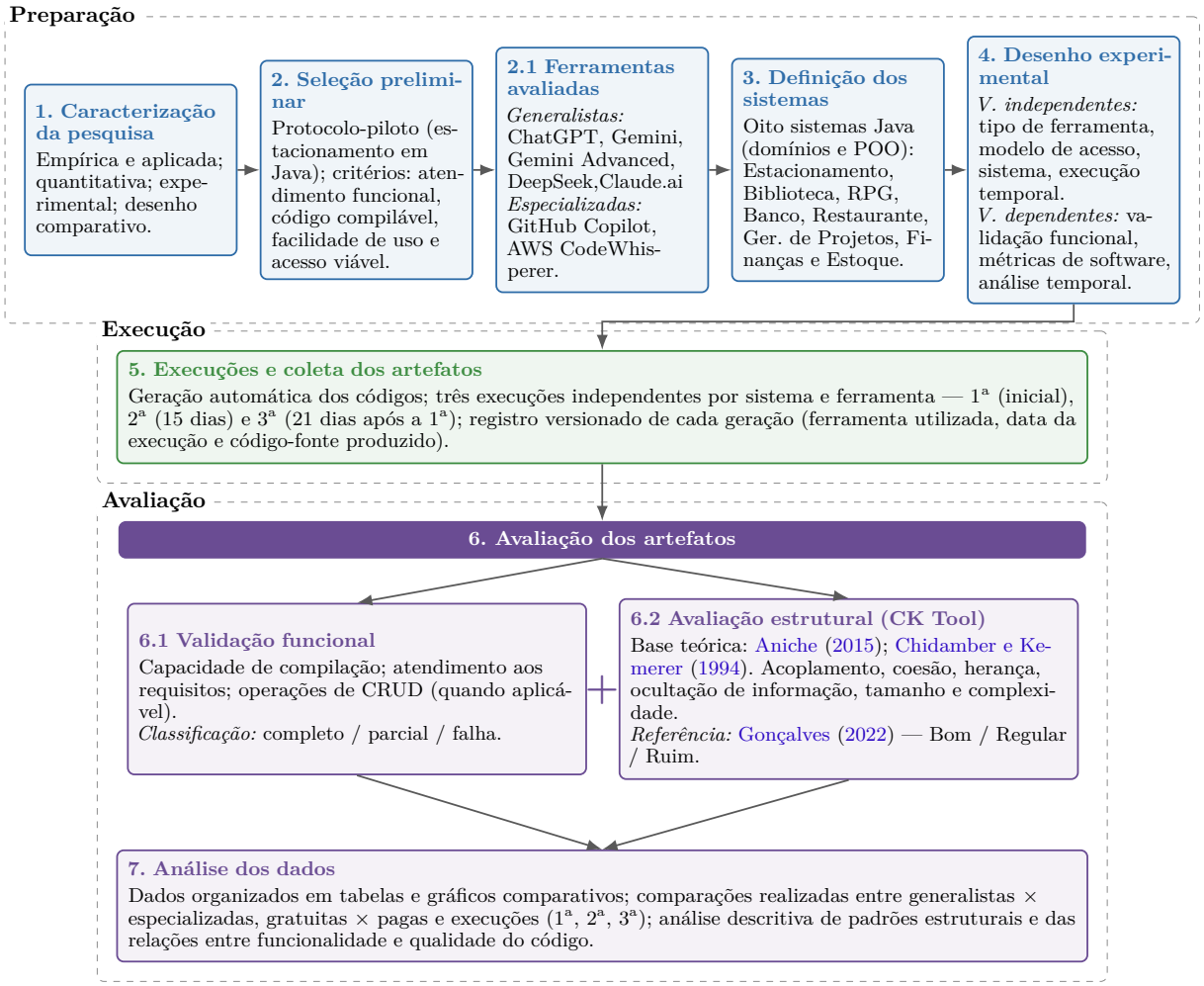
A primeira etapa metodológica consistiu no levantamento e na seleção das ferramentas de Inteligência Artificial Generativa disponíveis para geração de código Java. A seleção foi orientada pelos seguintes critérios: natureza da ferramenta, considerando modelos generalistas e assistentes especializados em programação; tipo de acesso, considerando versões gratuitas, acadêmicas ou pagas; facilidade de uso em ambientes de desenvolvimento; e capacidade de gerar código Java compilável.

Como teste preliminar de viabilidade, foi utilizado um sistema de gerenciamento de estacionamento em Java, inspirado no protocolo de Pereira (2025). O *prompt* utilizado foi:

> "Faça um sistema de gerenciamento de estacionamento completo em Java, em que seja possível: estacionar um veículo; remover veículo estacionado; verificar a quantidade de vagas disponíveis; e cadastrar a quantidade de vagas" (Pereira, 2025, p. 30).

Foram analisadas, ao todo, 13 ferramentas: AWS CodeWhisperer, ChatGPT, Claude.ai, CodeGeeX, Codex, DeepSeek, Gemini, Gemini Advanced, GitHub Copilot, Lovable.dev, Perplexity AI, Qwen, Tess AI.

Figura 1 – Fluxo da metodologia adotada neste trabalho.



Fonte: elaborado pela autora (2026).

As ferramentas a seguir foram descartadas por não atenderem aos critérios definidos no protocolo experimental, seja por não gerarem código compilável, por não oferecerem versão gratuita viável ou por não se adequarem ao escopo da pesquisa:

- **Perplexity AI:** utiliza modelos externos, sem diferencial relevante em relação às demais ferramentas selecionadas.
- **Qwen:** não apresentou diferenciais significativos frente aos modelos mais consolidados.
- **Tess AI:** versão gratuita limitada a sete dias, inviável para a experimentação contínua exigida pelas três execuções temporais.
- **Lovable.dev:** foco exclusivo em aplicações web (JavaScript, React), fora do escopo Java deste trabalho.
- **CodeGeeX:** *plugin* para o VSCode voltado ao autocompletar código, e não à geração completa de sistemas.
- **Codex:** versão gratuita sem suporte à geração de código no período dos testes, inviável por limitações de acesso.

Foram consideradas aptas para o protocolo principal as ferramentas que geraram código compilável, atenderam aos requisitos funcionais mínimos, apresentaram facilidade de uso e possuíam acesso viável para a realização dos experimentos. A partir desse procedimento, foram selecionadas sete ferramentas: ChatGPT, Gemini, Gemini Advanced, DeepSeek, GitHub Copilot, Claude.ai e AWS CodeWhisperer.

Para garantir a padronização e a rastreabilidade do experimento, registraram-se as versões específicas dos modelos utilizados durante os testes, bem como as configurações adotadas nas ferramentas que operam com seleção automática de modelos. A Tabela 1 apresenta a caracterização das ferramentas selecionadas.

Tabela 1 – Ferramentas de IAG selecionadas para o estudo.

Ferramenta	**Modelo / configuração utilizada**	**Categoria**	**Acesso**	**Observação metodológica**
ChatGPT	GPT-4o (Omni)	Modelo generalista	Gratuito	Versão básica gratuita disponível no período dos testes
Gemini	Gemini 3 Fast	Modelo generalista	Gratuito	Versão voltada à menor latência
Gemini Advanced	Gemini 3.1 Pro	Modelo generalista	Pago	Versão voltada a raciocínio avançado. Utilizada por meio de licença de estudante (Google One AI Premium para estudantes).
DeepSeek	DeepSeek-V3.2 com funcionalidade *Search* ativada	Modelo generalista	Gratuito	Busca ativada durante os testes
Claude.ai	Claude Sonnet 4.6	Modelo generalista / *chatbot*	Gratuito	Modelo Claude Sonnet 4.6 acessado pela plataforma claude.ai em sua versão gratuita, sem assinatura Claude Pro.
AWS CodeWhisperer	Seleção automática	Assistente especializado	Gratuito	A própria plataforma realiza o roteamento interno do modelo
GitHub Copilot	Seleção *Smart*/Inteligente	Assistente especializado	Pago	A ferramenta utiliza seleção automática conforme o contexto da tarefa. Utilizado por meio de licença educacional gratuita (GitHub Student Developer Pack).

Fonte: elaborado pela autora (2026).

Como algumas plataformas utilizam roteamento automático ou atualizações contínuas, as versões e configurações apresentadas na Tabela 1 correspondem ao registro realizado durante o período experimental. No caso de ferramentas com seleção automática de modelos, como o AWS CodeWhisperer e o GitHub Copilot, a análise considera a configuração efetivamente utilizada pela plataforma, sem atribuir manualmente um modelo fundacional fixo. Entende-se por modelo fundacional (*foundation model*) o modelo de

linguagem de grande escala, pré-treinado em grande volume de dados, que serve de base para a ferramenta e a partir do qual são derivadas as respostas. Nessas duas ferramentas, o modelo fundacional não é explicitado ao usuário: o AWS CodeWhisperer realiza internamente o roteamento do modelo, de modo que foi utilizada a configuração padrão da plataforma "Auto"; por outro lado, GitHub Copilot foi empregado no modo de seleção automática (*Smart*/Inteligente), no qual a própria ferramenta escolhe o modelo conforme o contexto da tarefa. Em ambos os casos, registrou-se a configuração ativa no período dos experimentos, ainda que o modelo de base subjacente não seja informado pela plataforma.

4.3 Definição dos Sistemas Avaliados

Após a seleção das ferramentas, foram definidos oito sistemas Java para compor o experimento: Estacionamento, Biblioteca, RPG, Banco, Restaurante, Gerenciamento de Projetos, Finanças e Estoque. Esses sistemas são os mesmos definidos no protocolo experimental de Pereira (2025) e foram selecionados por abranger diferentes domínios de aplicação e níveis de complexidade conceitual.

Os enunciados envolveram aspectos como algoritmos e estruturas de dados, relacionamentos entre entidades, validação de regras de negócio, tratamento de exceções e organização orientada a objetos. Cabe destacar que os enunciados não restringiram o tipo de interface a ser implementada; ainda assim, todas as ferramentas avaliadas geraram, em todas as execuções, sistemas com interface em modo caractere (terminal), executados via console Java. Não foram observadas, no conjunto avaliado, soluções com interface gráfica (Swing, JavaFX), arquitetura web ou microsserviços. Esse padrão de arquitetura é considerado na análise estrutural apresentada na Seção 5. Os enunciados completos utilizados nos experimentos foram disponibilizados no repositório GitHub do projeto.[5]

4.4 Desenho Experimental e Periodicidade das Execuções

O desenho experimental deste trabalho foi estruturado para comparar a qualidade funcional e estrutural de softwares Java gerados por diferentes ferramentas de Inteligência Artificial Generativa. Para isso, foram considerados oito sistemas Java, sete ferramentas e três execuções independentes para cada combinação entre ferramenta e sistema.

As variáveis independentes consideradas foram:

- **tipo de ferramenta**, classificada em modelos generalistas e assistentes especializados em geração de código;
- **modelo de acesso**, considerando ferramentas gratuitas e pagas;
- **sistema avaliado**, considerando os oito domínios definidos no protocolo experimental;
- **execução temporal**, composta pelas três execuções.

As variáveis dependentes corresponderam aos indicadores obtidos a partir da validação funcional, considerando compilação, atendimento aos requisitos e classificação dos resultados como completo, parcial ou falha; das métricas de software, extraídas por

[5] Disponível em: <https://github.com/Samyris/TCC>. Acesso em: 18 maio 2026.

meio da CK Tool e classificadas segundo os valores de referência de Gonçalves (2022); e da análise temporal, considerando a variação dos resultados funcionais e estruturais entre as três execuções.

Para analisar criticamente a evolução e a consistência das ferramentas, estabeleceu-se um cronograma de avaliação temporal para cada sistema. A primeira execução serviu como referência de resultado. A segunda ocorreu após um intervalo de 15 dias, e a terceira foi realizada 21 dias após a execução inicial. A definição de três execuções e desses intervalos seguiu critérios pragmáticos compatíveis com o cronograma do trabalho de conclusão de curso. O número de três execuções foi adotado para permitir a observação de padrões temporais sem inflar excessivamente o volume total de experimentos (que já totalizam 168 combinações de ferramenta × sistema × execução). O intervalo de 15 dias entre a primeira e a segunda execução foi determinado pela viabilidade operacional e pelo fluxo do estudo, enquanto o intervalo para a terceira execução (6 dias adicionais, totalizando 21 dias desde a primeira) buscou captar variações em uma janela temporal mais curta.

Esse distanciamento temporal permitiu investigar se eventuais atualizações das plataformas ou características probabilísticas dos modelos poderiam influenciar a completude funcional e a qualidade estrutural dos códigos gerados.

4.5 Validação Funcional

A validação funcional consistiu na verificação da compilação dos códigos e no atendimento aos requisitos especificados nos enunciados. Quando aplicável ao domínio do sistema, também foram verificadas operações básicas de CRUD (*Create*, *Read*, *Update*, *Delete*), isto é, criar, consultar, atualizar e excluir dados. Essa avaliação foi conduzida de forma manual, classe por classe e funcionalidade por funcionalidade, sem o uso de testes automatizados. Para cada sistema gerado, a autora executou o código no terminal e verificou, em relação ao enunciado original do *prompt*, se cada requisito explicitamente solicitado havia sido implementado e se o fluxo correspondente operava sem erros que impedissem seu uso.

Cada execução foi classificada em uma das seguintes categorias:

- **Completo:** código que compila e atende integralmente aos requisitos funcionais especificados no enunciado, executando corretamente todas as funcionalidades previstas. Pequenas imperfeições visuais, mensagens de console ou outros detalhes que não comprometam a execução nem o comportamento esperado das funcionalidades não descaracterizam essa classificação.

- **Parcial:** código que compila e executa, mas atende apenas parcialmente aos requisitos funcionais definidos no enunciado. Nessa categoria enquadram-se implementações que omitem requisitos, apresentam funcionalidades incompletas, substituem entradas previstas por dados fictícios, deixam de realizar persistência quando exigida ou não implementam corretamente operações de CRUD aplicáveis.

- **Falha:** código que não compila ou que, embora compile, apresenta erro de execução que impede a utilização de qualquer funcionalidade prevista no enunciado. Nessa categoria enquadram-se, por exemplo, programas que encerram sua execução devido a exceções não tratadas ou outros erros críticos antes que seja possível exercer as funcionalidades implementadas.

Essa classificação permitiu distinguir códigos apenas estruturalmente simples de sistemas efetivamente funcionais. Cabe ressaltar, ainda, que completude funcional não é sinônimo de ausência de erros: sistemas classificados como Completos podem conter erros pontuais (por exemplo, mensagens de erro inadequadas, validações ausentes ou comportamento inesperado em entradas atípicas).

4.6 Avaliação Estrutural

A avaliação estrutural dos softwares gerados foi realizada utilizando a CK Tool, ferramenta de análise estática desenvolvida por Aniche (2015), voltada à extração de métricas de software orientado a objetos. A ferramenta coleta 46 métricas de software, incluindo as métricas originalmente propostas por Chidamber e Kemerer (1994).

As métricas extraídas são apresentadas no Anexo A. Elas foram agrupadas nos seguintes oito aspectos de qualidade:

- **Acoplamento**: inclui as métricas CBO, FAN-IN, FAN-OUT, RFC e NOSI;
- **Coesão**: envolve as métricas LCOM, TCC e LCC;
- **Herança**: corresponde às métricas DIT e NOC;
- **Ocultação de informação**: corresponde às métricas privateMethodsQty, protectedMethodsQty, defaultMethodsQty, privateFieldsQty, protectedFieldsQty, defaultFieldsQty;
- **Tamanho e Complexidade - Parte 1**: WMC, corresponde às métricas LOC, totalMethodsQty, staticMethodsQty, publicMethodsQty, abstractMethodsQty, finalMethodsQty, synchronizedMethodsQty.
- **Tamanho e Complexidade - Parte 2**: corresponde às métricas totalFieldsQty, publicFieldsQty, finalFieldsQty, staticFieldsQty, synchronizedFieldsQty, anonymousClassesQty, innerClassesQty.
- **Tamanho e Complexidade - Parte 3**: corresponde às métricas numbersQty, assignmentsQty, stringLiteralsQty, variablesQty, lambdasQty, uniqueWordsQty, logStatementsQty, modifiers.
- **Tamanho e Complexidade - Parte 4**: corresponde às métricas returnQty, loopQty, comparisonsQty, tryCatchQty, parenthesizedExpsQty, mathOperationsQty, maxNestedBlocksQty.

Os valores obtidos foram comparados com os intervalos de referência apresentados por Gonçalves (2022) (Tabela 4, Anexo A), classificados nas seguintes categorias:

- **Bom/Frequente**: faixa de valores típica de softwares profissionais, indicando estrutura alinhada às boas práticas.
- **Regular/Ocasional**: faixa de valores intermediários, aceitável, mas que merece atenção por se afastar do padrão mais comum.
- **Ruim/Raro**: faixa de valores fora dos intervalos de referência, que indicam qualidade estrutural abaixo da esperada.

Essa classificação permitiu avaliar a qualidade estrutural dos códigos gerados pelas ferramentas de IAG em relação a parâmetros consolidados na literatura. As definições detalhadas das principais métricas utilizadas (Tabela 3) e seus respectivos valores de referência (Tabela 4) são apresentados no Anexo A.

4.7 Análise dos Dados

A análise dos dados foi conduzida de modo a responder, de forma articulada, às três questões de pesquisa apresentadas na Introdução. Para isso, combinaram-se três dimensões: validação funcional, avaliação estrutural e análise temporal.

Inicialmente, classificou-se cada execução quanto à completude funcional (completo, parcial ou falha), por sistema, ferramenta e execução, sintetizando-se os resultados em tabelas e gráficos. Essa etapa fornece a base para as comparações subsequentes, pois delimita quais artefatos podem ser considerados entregas plenamente funcionais.

A avaliação da qualidade estrutural dos códigos gerados pelas ferramentas, quando conduzida isoladamente, apresenta uma limitação metodológica relevante: desconsidera a frequência com que cada ferramenta efetivamente entrega um software funcionalmente completo. Uma ferramenta pode obter indicadores estruturais elevados simplesmente por gerar poucos sistemas completos e, em geral, justamente os mais simples, ao passo que outra, ao entregar implementações completas em uma maior diversidade de sistemas, pode exibir métricas aparentemente inferiores em decorrência da maior complexidade dos códigos efetivamente produzidos. Para mitigar esse viés e oferecer um critério de comparação mais conservador entre as ferramentas avaliadas, propõe-se, neste trabalho, um índice denominado Qualidade Efetiva, que combina, em um único valor, a qualidade estrutural dos códigos completos e a confiabilidade de entrega de cada ferramenta. O índice é definido como:

$$\text{Qualidade Efetiva} = \text{Taxa de Completude} \times \text{Percentual de Métricas Boas}$$

em que:

- Taxa de Completude corresponde à razão entre o número de tentativas em que a ferramenta produziu código funcionalmente completo e o total de tentativas realizadas (oito sistemas × três execuções, totalizando 24 tentativas por ferramenta);
- Percentual de Métricas Boas representa a proporção média de métricas classificadas como “Bom” entre os códigos completos gerados pela ferramenta, obtida por meio do procedimento de média de médias por aspecto de qualidade descrito adiante.

O procedimento adotado para o cálculo do Percentual de Métricas Boas, empregado em todos os gráficos e indicadores agregados deste trabalho, consiste em duas etapas. Inicialmente, computa-se a média do percentual “Bom” dentro de cada um dos cinco aspectos de qualidade considerados — Coesão, Acoplamento, Herança, Ocultação de Informação e Tamanho e Complexidade. Em seguida, calcula-se a média aritmética desses cinco aspectos de qualidade. Esse procedimento atribui peso equivalente a cada aspecto de qualidade, evitando que aqueles com maior quantidade de métricas — sobretudo as subdivisões de Tamanho e Complexidade — predominem na estatística agregada e mascarem fragilidades em aspectos com menor número de métricas, como a Coesão. Dessa forma, preserva-se a representatividade de todas as dimensões estruturais avaliadas.

A título de ilustração, considere uma ferramenta hipotética que produz código com 80% das métricas classificadas como “Bom”, mas que entrega código completo em apenas 9 das 24 tentativas, o que corresponde a uma Taxa de Completude de 37,5%. Sua Qualidade Efetiva resultaria em $0{,}375 \times 80\% = 30\%$. Em contrapartida, uma ferramenta com 60% de métricas “Bom” e Taxa de Completude de 90% (22 das 24 tentativas) apresentaria Qualidade Efetiva de $0{,}90 \times 60\% = 54\%$. Embora a primeira ferramenta aparente resultado superior sob uma análise estritamente estrutural, é a segunda que oferece o melhor compromisso entre qualidade arquitetural e confiabilidade prática. O índice penaliza, portanto, tanto as ferramentas que apresentam qualidade estrutural elevada acompanhada de baixa taxa de entrega quanto aquelas que combinam alta taxa de entrega com qualidade estrutural insuficiente. Trata-se de uma medida exploratória adequada para a comparação integrada entre as ferramentas no escopo deste estudo.

Embora útil para integrar a qualidade estrutural e a completude funcional em um único indicador, a Qualidade Efetiva é uma medida exploratória proposta neste trabalho. Como ainda não existem valores de referência ou estudos de validação desse índice na literatura, seus resultados devem ser interpretados como uma medida relativa para comparação entre as ferramentas avaliadas, e não como uma medida absoluta da qualidade de software.

A formulação multiplicativa do índice foi adotada para evitar que ferramentas com baixa taxa de completude fossem favorecidas apenas por apresentarem elevada qualidade estrutural nos poucos sistemas gerados de forma completa. Como consequência, ferramentas com capacidade de entrega reduzida recebem maior penalização no índice, ainda que apresentem bom desempenho estrutural nos casos bem-sucedidos. Para permitir uma análise mais abrangente dos resultados, este trabalho apresenta também, em paralelo, os percentuais condicionais de métricas classificadas como “Bom” (Qualidade Geral, sem ponderação), especialmente na Figura 12, possibilitando a interpretação separada da qualidade estrutural e da completude funcional.

A análise da RQ1 - *Qual o nível de qualidade estrutural dos softwares gerados pelas ferramentas?* baseou-se na distribuição das métricas de software nas faixas Bom/Frequente, Regular/Ocasional e Ruim/Raro, considerando-se exclusivamente os softwares gerados de forma completa e funcional. Dessa forma, evita-se que códigos parciais, simplificados ou baseados em dados fictícios distorçam a avaliação estrutural. Complementarmente, aplicou-se o índice de Qualidade Efetiva, descrito anteriormente nesta seção, para articular a qualidade estrutural com a taxa de completude de cada ferramenta, oferecendo uma comparação que considera, simultaneamente, a qualidade dos códigos completos e a capacidade da ferramenta de produzi-los.

A RQ2 - *Ferramentas especializadas em geração de código produzem softwares estruturalmente superiores aos produzidos por LLMs generalistas?* foi analisada agrupando-se as ferramentas nessas duas categorias e comparando-se a Qualidade Efetiva de cada grupo. Como análise complementar, a mesma comparação foi conduzida entre as versões gratuitas e pagas.

Por fim, a RQ3 - *As ferramentas apresentam estabilidade temporal em sua Qualidade Efetiva ao longo das diferentes execuções experimentais?* foi avaliada por meio da comparação da Qualidade Efetiva entre as três execuções experimentais. Como esse índice incorpora simultaneamente a qualidade estrutural dos códigos completos e a taxa de completude funcional, a análise temporal considera de forma integrada aspectos estruturais e funcionais do resultado das ferramentas.

Cabe destacar que este trabalho adota uma abordagem descritiva e exploratória, baseando-se em percentuais, médias e gráficos comparativos. Não foram aplicados testes estatísticos inferenciais (como qui-quadrado, ANOVA ou testes não paramétricos) para verificar se as diferenças observadas entre ferramentas, entre categorias (generalistas × especializadas, gratuitas × pagas) ou entre execuções temporais são estatisticamente significativas. Essa decisão deve-se principalmente ao tamanho amostral relativamente reduzido por ferramenta (24 tentativas) e ao fato de que as métricas avaliadas não são independentes entre si (várias métricas operam sobre as mesmas classes). Em consequência, afirmações como "o AWS CodeWhisperer é estruturalmente melhor que o DeepSeek" ou "ferramentas especializadas diferem significativamente das generalistas" devem ser lidas, ao longo deste trabalho, como observações descritivas no escopo dos sistemas e execuções avaliados, e não como inferências populacionais.

Os resultados das três dimensões foram, então, interpretados de forma integrada, buscando-se identificar relações entre completude funcional, qualidade estrutural e estabilidade temporal, e evitando conclusões baseadas em uma única dimensão.

5 RESULTADOS

Esta seção apresenta os resultados do trabalho, organizados em cinco partes. A primeira trata da avaliação funcional dos sistemas gerados, identificando quais execuções resultaram em código completo, parcial ou com falha. A segunda examina a qualidade estrutural dos softwares, considerando apenas os artefatos completos e funcionais. A terceira analisa a estabilidade estrutural das ferramentas ao longo das três execuções temporais. A quarta detalha as métricas de software por aspecto de qualidade (acoplamento, coesão, herança, ocultação de informação e tamanho e complexidade). Por fim, a quinta discute os resultados de forma integrada, respondendo às questões de pesquisa.

Para responder à questão central do trabalho — se as ferramentas de IAG geram código com boa qualidade estrutural — a avaliação estrutural considera apenas os softwares gerados de forma completa e funcional. Essa delimitação é necessária porque códigos parciais, simplificados ou baseados em dados fictícios tendem a apresentar bons indicadores estruturais justamente por implementarem menos regras de negócio, o que poderia distorcer a comparação entre as ferramentas.

5.1 Avaliação do Funcionamento dos Sistemas Gerados

Conforme descrito na Seção 4.4, cada ferramenta foi avaliada em três execuções independentes para cada um dos oito sistemas Java, sendo a segunda execução realizada 15 dias após a primeira e a terceira 21 dias após a execução inicial.

A Tabela 2 apresenta o resultado do funcionamento das ferramentas em cada sistema. A notação utilizada indica a sequência das três execuções, em que **C** representa funcionamento completo, **P** funcionamento parcial e **F** falha. Assim, por exemplo, a sequência P/C/C indica que a primeira execução foi parcial e as duas seguintes foram completas.

Tabela 2 – Resultado do funcionamento dos sistemas gerados pelas ferramentas nas três execuções.

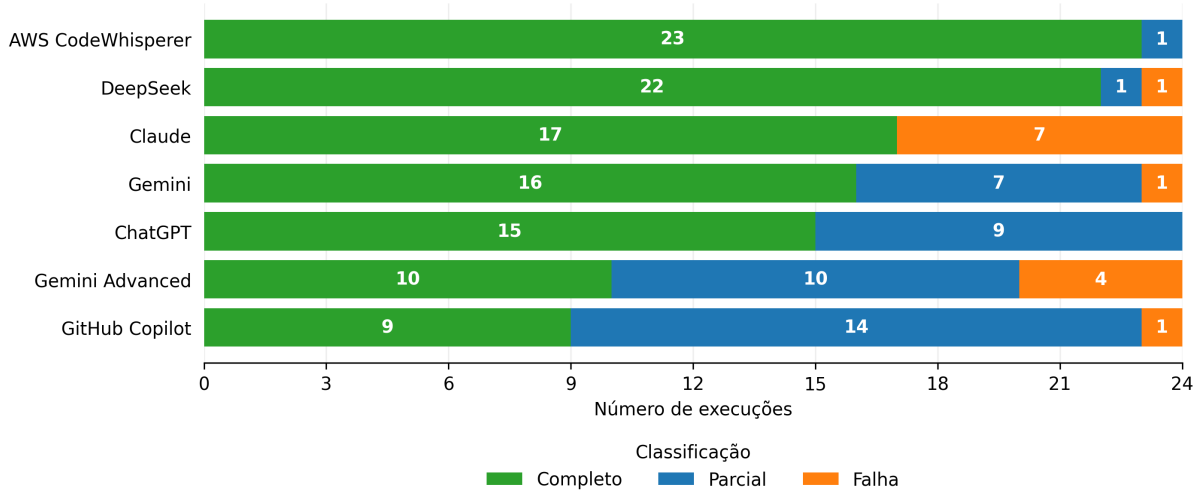
Sistema	Claude	AWS CW	ChatGPT	Gemini	Gemini Adv.	DeepSeek	Copilot
Estacionamento	C/C/C	C/C/C	C/C/C	P/C/C	C/C/C	C/C/C	C/C/C
Biblioteca	F/C/C	P/C/C	P/P/P	C/P/P	P/P/P	C/C/C	F/P/P
RPG	C/C/C	C/C/C	C/C/C	C/C/C	C/C/C	C/C/C	C/C/C
Banco	C/C/C	C/C/C	C/C/C	P/C/C	C/P/P	C/C/C	P/P/P
Restaurante	C/F/C	C/C/C	P/P/P	P/P/C	P/C/P	C/C/F	P/P/P
Ger. Projetos	F/C/C	C/C/C	P/P/C	C/C/F	F/F/P	C/C/C	P/P/P
Finanças	F/F/C	C/C/C	C/C/C	C/C/C	C/P/F	C/C/C	P/P/C
Estoque	F/F/C	C/C/C	C/P/C	C/C/P	F/C/P	C/P/C	P/C/C

Legenda: C = completo; P = parcial; F = falha.
Fonte: elaborado pela autora (2026).

Os dados da Tabela 2 evidenciam que a completude funcional variou entre ferramentas, sistemas e execuções. O AWS CodeWhisperer e o DeepSeek apresentaram maior regularidade no atendimento aos requisitos funcionais, com predominância de resultados completos. O ChatGPT apresentou funcionamento completo em alguns sistemas, como Estacionamento, RPG, Banco e Finanças, mas também mostrou funcionamento parcial recorrente em Biblioteca, Restaurante e Gerenciamento de Projetos. O GitHub Copilot apresentou elevada incidência de resultados parciais, especialmente em sistemas que exigiam persistência ou implementação mais completa das regras de negócio.

A Figura 2 apresenta a síntese quantitativa das classificações funcionais por ferramenta, consolidando os resultados da Tabela 2. Cada barra corresponde às 24 execuções de uma ferramenta (oito sistemas avaliados em três execuções), com seus segmentos indicando a quantidade de execuções classificadas como completo, parcial ou falha.

Figura 2 – Distribuição das classificações funcionais dos sistemas gerados pelas ferramentas avaliadas.



Cada barra representa 24 execuções por ferramenta, correspondentes a oito sistemas avaliados em três execuções. Os segmentos indicam a quantidade de execuções classificadas como completo, parcial ou falha.
Fonte: elaborado pela autora (2026).

Os resultados sintetizados na Figura 2 mostram que o AWS CodeWhisperer e o DeepSeek destacaram-se como as ferramentas que tiveram maior confiabilidade funcional

no experimento, com 23 e 22 execuções completas, respectivamente, de um total de 24 possíveis, enquanto o GitHub Copilot e o Gemini Advanced concentraram a maior proporção de parciais (14 e 10, respectivamente), sobretudo em sistemas que exigiam persistência de dados ou regras de negócio mais elaboradas. Dois comportamentos merecem destaque: o Claude não registrou nenhuma execução parcial, distribuindo suas 24 execuções entre completas (17) e falhas (7); e o Gemini Advanced apresentou a maior oscilação entre execuções de um mesmo sistema, chegando, no sistema de Finanças, a resultar nas três categorias (completa, parcial e falha). Esses resultados reforçam que a avaliação do funcionamento é essencial para a correta interpretação dos resultados estruturais, pois a geração de código compilável ou estruturalmente simples não garante o atendimento integral dos requisitos.

Observou-se ainda comportamento temporal heterogêneo entre as ferramentas: algumas (como o Claude.ai) apresentaram trajetória crescente ao longo das três execuções, enquanto outras (como o Gemini Advanced) apresentaram queda. Como o experimento foi conduzido em um intervalo relativamente curto (21 dias entre a primeira e a terceira execução), a atribuição dessas variações a mudanças efetivas nas plataformas deve ser feita com cautela. Entre as explicações plausíveis estão: (i) atualizações dos modelos subjacentes realizadas durante o período experimental, as quais podem ocorrer sem comunicação explícita aos usuários e produzir efeitos positivos ou negativos em tarefas específicas; (ii) a variabilidade inerente às LLMs, que podem gerar respostas distintas para um mesmo prompt mesmo na ausência de mudanças de versão, especialmente em tarefas mais complexas; e (iii) alterações na infraestrutura ou nos mecanismos internos de seleção de modelos empregados por algumas plataformas, que podem resultar em diferenças de comportamento ao longo do tempo. Considerando o número de execuções e a janela temporal adotados neste estudo, não foi possível distinguir empiricamente qual desses fatores exerceu maior influência sobre os resultados observados.

Quanto às categorias de ferramentas, a avaliação não revelou superioridade das especializadas em código (GitHub Copilot, Claude e AWS CodeWhisperer) sobre as generalistas (ChatGPT, Gemini, Gemini Advanced e DeepSeek): cada grupo reuniu ferramentas que tiveram maior confiabilidade nos sistemas (AWS CodeWhisperer e DeepSeek) e de baixa (GitHub Copilot e Gemini Advanced). Tampouco o acesso pago se mostrou vantajoso: na mesma família de modelos, o Gemini (gratuito) superou o Gemini Advanced (pago), com 16 execuções completas contra 9. Ambos os aspectos são retomados na discussão geral dos resultados.

5.2 Análise das Métricas de Software por Aspecto de Qualidade

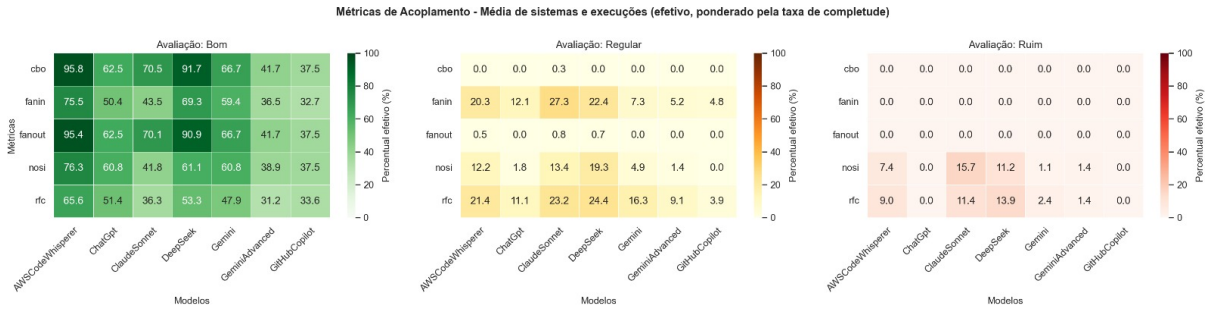
As Figuras 3 a 10 apresentam a análise das métricas de software por aspecto de qualidade, contemplando acoplamento, coesão, herança, ocultação de informação e tamanho e complexidade. Em todos os gráficos, os dados representam a média percentual das classes geradas em cada faixa (Bom, Regular ou Ruim), agregando os oito sistemas avaliados e as três execuções. Os percentuais exibidos correspondem aos valores de Qualidade Efetiva, conforme definidos na Seção 4.7, ou seja, são ponderados pela taxa de completude de cada ferramenta. Dessa forma, ferramentas que entregaram menor proporção de código funcionalmente completo têm seus percentuais proporcionalmente reduzidos, refletindo o compromisso entre qualidade estrutural e capacidade efetiva de entrega.

5.2.1 Acoplamento

As métricas de acoplamento avaliam o grau de dependência entre as classes do sistema. Um acoplamento elevado pode dificultar a manutenção e ampliar o risco de efeitos colaterais durante alterações nos sistemas.

A Figura 3 apresenta a distribuição percentual das classificações Bom/Frequente, Regular/Ocasional e Ruim/Raro para as métricas de acoplamento, agregadas por ferramenta de Inteligência Artificial Generativa. As métricas analisadas incluem CBO, FAN-IN, FAN-OUT, RFC e NOSI, sendo os percentuais calculados com base na proporção de classes enquadradas em cada faixa dos valores de referência adotados. Esse mesmo padrão de apresentação é utilizado nas Figuras 4 a 10 para as demais categorias de métricas estruturais.

Figura 3 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Acoplamento, por IAG.



Fonte: elaborado pela autora (2026).

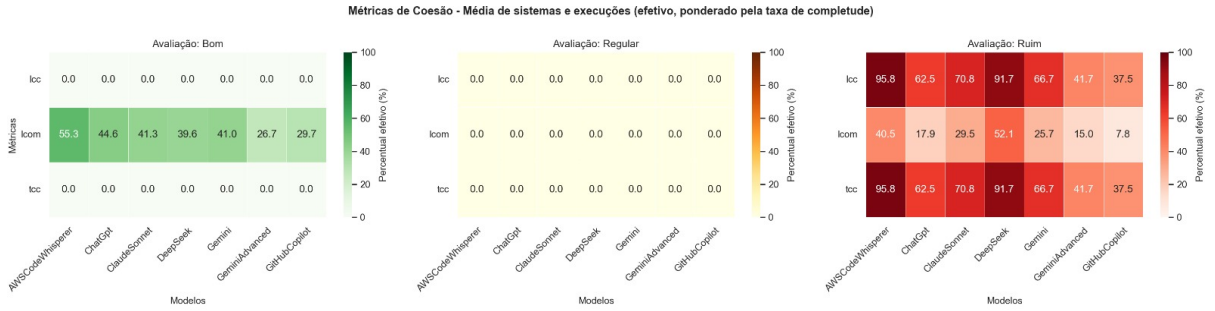
A análise das métricas de acoplamento indica que as ferramentas avaliadas produziram classes com baixo grau de dependência direta entre si, característica favorável à manutenção do software. As métricas CBO e FAN-OUT apresentaram classificação “Bom” em praticamente todas as classes dos códigos completos, indicando que as classes geradas referenciam e invocam poucas outras classes do sistema, característica que reflete baixo acoplamento e favorece a manutenção e a evolução do software.

Já as métricas RFC, NOSI e FAN-IN apresentaram maior dispersão entre as faixas de classificação. Em RFC, o AWS CodeWhisperer alcançou Qualidade Efetiva de 65,6% e o DeepSeek, 53,3%, enquanto Claude.ai e GitHub Copilot ficaram abaixo de 37%. Comportamento semelhante foi observado em NOSI, com o AWS CodeWhisperer atingindo 76,3% e o Claude.ai apenas 41,8%. Esses resultados revelam que algumas ferramentas produzem classes com maior número de métodos invocáveis e chamadas estáticas, o que pode dificultar sua compreensão isolada. De modo geral, o acoplamento mostrou-se baixo, embora existam classes pontuais que assumem papel central no fluxo de chamadas do sistema, sobretudo nos códigos produzidos por DeepSeek e Claude.ai.

5.2.2 Coesão

Coesão avalia o grau de relação entre os métodos e atributos de uma mesma classe. Em sistemas orientados a objetos, espera-se que classes apresentem alta coesão, concentrando responsabilidades bem definidas.

Figura 4 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Coesão, por IAG.



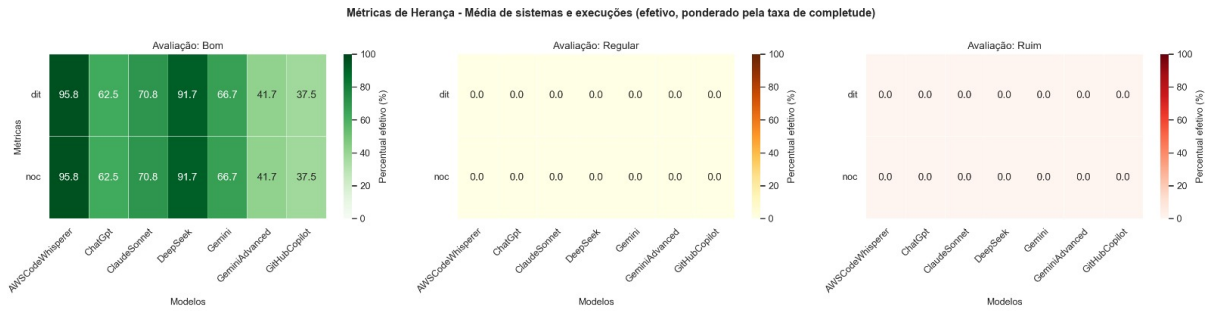
Fonte: elaborado pela autora (2026).

Os resultados de coesão evidenciam a limitação estrutural mais expressiva observada nos códigos gerados pelas LLMs. As métricas LCC e TCC apresentaram classificação "Ruim" em todas as classes dos códigos completos, indicando ausência quase total de pares de métodos que compartilhem atributos ou se invoquem mutuamente. A métrica LCOM foi a única a apresentar variação relevante entre as ferramentas, com Qualidade Efetiva de 55,3% no AWS CodeWhisperer, 44,6% no ChatGPT e valores inferiores a 42% nas demais. Esses resultados reforçam que a geração automática tende a produzir classes pouco coesas, que acumulam múltiplas responsabilidades e operam de forma mais procedimental do que efetivamente orientada a objetos. O padrão de baixa coesão persistiu em todas as ferramentas avaliadas, configurando-se como uma característica estrutural intrínseca aos modelos.

5.2.3 Herança

As métricas de herança permitem avaliar o uso de hierarquias de classes nos softwares orientados a objetos, envolvendo a profundidade da árvore de herança e número de subclasses.

Figura 5 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Herança, por IAG.



Fonte: elaborado pela autora (2026).

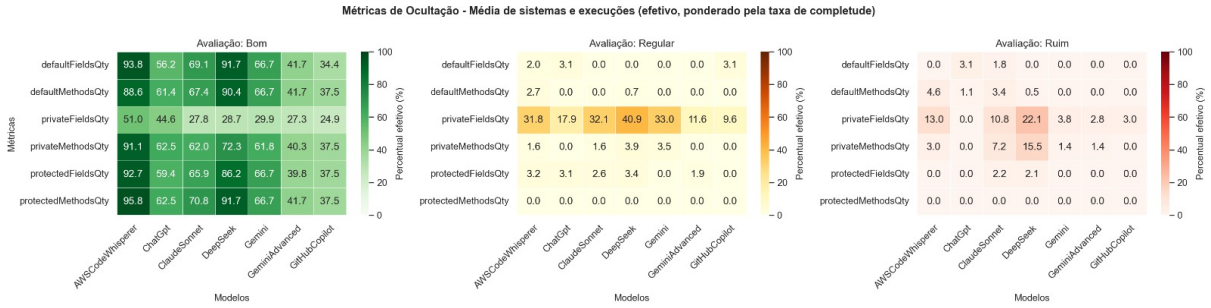
A análise das métricas de herança indica que as ferramentas avaliadas tendem a produzir arquiteturas planas, com pouca utilização de hierarquias de classes. As métricas DIT e NOC apresentaram classificação "Bom" em todas as classes dos códigos completos, sem ocorrências nas faixas "Regular" ou "Ruim". Isso indica que o uso de herança nos softwares gerados pelas ferramentas seguem, em geral, práticas similiares àquelas empregadas por

desenvolvedores de software. Nesse aspecto, a herança é empregada com parcimônia o que pode contribuir para manutenibilidade dos sistemas, pois árvores de herança profundas demandam maior esforço de compreensão do comportamento dos objetos das classes envolvidas.

5.2.4 Ocultação de Informação

A ocultação de informação está relacionada ao encapsulamento e ao controle da visibilidade de atributos e métodos. Em sistemas orientados a objetos, espera-se que a exposição interna das classes seja reduzida, favorecendo manutenção e proteção da estrutura interna.

Figura 6 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Ocultação de Informação, por IAG.



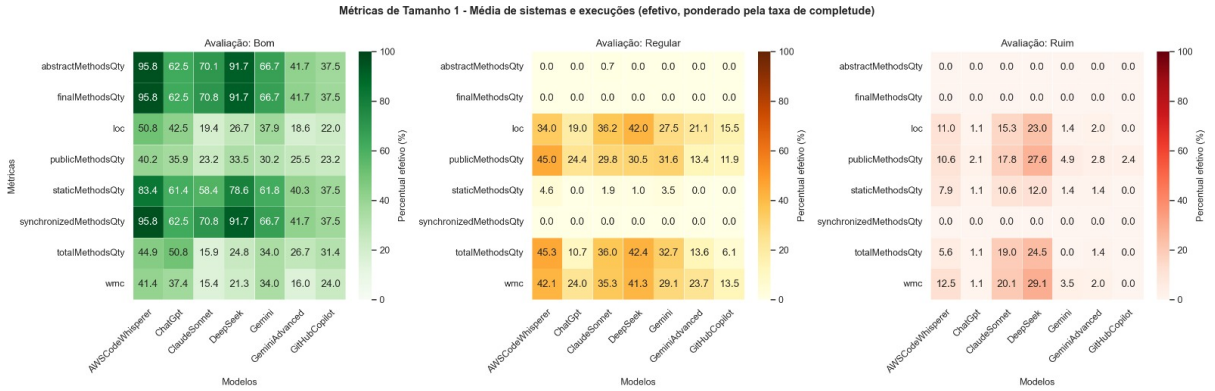
Fonte: elaborado pela autora (2026).

A análise das métricas de ocultação de informação revela um padrão misto. As métricas defaultFieldsQty, defaultMethodsQty, privateMethodsQty, protectedFieldsQty e protectedMethodsQty apresentaram classificação "Bom" em praticamente todas as classes dos códigos completos, indicando uso adequado dos modificadores de acesso. A exceção foi a métrica privateFieldsQty, que mede a quantidade de atributos privados por classe e cujos valores indicam classes que acumulam muitos dados internos. Essa métrica apresentou parcela expressiva de classes nas faixas "Regular" e "Ruim" em todas as ferramentas, com destaque para o DeepSeek, que concentrou 40,9% das classes em "Regular" e 22,1% em "Ruim". O excesso de atributos privados em uma mesma classe reforça o padrão já observado nas métricas de coesão: quanto mais atributos uma classe possui, maior a chance de que seus métodos atuem sobre subconjuntos distintos desses atributos, reduzindo a colaboração entre eles. Esse comportamento sugere que a geração automática frequentemente prioriza uma implementação direta dos requisitos, concentrando responsabilidades em classes centrais, em vez de promover uma decomposição arquitetural mais refinada.

5.2.5 Tamanho e Complexidade

As métricas de tamanho e complexidade avaliam aspectos como quantidade de linhas de código, métodos, atributos, variáveis, retornos, laços, comparações e blocos aninhados. Essas métricas são relevantes porque sistemas mais extensos e complexos tendem a demandar maior esforço de manutenção. As Figuras 7 a 10 apresentam a distribuição percentual das classificações para as quatro partes em que essas métricas foram organizadas.

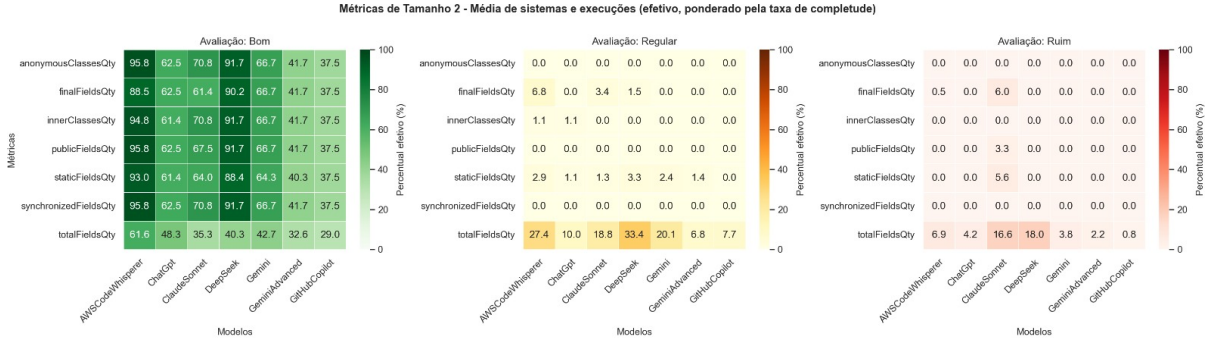
Figura 7 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Tamanho e Complexidade — Parte 1, por IAG.



Fonte: elaborado pela autora (2026).

A primeira parte das métricas de tamanho e complexidade revela um contraste relevante. As métricas abstractMethodsQty, finalMethodsQty, staticMethodsQty e synchronizedMethodsQty contam, respectivamente, o número de métodos abstratos, finais, estáticos e sincronizados por classe, sendo recursos cuja utilização excessiva pode comprometer a flexibilidade, a testabilidade e a manutenibilidade do código. Essas métricas apresentaram classificação “Bom” em praticamente todas as classes dos códigos completos, indicando que as ferramentas recorrem a esses recursos de forma comedida. Em contrapartida, as métricas LOC, WMC, totalMethodsQty e publicMethodsQty apresentaram parcelas expressivas nas faixas “Regular” e “Ruim” em todas as ferramentas. Em LOC, que mede o número de linhas de código por classe, mais de 30% das classes ficaram na faixa “Regular” em quase todas as ferramentas, com até 42,0% no DeepSeek. Em WMC, que mede a complexidade ciclomática acumulada dos métodos de uma classe, valores semelhantes foram observados, indicando classes longas e internamente complexas. Em totalMethodsQty e publicMethodsQty, o padrão se repete, sugerindo que as ferramentas tendem a produzir classes com muitos métodos, especialmente públicos.

Figura 8 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Tamanho e Complexidade — Parte 2, por IAG.

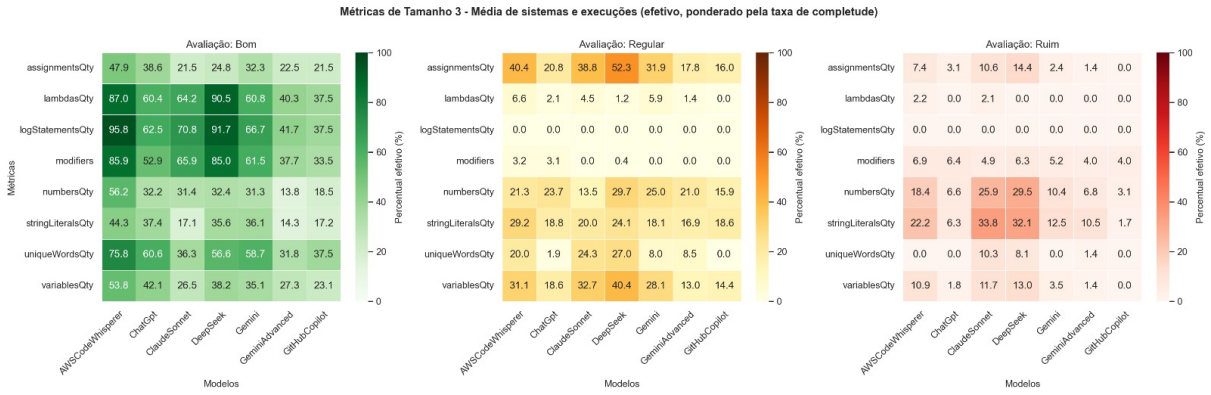


Fonte: elaborado pela autora (2026).

A segunda parte das métricas concentra indicadores ligados a atributos e estruturas internas das classes. A maior parte dessas métricas: anonymousClassesQty, innerClassesQty, finalFieldsQty, publicFieldsQty, staticFieldsQty e synchronizedFieldsQty, apresentou classificação “Bom” em praticamente todas as classes dos códigos completos, confirmando que

recursos como classes internas, classes anônimas e atributos com modificadores especiais são pouco utilizados. A exceção foi a métrica totalFieldsQty, que mede o número total de atributos por classe e apresentou parcela significativa de classes nas faixas “Regular” (até 33,4% no DeepSeek) e “Ruim” (até 18,0% no DeepSeek). Esse resultado complementa o já observado em privateFieldsQty, reforçando que as ferramentas tendem a produzir classes que acumulam muitos atributos, e não apenas atributos privados em excesso.

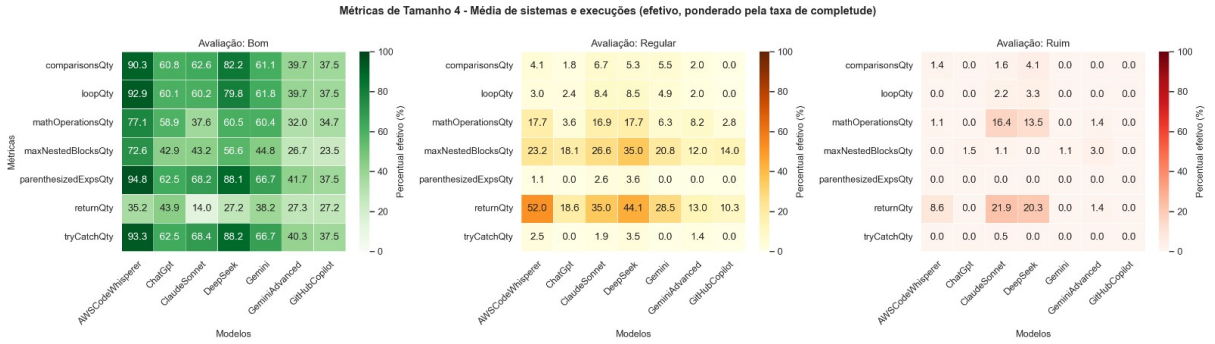
Figura 9 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Tamanho e Complexidade — Parte 3, por IAG.



Fonte: elaborado pela autora (2026).

A terceira parte das métricas avalia construções internas relacionadas à expressividade e ao vocabulário do código. As métricas logStatementsQty, lambdasQty e modifiers apresentaram classificação “Bom” em praticamente todas as classes, indicando uso comedido de recursos como logs, expressões lambda e modificadores. Em contrapartida, as métricas assignmentsQty, numbersQty, stringLiteralsQty, variablesQty e uniqueWordsQty apresentaram parcelas relevantes nas faixas “Regular” e “Ruim”, revelando classes com elevada densidade de operações internas e vocabulário pouco enxuto. A métrica assignmentsQty, que mede o número de atribuições por classe, ilustra esse padrão, com destaque para o DeepSeek, em que 52,3% das classes ultrapassaram o limite considerado típico para código profissional. Comportamento semelhante foi observado em stringLiteralsQty e numbersQty, que contam, respectivamente, strings e números embutidos diretamente no código: até 33,8% das classes do Claude.ai e 32,1% das do DeepSeek apresentaram valores na faixa “Ruim”. Esse resultado sugere que as ferramentas embutem grande quantidade de valores literais no código, padrão comumente associado à ausência de constantes nomeadas ou de separação adequada entre lógica e dados.

Figura 10 – Distribuição percentual das classificações Bom, Regular e Ruim para as métricas do aspecto de qualidade Tamanho e Complexidade — Parte 4, por IAG.



Fonte: elaborado pela autora (2026).

A quarta parte das métricas concentra-se em construções de controle de fluxo e operações lógicas. As métricas loopQty, comparisonsQty, tryCatchQty e parenthesizedExpsQty apresentaram classificação "Bom" em praticamente todas as classes dos códigos completos, indicando que laços, comparações e blocos try/catch são utilizados de forma comedida. As métricas mathOperationsQty e maxNestedBlocksQty apresentaram comportamento intermediário, com proporções relevantes na faixa "Regular" (até 35,0% em maxNestedBlocksQty no DeepSeek). A métrica mais crítica dessa parte foi returnQty, que mede o número de pontos de saída em métodos: parcela expressiva das classes ficou na faixa "Regular" em todas as ferramentas, com até 52,0% no AWS CodeWhisperer e 44,1% no DeepSeek, indicando métodos com múltiplos pontos de retorno — prática que tende a comprometer a legibilidade e a previsibilidade do fluxo de execução.

A análise conjunta das quatro partes das métricas de tamanho e complexidade revela um padrão consistente: as ferramentas avaliadas evitam construções sintáticas complexas, como laços profundos, blocos try/catch excessivos, classes internas e expressões lambda em excesso. Por outro lado, as métricas LOC, WMC, totalMethodsQty, totalFieldsQty, assignmentsQty e returnQty indicam, de forma recorrente, classes longas, com muitos atributos, muitas atribuições e métodos com múltiplos pontos de saída. Esse resultado dialoga com os achados sobre coesão e ocultação de informação: ao concentrar responsabilidades e atributos em poucas classes, a geração automática produz, como consequência, classes maiores e internamente mais complexas, ainda que com estrutura sintática simplificada.

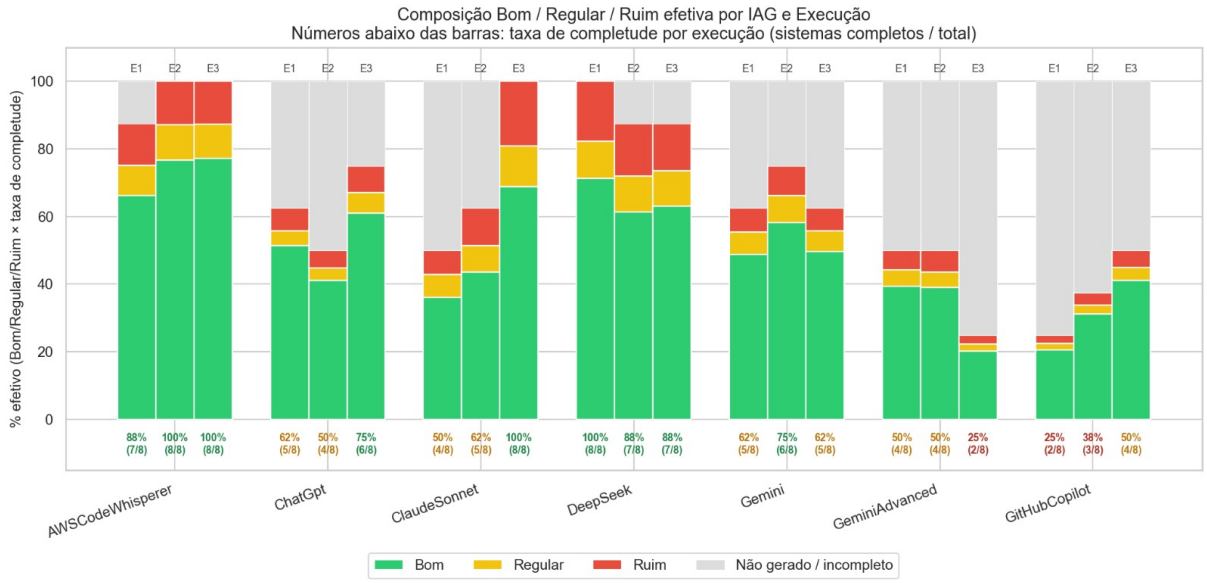
5.3 Qualidade Estrutural dos Softwares Completos

Após a avaliação funcional, os artefatos gerados foram analisados quanto à sua qualidade estrutural, por meio das métricas de software classificadas segundo os intervalos de referência derivados por Gonçalves (2022). As métricas foram agrupadas nas categorias Bom/Frequente, Regular/Ocasional e Ruim/Raro. Conforme justificado na introdução desta seção, a análise estrutural considera apenas os softwares gerados de forma completa e funcional, e os percentuais apresentados correspondem aos valores de Qualidade Efetiva, conforme definidos na Seção 4.7.

A Figura 11 apresenta a composição percentual efetiva das classificações Bom, Regular e Ruim das métricas de software, agregadas por ferramenta e por execução. A taxa de completude de cada execução, indicada abaixo das barras, é incorporada ao cálculo: as

três faixas coloridas representam a Qualidade Efetiva (Bom, Regular e Ruim ponderados pela taxa de completude), enquanto a faixa cinza no topo de cada barra representa a parcela das tentativas em que o código não foi gerado de forma completa.

Figura 11 – Composição percentual das classificações Bom, Regular e Ruim das métricas de software por IAG e execução.

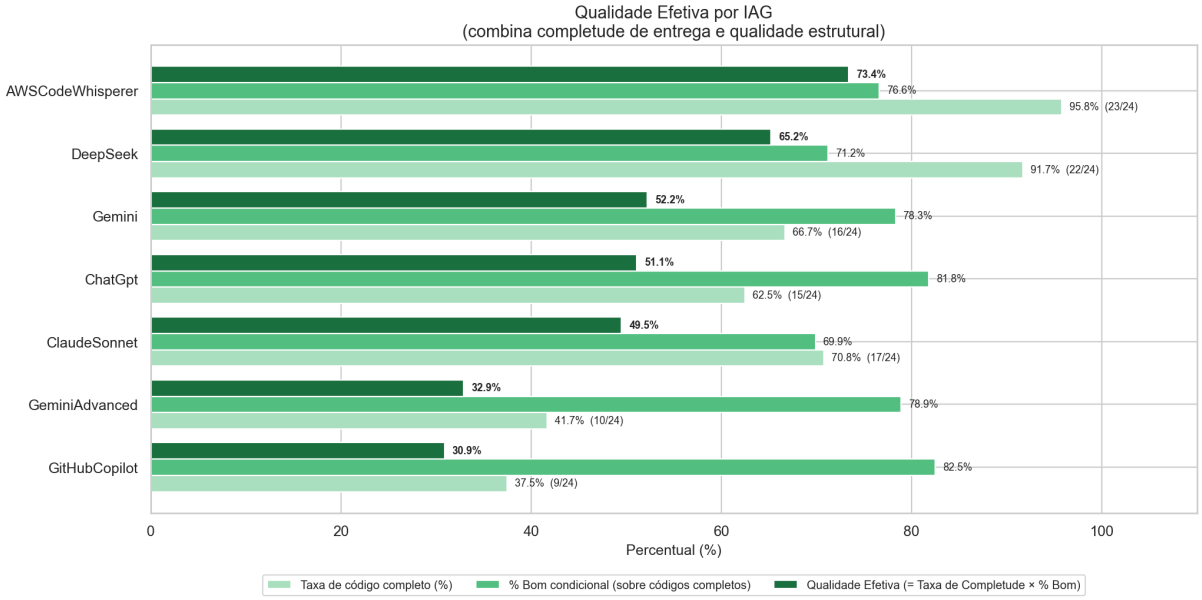


Composição percentual efetiva das classificações Bom/Frequente, Regular/Ocasional e Ruim/Raro das métricas de software, agregadas por IAG e por execução. Cada barra representa a Qualidade Efetiva (Bom + Regular + Ruim ponderados pela taxa de completude); a faixa cinza no topo indica a parcela das tentativas em que o código não foi gerado de forma completa. Os valores abaixo das barras indicam a taxa de completude da respectiva execução (sistemas completos / total).
Fonte: elaborado pela autora (2026).

A análise da Figura 11 indica que, quando se considera a Qualidade Efetiva, as ferramentas avaliadas apresentam resultado heterogêneo. O AWS CodeWhisperer e o DeepSeek destacaram-se pelos maiores percentuais de Bom efetivo, refletindo a combinação de alta taxa de completude com qualidade estrutural consistente ao longo das três execuções. O ChatGPT, o Claude.ai e o Gemini mantiveram-se em patamar intermediário, com variações entre execuções associadas, sobretudo, a oscilações na taxa de completude. Em contrapartida, o Gemini Advanced e o GitHub Copilot registraram os menores percentuais de Bom efetivo, indicando o impacto direto da baixa taxa de entrega sobre a Qualidade Efetiva. Esses resultados reforçam que a Qualidade Efetiva permite distinguir o resultado das ferramentas de forma mais discriminativa do que a análise estrutural isolada, uma vez que penaliza simultaneamente a baixa taxa de entrega e as fragilidades estruturais dos códigos completos.

A Figura 12 sintetiza o resultado da avaliação global das ferramentas a partir das três dimensões que compõem o índice de Qualidade Efetiva: a taxa de código completo, o percentual condicional de métricas "Bom" (calculado sobre os códigos completos) e a Qualidade Efetiva resultante. Na figura, as ferramentas estão ordenadas, de cima para baixo, pelo valor de Qualidade Efetiva.

Figura 12 – Qualidade Efetiva por IAG: combinação entre completude de entrega e qualidade estrutural



Cada ferramenta é representada por três barras: a barra em verde claro corresponde à taxa de código completo (proporção de tentativas que resultaram em código funcionalmente completo); a barra em verde intermediário corresponde ao percentual condicional de métricas "Bom" (calculado apenas sobre os códigos completos); e a barra em verde escuro corresponde à Qualidade Efetiva, definida como o produto entre os dois indicadores anteriores.

Fonte: elaborado pela autora (2026).

A Figura 12 compara a Qualidade Efetiva, que incorpora a taxa de completude funcional, com a Qualidade Geral, representada pelo percentual de métricas classificadas como "Bom" considerando apenas os códigos gerados de forma completa. Essa comparação permite distinguir a qualidade estrutural dos sistemas efetivamente produzidos da qualidade global das ferramentas, que também leva em conta sua capacidade de gerar sistemas completos. Assim, a análise conjunta das barras evidencia o impacto da completude funcional sobre a avaliação final de cada ferramenta.

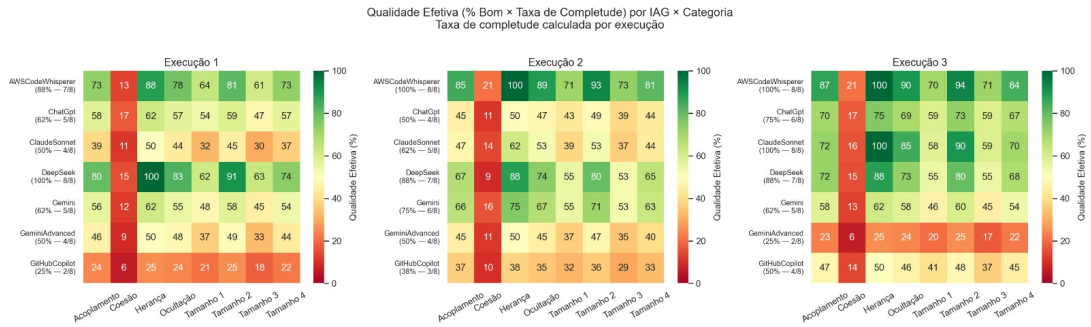
Esses resultados indicam que, sob a perspectiva da Qualidade Efetiva, as ferramentas avaliadas geram softwares com nível de qualidade heterogêneo, em contraste com a homogeneidade observada quando se considera apenas a qualidade estrutural dos códigos completos. O AWS CodeWhisperer e o DeepSeek apresentam os melhores resultados, com alta taxa de entrega de código completo e qualidade estrutural, com Qualidades Efetivas de 73,4% e 65,2%, respectivamente. O Claude.ai posicionou-se em patamar intermediário, com Qualidade Efetiva de 49,5%, resultado equilibrado entre taxa de completude e percentual condicional de "Bom". Vale destacar que, conforme observado na Seção 5.1, essa ferramenta apresentou um comportamento distinto das demais ao não registrar nenhuma execução parcial, distribuindo suas tentativas apenas entre completas e falhas. Ferramentas como o ChatGPT, o GitHub Copilot e o Gemini Advanced, embora apresentem percentuais condicionais de "Bom" comparáveis aos das melhores colocadas, são penalizadas pela menor taxa de geração de código completo, o que reduz expressivamente seu resultado final. Esse resultado evidencia que a completude é um fator que deve ser considerado ao se analisar a qualidade dos softwares gerados por IA.

5.4 Estabilidade Temporal das Ferramentas

A Seção 5.3 apresentou o resultado da análise global das ferramentas em termos de Qualidade Efetiva, sem distinguir o comportamento em cada um dos oito aspectos de qualidade. Esta seção aprofunda essa análise ao apresentar a Qualidade Efetiva por aspecto de qualidade em cada uma das três execuções, permitindo identificar em quais aspectos estruturais cada ferramenta apresenta maior ou menor estabilidade ao longo do tempo.

A Figura 13 apresenta a Qualidade Efetiva por aspecto de qualidade em cada uma das três execuções. Os mapas de calor, dispostos lado a lado, apresentam as ferramentas de LLMs nas linhas e os oito aspectos de qualidade nas colunas, com a taxa de completude de cada ferramenta indicada junto ao seu identificador.

Figura 13 – Qualidade Efetiva por IAG e aspecto de qualidade ao longo das três execuções.



Análise temporal da qualidade dos software gerados pelas ferramentas ao longo das três execuções experimentais, expressa pela Qualidade Efetiva (% Bom × Taxa de Completude) por aspecto de qualidade. Cada mapa de calor corresponde a uma execução, com as ferramentas nas linhas e os oito aspectos de qualidade nas colunas. A taxa de completude de cada ferramenta na respectiva execução é apresentada junto ao seu identificador (sistemas completos / total). Valores mais elevados, em tons de verde, indicam maior Qualidade Efetiva; valores mais baixos, em tons de vermelho, indicam menor Qualidade Efetiva.
Fonte: elaborado pela autora (2026).

A análise da Figura 13 revela que os mesmos aspectos de qualidade interna de software ocupam as posições de melhor e pior resultado nas três execuções, ainda que os valores absolutos variem em função da taxa de completude. Em todas as execuções, a coesão concentra os menores valores de Qualidade Efetiva, enquanto acoplamento e herança figuram entre os aspectos com melhor resultado, padrão que se repete para todas as ferramentas. Esse comportamento indica que as fragilidades estruturais observadas, sobretudo a baixa coesão, configuram-se como características persistentes dos modelos avaliados, ainda que os valores absolutos possam apresentar variações relevantes entre execuções, tanto para mais quanto para menos.

No que se refere aos comportamentos temporais individuais das ferramentas, observam-se três padrões distintos. O AWS CodeWhisperer e o DeepSeek apresentaram alta estabilidade, com pequenas variações entre execuções. O Claude.ai apresentou o comportamento crescente mais expressivo: na primeira execução, registrou apenas 39% em Acoplamento e 45% em Tamanho 2, mas evoluiu progressivamente até a terceira execução, em que a taxa de completude alcançou 100% e a Qualidade Efetiva subiu de forma significativa, com Acoplamento atingindo 72%, Herança 100% e Tamanho 2 alcançando o valor de 90%. O GitHub Copilot também apresentou trajetória crescente, embora mais

gradual. Em contrapartida, o Gemini Advanced apresentou comportamento decrescente, com queda expressiva na terceira execução, em que a taxa de completude reduziu a 25% e os valores de Qualidade Efetiva caíram a menos da metade dos observados nas execuções anteriores. O ChatGPT e o Gemini apresentaram oscilações, sem tendência clara.

Em conjunto, esses resultados indicam que a estabilidade temporal deve ser compreendida em duas dimensões complementares. A primeira refere-se ao à qualidade estrutural dos softwares gerados, que se manteve consistente. Isso indica que as forças e as fragilidades de cada modelo se preservam ao longo do tempo, configurando-se como características intrínsecas de cada ferramenta avaliada. A segunda dimensão refere-se à taxa de completude, que oscilou de forma relevante para algumas ferramentas, como evidencia o caso do Gemini Advanced, cuja queda na terceira execução reduziu sua Qualidade Efetiva à metade. Conclui-se, portanto, que uma ferramenta pode preservar a qualidade dos softwares gerados ao longo do tempo e, ainda assim, apresentar Qualidade Efetiva instável, caso sua capacidade de gerar código completo varie entre execuções.

Esse resultado responde à RQ3 ao demonstrar que a estabilidade temporal das ferramentas não depende apenas da preservação da qualidade estrutural dos softwares gerados, mas também da consistência na geração de códigos completos. Assim, ferramentas estruturalmente estáveis podem apresentar resultado global instável quando sua taxa de completude varia significativamente entre execuções.

5.5 Discussão Geral dos Resultados

Os resultados permitem responder às três questões de pesquisa.

Em relação à **RQ1** - *Qual o nível de qualidade estrutural dos softwares gerados pelas ferramentas?*, os dados indicam que, quando se considera apenas o percentual de métricas "Bom" sobre os códigos completos, as ferramentas avaliadas apresentam qualidade estrutural comparável, com predominância clara nos aspectos de qualidade de acoplamento e herança e com a coesão como principal limitação recorrente. No entanto, quando se incorpora a taxa de completude por meio do índice de Qualidade Efetiva proposto na Seção 4.7, o resultado das ferramentas torna-se heterogêneo: o AWS CodeWhisperer (73,4%) e o DeepSeek (65,2%) destacam-se das demais, ao passo que o Gemini Advanced (32,9%) e o GitHub Copilot (30,9%) ocupam as últimas posições, apesar de apresentarem percentuais condicionais de "Bom" elevados. Esses resultados evidenciam que a qualidade estrutural não pode ser dissociada da capacidade de gerar código completo, sob pena de produzir conclusões enviesadas a favor de ferramentas que entregam pouco.

No que se refere à **RQ2** - *Ferramentas especializadas em geração de código produzem softwares estruturalmente superiores aos produzidos por LLMs generalistas?*, os resultados não evidenciaram superioridade consistente das ferramentas especializadas (GitHub Copilot, AWS CodeWhisperer e Claude.ai) em relação às generalistas (ChatGPT, Gemini, Gemini Advanced e DeepSeek). Ambos os grupos reúnem ferramentas de alto resultado, AWS CodeWhisperer entre as especializadas e DeepSeek entre as generalistas, e ferramentas de baixo resultado, GitHub Copilot entre as especializadas e Gemini Advanced entre as generalistas. De forma semelhante, não se verificaram diferenças expressivas entre versões gratuitas e pagas: na mesma família de modelos, o Gemini (gratuito) superou o Gemini Advanced (pago), tanto em taxa de completude quanto em Qualidade Efetiva.

Por fim, em relação à **RQ3** - *As ferramentas apresentam estabilidade temporal em sua Qualidade Efetiva ao longo das diferentes execuções experimentais?*, a análise

temporal revelou duas dimensões distintas de estabilidade. Na primeira, observou-se que o resultado relativo entre os aspectos de qualidade mantém-se consistente nas três execuções para todas as ferramentas: a Coesão é sempre o aspecto de qualidade mais frágil, ao passo que Acoplamento e Herança são consistentemente as mais bem avaliados. Essas características configuram-se, portanto, como padrões intrínsecos dos modelos avaliados. Na segunda dimensão, observou-se que a taxa de completude pode variar de forma relevante entre execuções, como ilustram os casos do Claude.ai (crescente) e do Gemini Advanced (decrescente). Conclui-se, portanto, que uma ferramenta pode preservar suas características estruturais ao longo do tempo e, ainda assim, apresentar Qualidade Efetiva instável, caso sua capacidade de gerar código completo varie entre execuções.

Os resultados obtidos mostram convergência com estudos anteriores da literatura. A recorrência de problemas relacionados à coesão confirma os achados de Pereira (2025), que também identificou esse aspecto como a principal fragilidade estrutural em softwares Java gerados por IAG. De forma semelhante, os resultados observados corroboram as conclusões de Yetiştiren *et al.* (2023) e Sabra *et al.* (2025) , que apontaram a presença frequente de limitações arquiteturais e problemas de design em códigos produzidos por modelos generativos. Além disso, a variação observada na taxa de completude entre sistemas e execuções está alinhada às observações de Wang *et al.* (2025), segundo as quais o resultado das LLMs tende a deteriorar-se à medida que aumenta a complexidade das tarefas propostas. Dessa forma, os resultados deste estudo reforçam evidências já reportadas na literatura e ampliam seu escopo ao incorporar a análise temporal e o índice de Qualidade Efetiva.

Os resultados reforçam a necessidade de uma avaliação multidimensional que combine validação funcional, métricas estruturais e análise temporal. O índice de Qualidade Efetiva, proposto neste trabalho, permitiu evidenciar, no escopo das ferramentas e sistemas avaliados, diferenças que uma análise estrutural isolada não tornaria visíveis.

6 AMEAÇAS À VALIDADE

Este trabalho possui as seguintes limitações principais que devem ser consideradas na interpretação dos resultados e que apontam possibilidades para investigações futuras:

Número e periodicidade das execuções. Foram realizadas três execuções por combinação de ferramenta e sistema, em uma janela total de 21 dias. Esse delineamento foi adotado em função do cronograma do trabalho e da viabilidade da avaliação manual. Reconhece-se, contudo, que o número de execuções é reduzido para caracterizar com maior robustez a evolução temporal das plataformas, e que a estabilidade observada pode estar associada à curta janela analisada.

Variabilidade probabilística dos modelos. As oscilações observadas entre execuções podem refletir tanto mudanças nas plataformas quanto a variabilidade inerente aos LLMs, que podem produzir respostas distintas para o mesmo prompt mesmo sem alteração de versão. Com o desenho experimental adotado, não foi possível separar essas fontes de variação.

Prompts utilizados. Os enunciados foram simples e abertos, seguindo o protocolo de Pereira (2025), o que assegura comparabilidade entre estudos. Por outro lado, esse tipo de prompt amplia o espaço de decisão das ferramentas e pode tornar os resultados sensíveis a pequenas variações no enunciado. Assim, os achados devem ser interpretados como característicos do conjunto específico de prompts adotado, e não como propriedades

absolutas das ferramentas.

Avaliação funcional e coleta de erros. A classificação Completo/Parcial/Falha foi conduzida manualmente, com base nos requisitos dos enunciados, sem o uso de testes automatizados ou múltiplos avaliadores independentes. Além disso, completude funcional não equivale à ausência de bugs: sistemas classificados como completos podem conter falhas pontuais, como validações ausentes ou comportamentos inesperados em entradas específicas.

Ausência de análise estatística inferencial. A análise baseou-se em estatística descritiva, percentuais, médias e gráficos comparativos, sem aplicação de testes inferenciais. Dessa forma, as diferenças observadas entre ferramentas, categorias ou execuções devem ser interpretadas como observações descritivas restritas ao conjunto de sistemas e execuções avaliados, e não como inferências populacionais.

Índice de Qualidade Efetiva. O índice proposto constitui uma medida exploratória ainda não validada em estudos anteriores. Sua forma multiplicativa penaliza fortemente ferramentas com baixa taxa de completude, mesmo quando os poucos sistemas completos apresentam boa qualidade estrutural. Por isso, os resultados foram apresentados em conjunto com o percentual condicional de métricas classificadas como "Bom", permitindo distinguir qualidade estrutural e completude funcional.

Versões pagas. As versões pagas utilizadas, Gemini Advanced e GitHub Copilot, foram acessadas por meio de licenças acadêmicas ou estudantis quando disponíveis. Esse aspecto não compromete os resultados estruturais e funcionais observados, mas deve ser considerado em análises de custo-benefício.

Validação cruzada das métricas. A avaliação estrutural baseou-se exclusivamente na CK Tool e nos valores de referência derivados por Gonçalves (2022). Estudos futuros podem incorporar outras ferramentas de análise estática para verificar a estabilidade das classificações obtidas.

Arquitetura dos sistemas gerados. Todos os sistemas gerados apresentaram interface em modo caractere, executada via terminal. Portanto, os achados referem-se especificamente a softwares Java de console, não podendo ser generalizados diretamente para sistemas com interface gráfica, aplicações web ou arquiteturas baseadas em microsserviços.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho teve como objetivo avaliar a qualidade estrutural de softwares Java gerados por ferramentas de Inteligência Artificial Generativa, considerando modelos generalistas e assistentes especializados em geração de código. Para isso, foi conduzido um estudo empírico envolvendo sete ferramentas, oito sistemas e três execuções por sistema, combinando análise funcional, métricas de software e avaliação temporal.

Quanto ao nível de qualidade estrutural (RQ1), observou-se que, entre os softwares gerados de forma completa, as ferramentas avaliadas apresentaram padrões estruturais relativamente semelhantes, com predominância de classificações "Bom" nas métricas de acoplamento e herança e com a coesão como principal limitação recorrente. Entretanto, ao incorporar a taxa de completude por meio do índice de Qualidade Efetiva, emergem diferenças relevantes entre as ferramentas, destacando-se AWS CodeWhisperer e DeepSeek pelos melhores resultados globais. Não se observou superioridade consistente das ferramentas especializadas sobre as generalistas, nem entre versões gratuitas e pagas (RQ2). Quanto à

estabilidade temporal (RQ3), observaram-se duas dimensões distintas: o resultado relativo entre os aspectos de qualidade manteve-se consistente nas três execuções, configurando-se como característica intrínseca dos modelos, ao passo que a taxa de completude apresentou variações relevantes para algumas ferramentas, como ilustram os casos do Claude.ai (crescente) e do Gemini Advanced (decrescente).

Os resultados estruturais revelaram um quadro misto: embora as ferramentas demonstrem bom controle do acoplamento, persistem fragilidades arquiteturais relevantes, como baixa coesão, concentração de responsabilidades em poucas classes e presença expressiva de classes longas e internamente complexas. Soma-se a isso o fato de que a geração de código completo e funcional permanece um desafio não trivial: mesmo em sistemas de escopo relativamente delimitado, todas as ferramentas registraram ao menos uma execução parcial ou com falha, com destaque para GitHub Copilot e Gemini Advanced, que concentraram a maior proporção de entregas incompletas.

Diante desse panorama, a principal conclusão do estudo é que a qualidade estrutural e a capacidade de gerar código completo devem ser avaliadas em conjunto para que se obtenha um quadro fiel do resultado das ferramentas de IAG: análises baseadas exclusivamente em métricas estruturais, sem considerar a completude funcional, podem produzir conclusões enviesadas a favor de ferramentas que entregam pouco código completo. Ainda assim, o uso das ferramentas de IAG como suporte ao desenvolvimento humano parece ser o caminho mais promissor, em que o desenvolvedor humano mantém a responsabilidade pelas decisões arquiteturais, pela validação funcional e pelo refinamento estrutural do código gerado.

As principais contribuições deste trabalho são:

1. definição de um protocolo experimental para avaliar softwares Java gerados por ferramentas de Inteligência Artificial Generativa;
2. comparação entre modelos generalistas e assistentes especializados em geração de código, bem como entre versões gratuitas e pagas;
3. avaliação funcional dos sistemas gerados, distinguindo execuções completas, parciais e falhas;
4. aplicação de métricas de software para análise de acoplamento, coesão, herança, ocultação de informação, tamanho e complexidade, restrita aos softwares completos;
5. proposição do índice de Qualidade Efetiva, que combina, em um único valor, a qualidade estrutural dos códigos completos e a taxa de completude da ferramenta, permitindo evidenciar diferenças entre ferramentas que uma análise estrutural isolada não tornaria visíveis;
6. evidência de que a qualidade estrutural e a completude funcional são dimensões complementares, devendo ser avaliadas em conjunto para uma comparação fiel entre ferramentas de IAG

7.1 Trabalhos Futuros

Como trabalhos futuros, recomenda-se investigar o esforço necessário para corrigir sistemas que apresentaram funcionamento parcial ou falha. Essa análise permitiria estimar

quanto trabalho humano seria exigido para transformar os artefatos gerados em softwares plenamente funcionais.

Outra possibilidade consiste em aprofundar o estudo sobre engenharia de *prompt*, avaliando como diferentes níveis de detalhamento, restrições arquiteturais, padrões de projeto e requisitos não funcionais influenciam a qualidade estrutural dos códigos gerados.

Também se sugere replicar o experimento em outras linguagens de programação, como Python, C# ou JavaScript, a fim de verificar se os padrões observados em Java se repetem em outros ecossistemas. Além disso, estudos futuros podem incorporar outras ferramentas de análise estática e métricas arquiteturais complementares, ampliando o escopo da avaliação estrutural adotado neste trabalho.

Por fim, recomenda-se investigar estratégias de geração orientadas à qualidade, isto é, *prompts* explicitamente formulados para induzir maior coesão, menor acoplamento, melhor modularidade e melhor organização arquitetural dos sistemas gerados.

REFERÊNCIAS

Aniche, M. **CK: Java code metrics calculator**. 2015. Versão 0.7. [S. l.]: GitHub. Disponível em: <https://github.com/mauricioaniche/ck>. Acesso em: 8 jun. 2026. Citado 6 vezes nas páginas 2, 5, 7, 9, 13 e 34.

Bartié, A. **Garantia da qualidade de software: adquirindo maturidade organizacional**. Rio de Janeiro: Elsevier, 2013. Citado na página 4.

Becker, J.; Rush, N.; Barnes, E.; Rein, D. **Measuring the impact of early-2025 AI on experienced open-source developer productivity**. 2025. ArXiv:2507.09089. Relatório técnico do METR — Model Evaluation and Threat Research. Disponível em: <https://arxiv.org/abs/2507.09089>. Acesso em: 18 maio 2026. Citado na página 7.

Chidamber, S. R.; Kemerer, C. F. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 476–493, 1994. Disponível em: <https://doi.org/10.1109/32.295895>. Acesso em: 18 maio 2026. Citado 5 vezes nas páginas 6, 7, 9, 13 e 34.

Gonçalves, D. S. **Derivação de valores de referência de métricas de software**. 2022. Dissertação (Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Computação)) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 2022. Citado 9 vezes nas páginas 2, 4, 7, 9, 12, 13, 24, 34 e 35.

International Organization for Standardization. **ISO/IEC 25010:2023 — Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model**. 2. ed. Geneva, 2023. Citado na página 5.

Masuda, S.; Ono, K.; Yasue, T.; Hosokawa, N. A survey of software quality for machine learning applications. In: **Proceedings of the 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)**. Västerås, Sweden: IEEE, 2018. p. 279–284. Disponível em: <https://doi.org/10.1109/ICSTW.2018.00061>. Acesso em: 18 maio 2026. Citado na página 5.

Nguyen-Duc, A.; Cabrero-Daniel, B.; Przybylek, A.; Arora, C.; Khanna, D.; Herda, T.; Rafiq, U.; Melegati, J.; Guerra, E.; Kemell, K.-K. *et al.* Generative artificial intelligence for software engineering — a research agenda. **Software: Practice and Experience**, Wiley, 2023. Disponível em: <https://doi.org/10.1002/spe.3243>. Acesso em: 18 maio 2026. Citado 2 vezes nas páginas 1 e 3.

Nguyen, N.; Nadi, S. An empirical evaluation of GitHub Copilot's code suggestions. In: **Proceedings of the 19th International Conference on Mining Software Repositories (MSR '22)**. Pittsburgh, PA, USA: ACM, 2022. p. 1–5. Disponível em: <https://doi.org/10.1145/3524842.3528470>. Acesso em: 18 maio 2026. Citado na página 3.

Pereira, G. M. **Análise da qualidade de softwares gerados por inteligência artificial generativa**. 2025. Dissertação (Monografia (Graduação em Engenharia de Computação)) — Centro Federal de Educação Tecnológica de Minas Gerais, Belo Horizonte, 2025. Citado 5 vezes nas páginas 2, 4, 8, 11 e 29.

Pressman, R. S.; Maxim, B. R. **Software engineering: a practitioner's approach**. 9. ed. New York: McGraw-Hill Education, 2020. Citado na página 5.

Sabra, A.; Schmitt, O.; Tyler, J. **Assessing the quality and security of AI-generated code: a quantitative analysis**. 2025. ArXiv:2508.14727. Disponível em: <https://arxiv.org/abs/2508.14727>. Acesso em: 18 maio 2026. Citado 3 vezes nas páginas 1, 3 e 29.

Wang, Z.; Zhou, Z.; Song, D.; Huang, Y.; Chen, S.; Ma, L.; Zhang, T. Towards understanding the characteristics of code generation errors made by large language models. In: **Proceedings of the 47th IEEE/ACM International Conference on Software Engineering (ICSE)**. Ottawa, Canada: IEEE Computer Society, 2025. Citado 3 vezes nas páginas 1, 4 e 29.

Watanabe, K.; Tsuchida, R.; Monno, T.; Huang, B.; Yamasaki, K.; Fan, Y.; Shimari, K.; Matsumoto, K. **How AI coding agents communicate: a study of pull request description characteristics and human review responses**. 2026. ArXiv:2602.17084. Disponível em: <https://arxiv.org/abs/2602.17084>. Acesso em: 18 maio 2026. Citado na página 7.

Yetiştiren, B.; Özsoy, I.; Ayerdem, M.; Tüzün, E. **Evaluating the code quality of AI-assisted code generation tools: an empirical study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT**. 2023. ArXiv:2304.10778. Disponível em: <https://arxiv.org/abs/2304.10778>. Acesso em: 18 maio 2026. Citado 3 vezes nas páginas 1, 3 e 29.

ANEXO A – Métricas de Software Utilizadas

Este anexo descreve o conjunto de métricas de software utilizadas neste trabalho, organizadas por aspecto de qualidade (Tabela 3), bem como os valores de referência adotados para sua classificação (Tabela 4).

Tabela 3 – Descrição de algumas métricas de software utilizadas, agrupadas por aspecto de qualidade.

Aspecto de Qualidade	**Sigla**	**Nome**	**Descrição**
Acoplamento	CBO	*Coupling Between Objects*	Número de classes conectadas a uma classe.
	FAN-IN	*Fan-in*	Número de classes que chamam a classe analisada.
	FAN-OUT	*Fan-out*	Número de classes chamadas pela classe analisada; valores altos indicam muitas responsabilidades.
	RFC	*Response For a Class*	Métodos que podem ser chamados em resposta a um método da classe; valores altos indicam interface complexa.
	NOSI	*Number of Static Invocations*	Chamadas a métodos estáticos realizadas dentro da classe.
Coesão	LCOM	*Lack of Cohesion in Methods*	Grau de coesão dos métodos da classe; valores altos indicam baixa coesão.
	TCC	*Tight Class Cohesion*	Proporção de pares de métodos que acessam atributos comuns; valores altos indicam colaboração entre métodos.
	LCC	*Loose Class Cohesion*	Semelhante ao TCC, mas considera apenas chamadas diretas entre métodos.
Herança	DIT	*Depth of Inheritance Tree*	Profundidade da árvore de herança; valores altos dificultam a manutenção.
	NOC	*Number of Children*	Número de classes filhas; requer cuidado para evitar impactos em cascata.
Ocultação de Informação	—	Ocultação de informações	Avaliam níveis de acesso (*public*, *protected*, *private*); métodos protegidos reduzem exposição desnecessária.
Tamanho e Complexidade	LOC	*Lines of Code*	Linhas de código, desconsiderando comentários e linhas vazias.
	WMC	*Weighted Methods per Class*	Soma da complexidade dos métodos da classe; valores altos indicam maior esforço de manutenção e menor reutilização.

Fonte: adaptado de Chidamber e Kemerer (1994) e Aniche (2015).

VALORES DE REFERÊNCIA

A Tabela 4 apresenta os valores de referência derivados estatisticamente por Gonçalves (2022), utilizados como base para a classificação das métricas de software em três faixas (Bom/Frequente, Regular/Ocasional e Ruim/Raro) ao longo deste trabalho.

Tabela 4 – Valores de referência de métricas de softwares orientados a objetos.

Métrica	**Bom/Frequente**	**Regular/Ocasional**	**Ruim/Raro**
lcc	$= 1$	$0{,}6044 \leq lcc < 1$	$< 0{,}6044$
lcom	$< 0{,}4$	$0{,}4 \leq lcom < 0{,}7917$	$\geq 0{,}7917$
tcc	$= 1$	$0{,}4483 \leq tcc < 1$	$< 0{,}4483$
assignmentsQty	≤ 5	$5 < assignmentsQty \leq 21$	> 21

Continua na próxima página...

(Continuação da Tabela 4)

Métrica	**Bom/Frequente**	**Regular/Ocasional**	**Ruim/Raro**
cbo	≤ 10	$10 < cbo \leq 22$	> 22
dit	≤ 2	$2 < dit \leq 3$	> 3
fanin	≤ 2	$2 < fanin \leq 7$	> 7
fanout	≤ 7	$7 < fanout \leq 15$	> 15
finalFieldsQty	≤ 1	$1 < finalFieldsQty \leq 3$	> 3
loc	≤ 36	$36 < loc \leq 114$	> 114
maxNestedBlocksQty	≤ 1	$1 < maxNestedBlocksQty \leq 3$	> 3
modifiers	≤ 2	$2 < modifiers \leq 17$	> 17
nosi	≤ 2	$2 < nosi \leq 9$	> 9
numbersQty	≤ 1	$1 < numbersQty \leq 8$	> 8
privateFieldsQty	≤ 1	$1 < privateFieldsQty \leq 4$	> 4
publicMethodsQty	≤ 3	$3 < publicMethodsQty \leq 9$	> 9
returnQty	≤ 2	$2 < returnQty \leq 8$	> 8
rfc	≤ 9	$9 < rfc \leq 29$	> 29
stringLiteralsQty	≤ 3	$3 < stringLiteralsQty \leq 15$	> 15
totalFieldsQty	≤ 2	$2 < totalFieldsQty \leq 5$	> 5
totalMethodsQty	≤ 5	$5 < totalMethodsQty \leq 12$	> 12
uniqueWordsQty	≤ 43	$43 < uniqueWordsQty \leq 96$	> 96
variablesQty	≤ 5	$5 < variablesQty \leq 19$	> 19
wmc	≤ 7	$7 < wmc \leq 22$	> 22
abstractMethodsQty	≤ 3	$3 < abstractMethodsQty \leq 6$	> 6
anonymousClassesQty	≤ 2	$2 < anonymousClassesQty \leq 5$	> 5
comparisonsQty	≤ 5	$5 < comparisonsQty \leq 13$	> 13
defaultFieldsQty	≤ 3	$3 < defaultFieldsQty \leq 5$	> 5
defaultMethodsQty	≤ 2	$2 < defaultMethodsQty \leq 6$	> 6
finalMethodsQty	≤ 3	$3 < finalMethodsQty \leq 7$	> 7
innerClassesQty	≤ 2	$2 < innerClassesQty \leq 4$	> 4
lambdasQty	≤ 3	$3 < lambdasQty \leq 8$	> 8
logStatementsQty	≤ 4	$4 < logStatementsQty \leq 9$	> 9
loopQty	≤ 3	$3 < loopQty \leq 8$	> 8
mathOperationsQty	≤ 4	$4 < mathOperationsQty \leq 11$	> 11
noc	≤ 4	$4 < noc \leq 9$	> 9
parenthesizedExpsQty	≤ 3	$3 < parenthesizedExpsQty \leq 8$	> 8
privateMethodsQty	≤ 3	$3 < privateMethodsQty \leq 6$	> 6
protectedFieldsQty	≤ 2	$2 < protectedFieldsQty \leq 7$	> 7
protectedMethodsQty	≤ 3	$3 < protectedMethodsQty \leq 5$	> 5
publicFieldsQty	≤ 2	$2 < publicFieldsQty \leq 7$	> 7
staticFieldsQty	≤ 2	$2 < staticFieldsQty \leq 6$	> 6
staticMethodsQty	≤ 3	$3 < staticMethodsQty \leq 6$	> 6
synchronizedFieldsQty	≤ 0	$0 < synchronizedFieldsQty \leq 0$	> 0
synchronizedMethodsQty	≤ 3	$3 < synchronizedMethodsQty \leq 6$	> 6
tryCatchQty	≤ 2	$2 < tryCatchQty \leq 6$	> 6

Fonte: adaptado de Gonçalves (2022).

Agradecimentos

A autora agradece aos orientadores, Professora Kecia Aline Marques Ferreira e Professor Rogério Martins Gomes, pela orientação, disponibilidade e contribuições acadêmicas ao longo do desenvolvimento deste trabalho. Agradece também ao Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG) pelo suporte institucional e aos docentes que contribuíram para sua formação acadêmica.

Este trabalho contou com o apoio de ferramentas de inteligência artificial generativa, utilizadas sob supervisão da autora e dos orientadores como recursos auxiliares de revisão textual, organização de dados e apoio à análise gráfica. Ressalta-se que as interpretações, decisões metodológicas e redação final foram conduzidas pela autora, com acompanhamento dos orientadores.

MINISTÉRIO DA EDUCAÇÃO
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
DEPARTAMENTO DE COMPUTAÇÃO - NG

FOLHA DE APROVAÇÃO DE TCC Nº 24 / 2026 - DECOM (11.56.03)

Nº do Protocolo: 23062.033162/2026-42

Belo Horizonte-MG, 25 de junho de 2026.

SAMYRIS ALVES RODRIGUES

UMA ANÁLISE ABRANGENTE DA QUALIDADE ESTRUTURAL DE SOFTWARES JAVA GERADOS POR INTELIGÊNCIA ARTIFICIAL

Trabalho de Conclusão de Curso apresentado ao Curso de **Engenharia de Computação** do Centro Federal de Educação Tecnológica de Minas Gerais, do Campus **Nova Gameleira**, como parte do requisito para obtenção do grau de Bacharel em **Engenharia de Computação**.

Aprovado em 17 de junho de 2026.

Banca Examinadora:

Profª. Drª. Kecia Aline Marques Ferreira - Orientadora
Centro Federal de Educação Tecnológica de Minas Gerais

Profª. Drª. Rogério Martins Gomes - Co-orientador
Centro Federal de Educação Tecnológica de Minas Gerais

Profª. Drª. Daniela Cristina Cascini Kupsch
Centro Federal de Educação Tecnológica de Minas Gerais

Prof. Dr. Edson Marchetti da Silva
Centro Federal de Educação Tecnológica de Minas Gerais

Belo Horizonte
2026

(Assinado digitalmente em 26/06/2026 14:23)
DANIELA CRISTINA CASCINI KUPSCH
PROFESSOR ENS BASICO TECN TECNOLOGICO
CECOM (11.51.11)
Matrícula: 2092029

(Assinado digitalmente em 26/06/2026 14:44)
EDSON MARCHETTI DA SILVA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 1509224

(Assinado digitalmente em 26/06/2026 15:01)
KECIA ALINE MARQUES FERREIRA
PROFESSOR ENS BASICO TECN TECNOLOGICO
DECOM (11.56.03)
Matrícula: 1670931

(Assinado digitalmente em 26/06/2026 13:52)
ROGERIO MARTINS GOMES
ASSINANTE EXTERNO
CPF: ###.###.846-##

Visualize o documento original em **https://sig.cefetmg.br/public/documentos/index.jsp** informando seu número: **24**, ano: **2026**, tipo: **FOLHA DE APROVAÇÃO DE TCC**, data de emissão: **25/06/2026** e o código de verificação: **e7dba2adbf**