UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MARIANA REGINA CABRINHA DE LIMA

REFATORAÇÃO DE CÓDIGO: COMPARAÇÃO ENTRE ABORDAGENS MANUAIS E INTELIGÊNCIA ARTIFICIAL GENERATIVA

PONTA GROSSA

2026

versão 11.0 (abr. 25)

MARIANA REGINA CABRINHA DE LIMA

REFATORAÇÃO DE CÓDIGO: COMPARAÇÃO ENTRE ABORDAGENS MANUAIS E INTELIGÊNCIA ARTIFICIAL GENERATIVA

Code Refactoring: Comparison Between Manual Approaches And Generative Artificial Intelligence

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.
Orientador(a): Profª. Drª. Simone Nasser Matos

PONTA GROSSA

2026

MARIANA REGINA CABRINHA DE LIMA

REFATORAÇÃO DE CÓDIGO: COMPARAÇÃO ENTRE ABORDAGENS MANUAIS E INTELIGÊNCIA ARTIFICIAL GENERATIVA

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 25/05/2026

Eliana Cláudia Mayumi Ishikawa
Doutorado em Ensino de Ciência e Tecnologia
Universidade Tecnológica Federal do Paraná

Luiz Rafael Schmitke
Doutorado em Informática
Universidade Tecnológica Federal do Paraná

Simone Nasser Matos
Doutorado em Engenharia Eletrônica e Computação
Universidade Tecnológica Federal do Paraná

PONTA GROSSA

2026

AGRADECIMENTOS

Gostaria de registrar minha mais sincera gratidão a todas as pessoas que, de alguma forma, contribuíram para que este trabalho se tornasse realidade. Agradeço especialmente à minha mãe e à minha avó Guilhermina, por sempre acreditarem em mim. Agradeço à minha família pelo carinho e pelo apoio.

Aos meus melhores amigos, que sempre me incentivaram a cuidar da minha saúde mental e me mostraram que parar um pouco também faz parte do processo. Obrigada por cada conversa, cada abraço e cada risada.

A todos os amigos e professores que conheci ao longo da minha trajetória na UTFPR, sou imensamente grata. Levo comigo cada aprendizado, cada troca e cada inspiração que recebi ao longo desses anos.

Por fim, agradeço a todos que, direta ou indiretamente, fizeram parte dessa jornada. Cada um de vocês também faz parte desta conquista.

RESUMO

A refatoração de código é uma prática essencial na Engenharia de Software, voltada à melhoria da estrutura interna do código sem alterar seu comportamento externo, contribuindo para a qualidade, a manutenibilidade e a evolução dos sistemas ao longo do tempo. Com o avanço das ferramentas de Inteligência Artificial Generativa, surge a possibilidade de automatizar esse processo, levantando questões sobre a eficácia e as limitações dessas abordagens em comparação à intervenção humana. Este trabalho apresenta uma comparação entre a refatoração de código realizada manualmente e com o auxílio das ferramentas GitHub Copilot e Amazon Q Developer, tendo como objetos de estudo três jogos educacionais desenvolvidos no âmbito do Projeto de Desenvolvimento de Software Educacional da Universidade Tecnológica Federal do Paraná (UTFPR). A refatoração manual foi conduzida com base em técnicas clássicas descritas por Fowler, enquanto a refatoração automatizada utilizou um prompt padronizado aplicado às mesmas ferramentas. A avaliação dos resultados foi realizada por meio de métricas de qualidade obtidas com o SonarQube, contemplando atributos relacionados à manutenibilidade, compreensibilidade, reusabilidade e complexidade estrutural. Os resultados indicam que as ferramentas de IA Generativa foram capazes de promover melhorias relevantes em diferentes métricas de qualidade, embora tenham apresentado limitações relacionadas à preservação de contexto, geração de erros de compilação e introdução de alterações que extrapolam o escopo tradicional da refatoração. Observou-se ainda que cada ferramenta apresentou melhor desempenho em atributos específicos, enquanto a abordagem manual demonstrou maior controle sobre as decisões arquiteturais e aplicação direcionada das técnicas de refatoração. Os resultados contribuem para a compreensão do potencial e das limitações da IA Generativa como apoio à manutenção e evolução de software.

Palavras-chave: refatoração de código; inteligência artificial generativa; qualidade de software.

ABSTRACT

Code refactoring is an essential practice in Software Engineering, aimed at improving the internal structure of code without altering its external behavior, thereby contributing to the quality, maintainability, and long-term evolution of software systems. With the advancement of Generative Artificial Intelligence tools, the possibility of automating this process has emerged, raising questions about the effectiveness and limitations of such approaches in comparison to human intervention. This paper presents a comparison between manual code refactoring and refactoring assisted by GitHub Copilot and Amazon Q Developer, using three educational games developed within the Educational Software Development Project at the Federal University of Technology – Paraná (UTFPR) as case studies. Manual refactoring was conducted based on classical techniques described by Fowler, while automated refactoring was performed using a standardized prompt applied to both tools. The results were evaluated through software quality metrics obtained using SonarQube, considering attributes related to maintainability, comprehensibility, reusability, and structural complexity. The findings indicate that Generative AI tools were capable of achieving significant improvements in several quality metrics. However, they also exhibited limitations related to context preservation, generation of compilation errors, and the introduction of modifications that extend beyond the traditional scope of refactoring. Furthermore, each tool demonstrated superior performance in specific quality attributes, whereas the manual approach provided greater control over architectural decisions and the targeted application of refactoring techniques. The results contribute to a better understanding of the potential and limitations of Generative Artificial Intelligence as a support mechanism for software maintenance and evolution.

Keywords: code refactoring; generative artificial intelligence; software quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

A refatoração de código, conforme definida por Fowler (2020, p. 12), é "o processo de modificar um sistema de software de modo que não altere o comportamento externo do código, embora melhore a sua estrutura interna". Tradicionalmente realizada manualmente, a refatoração tem como objetivo aprimorar aspectos como manutenibilidade, compreensibilidade e reusabilidade do código. No entanto, com os recentes avanços tecnológicos, a Inteligência Artificial (IA) surgiu como uma alternativa para a automação dessa atividade, trazendo novas possibilidades, mas também levantando questões sobre as capacidades, confiabilidade e limitações dessas ferramentas (DePalma *et al.*, 2024).

As ferramentas de Inteligência Artificial Generativa (IAG), como GitHub Copilot, ChatGPT e Sourcery, são capazes de gerar trechos de código a partir de comandos em linguagem natural (Noring *et al.*, 2024). Treinadas em grandes volumes de dados, essas ferramentas interpretam diferentes linguagens de programação, como Python, Java, JavaScript, C++, podendo ser aplicadas na automação de tarefas antes realizadas manualmente (Almanasra; Suwais, 2025). Apesar dos ganhos em produtividade, o uso dessas ferramentas no desenvolvimento de software levanta questões quanto à introdução de vulnerabilidades no sistema, à compreensão limitada do contexto completo dos projetos e à limitação na aderência a padrões de qualidade estabelecidos (Almanasra; Suwais, 2025; Rebai *et al.*, 2020).

Enquanto a refatoração manual depende da análise crítica e da expertise de desenvolvedores humanos, as abordagens automatizadas baseadas em IA conseguem identificar padrões e sugerir modificações de forma rápida, embora apresentem limitações. De acordo com Gouveia (2024), algumas ferramentas, como o GitHub Copilot[1], auxiliam na geração de código em tempo real, enquanto outras, como o Sourcery[2], são voltadas à sugestão de melhorias em trechos de código já existentes. Além disso, ferramentas como o ChatGPT[3] permitem a geração de código a partir de descrições em linguagem natural, e modelos como o OpenAI Codex[4], treinados em vastos repositórios de código, oferecem suporte a múltiplas funcionalidades (Gouveia, 2024; Lira; Neto; Osorio, 2024).

A principal problemática do uso de IAs Generativas na refatoração reside na comparação entre a qualidade do código refatorado manualmente e aquele modificado por IA, levantando dúvidas quanto à confiabilidade das ferramentas automáticas (AlOmar *et al.*, 2021; DePalma *et al.*, 2024). Embora úteis na identificação de padrões e sugestões de melhoria, essas tecnologias apresentam limitações na garantia da qualidade estrutural do código produzido. Sem a avaliação humana sobre as mudanças propostas, alterações sugeridas por IAs, embora sintaticamente corretas, podem não trazer benefícios efetivos ao projeto em termos de desempenho e manutenção (AlOmar, 2025).

[1] GitHub Copilot. Disponível em: https://github.com/features/copilot. Acesso em 30 de maio de 2025.
[2] Sourcery. Disponível em: https://sourcery.ai. Acesso em 30 de maio de 2025.
[3] ChatGPT. Disponível em: https://chatgpt.com. Acesso em 30 de maio de 2025.
[4] OpenAI Codex. Disponível em: https://openai.com/codex. Acesso em: 30 de maio de 2025.

Diante desse contexto, o presente trabalho realizou uma análise comparativa entre a refatoração manual, realizada por uma desenvolvedora com experiência prévia em desenvolvimento de software, e a efetuada com auxílio de Inteligência Artificial Generativa. Para isso, foram aplicadas técnicas de refatoração a códigos de jogos desenvolvidos pelo Laboratório de Engenharia de Software e Inteligência Computacional (LESIC) (LESIC, 2026) da UTFPR. Esses códigos fazem parte do Projeto de Desenvolvimento de Software Educacional (DSE, 2026), sendo compostos por jogos desenvolvidos por acadêmicos dos cursos de computação da UTFPR Campus Ponta Grossa, com foco no apoio a práticas educacionais e no desenvolvimento de habilidades computacionais.

A eficácia de cada abordagem foi avaliada com base em métricas de qualidade de código, considerando os seguintes critérios: i) manutenibilidade, definida como a facilidade de modificar o software ao longo do tempo (Sommerville, 2011, p. 466); ii) compreensibilidade (ou legibilidade), que se refere à facilidade de entendimento do código por outros desenvolvedores (Riemer, 2024); iii) reusabilidade, que corresponde à capacidade de reutilizar componentes em diferentes partes do sistema ou em novos projetos; e iv) complexidade estrutural, fator que influencia diretamente tanto a manutenibilidade quanto a compreensão do código (Riemer, 2024; Sommerville, 2011, p. 473).

Este estudo contribuiu para uma melhor compreensão das potencialidades e limitações das ferramentas de IA Generativa no contexto da refatoração de código, além de identificar cenários em que a intervenção humana ainda se faz necessária para garantir a qualidade do software.

1.1 Objetivos

Esta seção descreve os objetivos que motivaram a elaboração deste trabalho.

1.1.1 Objetivo geral

Comparar os resultados da refatoração de código realizada manualmente e por Inteligência Artificial Generativa, considerando atributos de qualidade relacionados à manutenibilidade, compreensibilidade, reusabilidade e complexidade estrutural.

1.1.2 Objetivos específicos

Os objetivos específicos que nortearam o desenvolvimento deste trabalho foram:

- Selecionar três jogos desenvolvidos pelo Projeto de Desenvolvimento de Software Educacional (DSE) para serem utilizados como objetos de estudo;
- Aplicar técnicas e boas práticas de refatoração manualmente;

- Executar a refatoração utilizando ferramentas de Inteligência Artificial Generativa;
- Comparar os resultados das abordagens manual e automatizada com base em métricas de qualidade de código.

1.2 Justificativa

O crescente uso de IAs Generativas no contexto da refatoração de código apresenta tanto benefícios quanto desafios. De um lado, essas ferramentas prometem acelerar o processo, aumentando a produtividade e sugerindo melhorias que poderiam passar despercebidas por desenvolvedores humanos (DePalma *et al.*, 2024). De outro lado, surgem preocupações quanto à qualidade do código gerado, à aderência às boas práticas de engenharia de software e ao risco de dependência excessiva de sistemas automatizados (Almanasra; Suwais, 2025).

Compreender as diferenças entre as abordagens manual e automatizada torna-se fundamental para avaliar o impacto das tecnologias de IA no desenvolvimento de software. Além disso, ao investigar critérios de qualidade em diferentes contextos, pode-se gerar *insights* relevantes tanto para desenvolvedores quanto para empresas sobre como aplicar essas tecnologias com mais segurança e eficiência (Asare; Nagappan; Asokan, 2023).

1.3 Organização do trabalho

Este trabalho está organizado da seguinte forma: O Capítulo 1 apresenta o tema, os objetivos, a justificativa e a estrutura geral do trabalho. O Capítulo 2 detalha a metodologia utilizada para a realização da pesquisa, incluindo a seleção dos códigos, a aplicação das técnicas de refatoração e a escolha das ferramentas de IA. O Capítulo 3 apresenta os resultados obtidos com as refatorações manuais e automatizadas e analisa os dados usando métricas de qualidade. O Capítulo 4 sintetiza as principais conclusões do trabalho, suas limitações e propostas para pesquisas futuras.

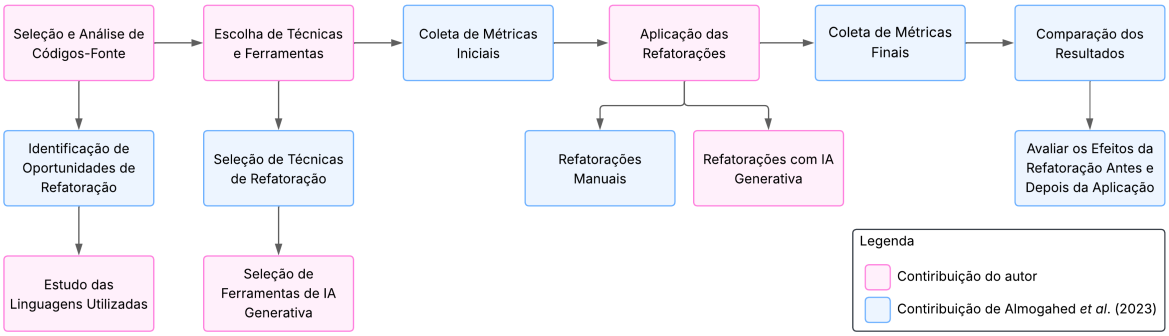
2 METODOLOGIA

Este capítulo descreve a metodologia adotada para a realização do estudo comparativo entre a refatoração manual e a refatoração automatizada por meio de ferramentas de Inteligência Artificial Generativa. A seção 2.1 apresenta uma visão geral da metodologia utilizada neste trabalho. A seção 2.2 detalha o processo de seleção e análise dos códigos-fonte a serem refatorados. A seção 2.3 descreve as ferramentas de IA Generativa selecionadas. A seção 2.4 aborda a execução da refatoração manual dos códigos, enquanto a seção 2.5 trata da aplicação da refatoração com o auxílio de Inteligência Artificial Generativa. A seção 2.6 apresenta os critérios de avaliação adotados, detalhando as métricas utilizadas.

2.1 Visão Geral

Este trabalho adotou uma abordagem de estudo empírico comparativo, fundamentada na *Metodologia de Aplicação de Técnicas de Refatoração* descrita por Almogahed *et al*. (2023), a qual descreve etapas para aplicação de refatorações e avaliação de seus impactos em atributos de qualidade. A metodologia empregada, conforme ilustra a Figura 1, adapta e estende essa abordagem para a realização da comparação entre as refatorações manuais e aquelas realizadas com o auxílio de ferramentas de IA Generativa.

Figura 1 – Visão Geral da Metodologia



Fonte: Autoria própria (2026).

O processo iniciou-se com uma revisão bibliográfica sobre técnicas de refatoração, boas práticas de programação e o uso de Inteligência Artificial Generativa na automação do desenvolvimento de software. Foram consultadas fontes como artigos científicos, livros, documentações técnicas e repositórios especializados.

A partir da revisão, foram selecionados os códigos-fonte a serem analisados, provenientes de projetos desenvolvidos no Laboratório de Engenharia de Software e Inteligência Computacional (LESIC, 2026) da UTFPR. A escolha priorizou projetos com maior potencial para refatoração, além de apresentarem complexidade e tamanho adequados para viabilizar a aplicação prática das técnicas e a análise dos efeitos. Também foi realizado um estudo aprofundado

nas linguagens de programação utilizadas nesses projetos, a fim de embasar o processo de refatoração.

Na etapa seguinte, foram definidas as técnicas de refatoração consideradas no estudo, com base no catálogo de Fowler (2026) e em sua aplicabilidade aos projetos analisados. Entre as principais técnicas avaliadas estão *Extract Method*, *Extract Class*, *Remove Dead Code*, *Replace Magic Number*, *Rename Method*, *Encapsulate Field*.

Após a definição das técnicas, foram estabelecidos os critérios de comparação entre as abordagens manual e assistida por Inteligência Artificial Generativa, sendo eles manutenibilidade, compreensibilidade, reusabilidade e complexidade estrutural, além da análise do tempo necessário para execução das refatorações e do esforço requerido em cada abordagem. Antes da realização de qualquer modificação, foram coletadas métricas iniciais dos projetos por meio de ferramentas de análise estática, constituindo a linha de base para as avaliações posteriores.

As refatorações manuais foram realizadas observando os princípios de preservação do comportamento e as boas práticas de design. Todo o processo foi documentado, incluindo tempo despendido, decisões tomadas e dificuldades enfrentadas. A validação funcional foi conduzida por meio de execução manual dos sistemas, assegurando a equivalência comportamental.

Na sequência, os mesmos projetos foram submetidos à refatoração com o auxílio das ferramentas GitHub Copilot e Amazon Q Developer. Para garantir a comparabilidade entre os resultados, foi utilizado um único *prompt* padronizado durante todo o experimento. As sugestões geradas pelas ferramentas foram incorporadas com intervenção humana restrita às situações necessárias para correção de erros de compilação, execução ou integração das modificações propostas. Após cada etapa, também foi realizada validação funcional dos sistemas.

Concluídas as refatorações, foram coletadas novamente as métricas de qualidade por meio de ferramentas de análise estática. Adicionalmente, foram registrados dados complementares, como tempo de execução, quantidade de interações necessárias com as ferramentas de IA, correções realizadas para resolução de erros e observações qualitativas sobre os resultados obtidos em cada abordagem.

Por fim, os resultados foram comparados com base nas métricas coletadas e em critérios qualitativos, como clareza do código, aderência a boas práticas e facilidade de uso das abordagens. Essa análise permitiu identificar os pontos fortes e limitações de cada método, contribuindo para uma avaliação crítica da viabilidade da Inteligência Artificial Generativa como suporte à refatoração de código.

A metodologia adotada buscou assegurar a confiabilidade e reprodutibilidade dos resultados, em conformidade com as boas práticas da engenharia de software e da pesquisa científica. Cada uma de suas etapas é detalhada nas próximas seções.

2.2 Seleção dos Casos de Estudo

A escolha dos códigos-fonte utilizados nesta pesquisa baseou-se em critérios técnicos e metodológicos, com o objetivo de garantir uma base sólida para a aplicação e comparação entre abordagens de refatoração manual e assistida por IA Generativa.

Os projetos analisados foram desenvolvidos pelo Laboratório de Engenharia de Software e Inteligência Computacional da Universidade Tecnológica Federal do Paraná (UTFPR) e encontram-se disponíveis publicamente na página do projeto (LESIC, 2026).

2.2.1 Critérios de Seleção

O processo teve início com um levantamento dos jogos educacionais disponíveis, juntamente com a identificação das tecnologias utilizadas em cada um deles. O Quadro 1 apresenta um resumo dos mesmos, indicando o ano, autor, linguagem e ferramentas utilizadas.

Quadro 1 – Resumo dos Jogos Analisados

Ano	Nome do Jogo	Autor(es)	Linguagem/Ferramentas
2021	Querida Floresta	Luan Leme	Unity / C#
2023	Querida Floresta 2.0	Wagner Pichler Junior	Unity / C#
2023	Memoletrando	Lucas Paris e Rafael de Campos	Angular / TypeScript / Bootstrap
2024	ECO-MAN	Lukas Kenji Miazaki Hashimoto	Pygame / Python
2024	Florarium	Gabriel De Andrade Sauter	Godot / GDScript
2024	SpeakSnake	Lázaro Santos	JavaScript / HTML / CSS / PHP
2024	VacinandoEdu	Eduardo de Almeida Bento Dias	Flutter / Dart (Android Studio)

Fonte: Autoria própria (2026).

Os sete jogos inicialmente avaliados foram submetidos à análise estática utilizando o SonarQube. Desses, os três projetos selecionados foram aqueles que apresentaram simultaneamente melhor aderência aos seguintes critérios:

1. Quantidade e diversidade de problemas detectados por ferramentas de análise estática, sendo utilizada a ferramenta SonarQube para identificação de *issues* (falhas, vulnerabilidades ou problemas de qualidade no código) relacionados à manutenibilidade, compreensibilidade, reusabilidade e complexidade estrutural. Conforme Lenarduzzi *et al*. (2023), o SonarQube é uma das ferramentas mais utilizadas para análise estática de código.

2. Relevância da linguagem de programação utilizada, priorizando jogos desenvolvidos em linguagens com maior suporte por ferramentas de Inteligência Artificial Generativa. Conforme estudos encontrados na literatura (Almogahed *et al*., 2023; Almanasra;

Suwais, 2025), linguagens como JavaScript e C# ocupam posições de destaque tanto em pesquisas relacionadas à refatoração de código quanto em estudos sobre ferramentas de IA aplicadas à engenharia de software.

3. Volume de código-fonte disponível, uma vez que projetos com maior quantidade de código oferecem um ambiente mais rico para a aplicação e avaliação de técnicas de refatoração.

2.2.2 Projetos Selecionados e Arquitetura Lógica

Com base nos critérios de seleção definidos, três jogos educacionais foram escolhidos para análise: Memoletrando, Querida Floresta 2.0 e SpeakSnake. A seguir, são apresentados os principais motivos de escolha, a arquitetura lógica de cada sistema e as métricas de qualidade extraídas pelo SonarQube.

2.2.2.1 Memoletrando

Desenvolvido com Angular, TypeScript, Ionic, Laravel e Bootstrap, o jogo Memoletrando foi selecionado por apresentar uma quantidade significativa de problemas relacionados à manutenibilidade, identificados por meio da ferramenta SonarQube, além de alta complexidade estrutural e taxa de duplicação de código.

A arquitetura do projeto é organizada em múltiplas camadas, favorecendo a separação de responsabilidades e permitindo a análise de refatorações em diferentes contextos tecnológicos. Sua estrutura é composta pelas seguintes camadas:

- Camada de Interface (UI): Implementada com Angular, Ionic, HTML e CSS, responsável pela apresentação visual do sistema e interação com o usuário por meio de componentes, páginas e *templates*.
- Camada de Aplicação: Composta por *scripts* em TypeScript e serviços responsáveis pelo gerenciamento do fluxo da aplicação, regras de negócio do *front-end* e comunicação com os serviços do *back-end*.
- Camada de Serviços e Negócio: Implementada com PHP e o *framework* Laravel, responsável pelo processamento das regras de negócio, validações e gerenciamento das requisições da aplicação.
- Camada de Dados: Estruturada segundo o padrão MVC (Model-View-Controller) do Laravel e integrada a um banco de dados MySQL para armazenamento e recuperação das informações do sistema.

A organização modular do projeto e a diversidade de tecnologias empregadas possibilitam a observação dos efeitos das refatorações em diferentes camadas arquiteturais. A Tabela 1 apresenta a distribuição das linhas de código do projeto Memoletrando.

Tabela 1 – Distribuição de linhas de código no Memoletrando

Tecnologia	Linhas de Código
Angular (Front-End)	3.929
Ionic (Front-End)	11.117
Laravel (Back-End)	7.757
Public	27
Total	**22.830**

Fonte: Autoria própria (2026).

2.2.2.2 Querida Floresta 2.0

O jogo Querida Floresta 2.0, desenvolvido com a *engine* Unity e a linguagem C#, foi selecionado por apresentar uma quantidade significativa de problemas relacionados à manutenibilidade, identificados por meio da análise estática realizada com o SonarQube, além de alta complexidade estrutural. A arquitetura lógica do sistema pode ser organizada em três camadas principais:

- Camada de Interface (UI): Responsável pela interação com o jogador, abrangendo elementos visuais, gerenciamento de cenas, componentes de interface e *scripts* relacionados à experiência do usuário.
- Camada de Aplicação e Controle: Composta pelos *scripts* responsáveis pela lógica do jogo, controle de estados, gerenciamento de áudio, carregamento de cenas, transições e demais funcionalidades que coordenam o comportamento da aplicação.
- Camada de Dados: Formada por classes responsáveis pelo armazenamento e gerenciamento de informações do jogador, incluindo progresso, configurações e estatísticas de utilização do sistema.

A estrutura do projeto segue o padrão organizacional da Unity, com separação entre diretórios de Assets, Scenes e Scripts, promovendo a modularização e distribuição de responsabilidades. A Tabela 2 apresenta a distribuição das linhas de código do jogo.

Tabela 2 – Distribuição de linhas de código no Querida Floresta 2.0

Seção	Diretórios ou Arquivos Incluídos na Contagem	LOC
Cenas (UI + Lógica Local)	AuthMenu, MainMenu, PlayersForest, Quiz2, Scene1-9	2.668
Gerenciamento de Dados	Data, RegisterMenu	540
Estatísticas e Relatórios	Statistics, ReportCreator.cs	86
Gerenciamento de Áudio	AudioController, MusicPlayer, NarratorController	77
Lógica Geral	ConnectElements, SceneSelection, SceneLoader.cs	315
Utilitários	Utils, AplicationModel.cs, RectTransformExtensions.cs	78
Total do Código-Fonte Principal	**Project/Scripts**	**3.764**
Dependências e Configuração	Packages, RainMaker, NuGet.config, packages.config	45.373
Total Geral		**49.137**

Fonte: Autoria própria (2026).

2.2.2.3 SpeakSnake

O jogo SpeakSnake foi selecionado por apresentar elevado índice de duplicação de código e problemas de manutenibilidade identificados por meio da análise estática realizada com o SonarQube. Além disso, sua arquitetura simples e pouco modularizada constitui um cenário interessante para avaliação dos impactos da refatoração manual e assistida por Inteligência Artificial Generativa, permitindo observar como diferentes abordagens atuam em projetos com baixa organização estrutural.

Desenvolvido com HTML, CSS, JavaScript e PHP, o projeto adota uma arquitetura monolítica, sem utilização de *frameworks* ou padrões arquiteturais formalmente definidos. Sua estrutura pode ser organizada nas seguintes camadas:

- Camada de Interface (UI): Implementada por meio de arquivos HTML e CSS, responsáveis pela apresentação visual do sistema e pela interação com o usuário.
- Camada de Aplicação: Composta por *scripts* JavaScript responsáveis pelas funcionalidades do jogo, manipulação da interface, validações e controle do fluxo de execução.
- Camada de Dados: Implementada em PHP, responsável pelo acesso ao banco de dados, processamento das regras de negócio e comunicação com o *front-end* por meio de requisições AJAX.

Diferentemente dos demais projetos analisados, o SpeakSnake apresenta uma organização estrutural mais simples, com arquivos concentrados em poucos diretórios e limitada separação de responsabilidades entre os componentes do sistema. A Tabela 3 apresenta a distribuição das linhas de código do projeto SpeakSnake.

Tabela 3 – Distribuição de linhas de código no SpeakSnake

Camada	**Linhas de Código**
Interface (HTML e CSS)	797
Lógica (JavaScript)	932
Dados (PHP puro)	68
Total	**1.797**

Fonte: Autoria própria (2026).

2.3 Seleção das Ferramentas de IA Generativa

Diversas ferramentas de IA Generativa estão disponíveis no mercado com funcionalidades voltadas à geração, sugestão e otimização de código. O Quadro 2 traz uma comparação entre diferentes ferramentas de IA Generativa avaliadas na fase de seleção, com destaque para suas funcionalidades e limitações.

Apesar da diversidade de ferramentas de Inteligência Artificial Generativa disponíveis, nem todas se mostraram adequadas aos objetivos deste estudo. Ferramentas como o ChatGPT,

Quadro 2 – Ferramentas de IA Generativa Consideradas para Refatoração de Código

Ferramenta	Funcionalidade Principal	Limitações
ChatGPT	Geração e revisão de código via linguagem natural	Ferramenta de propósito geral, não é específica à geração de código; compreensão limitada do contexto de um projeto.
IntelliCode	Sugestões de código inteligentes integradas à IDE Visual Studio.	Limitações em interpretar e resolver problemas complexos; sua documentação técnica e suporte a linguagens são restritos.
Kite AI	Autocompletar inteligente com sugestões de código.	Desempenho comprometido em projetos com múltiplas linguagens de programação; ferramenta descontinuada em 2021[1].
OpenAI Codex	Geração de código a partir de descrições em linguagem natural.	Como o GitHub Copilot incorpora suas funcionalidades, seu uso seria redundante. Além disso, o Copilot oferece mais recursos e melhor integração com IDEs.
Sourcery	Sugestões de refatoração de código para Python.	Restrito à linguagem Python.
Tabnine	Sugestões preditivas de código baseadas em IA.	Entendimento limitado do contexto do projeto; pode gerar sugestões superficiais ou redundantes.

Fonte: Autoria própria (2025), com base em Gouveia (2024) e Haque (2025).

embora versáteis, apresentam limitações quanto à compreensão do contexto completo de projetos de software, o que pode impactar a qualidade das sugestões de refatoração. O IntelliCode e o Tabnine, por sua vez, oferecem suporte mais restrito em termos de profundidade analítica, sendo mais voltados à sugestão preditiva de código do que à refatoração.

Já o Kite AI foi desconsiderado por se tratar de uma ferramenta descontinuada, o que inviabiliza sua utilização. O Sourcery, embora possivelmente eficaz, possui limitação quanto às linguagens suportadas, sendo restrito ao Python, o que não atende integralmente às tecnologias utilizadas nos projetos analisados. Por fim, o OpenAI Codex não foi selecionado por ter suas funcionalidades amplamente incorporadas ao GitHub Copilot, tornando seu uso redundante no contexto desta pesquisa.

Dessa forma, optou-se pela utilização de ferramentas que apresentam maior aderência aos critérios definidos, especialmente em termos de compatibilidade, integração com o ambiente de desenvolvimento e aplicabilidade prática na refatoração de código. As ferramentas selecionadas foram: GitHub Copilot (Gouveia, 2024; Lira; Neto; Osorio, 2024; Asare; Nagappan; Asokan, 2023; Lira; Neto; Osorio, 2024) e Amazon Q Developer[1] (Cheng *et al.*, 2024; Gouveia, 2024; Haque, 2025; Huang *et al.*, 2024).

As ferramentas foram testadas na IDE Visual Studio Code, ambiente de desenvolvimento original dos jogos, de forma a simular cenários reais de manutenção. A escolha das ferramentas também considerou sua capacidade de integração com repositórios Git, facilitando a comparação entre versões do código antes e depois das refatorações.

[1] O Amazon CodeWhisperer é atualmente denominado Amazon Q Developer (ou simplesmente Amazon Q), após a reformulação do serviço anunciada pela Amazon em 2024.

2.4 Aplicação da Refatoração Manual

A refatoração manual foi conduzida com o objetivo de promover melhorias estruturais nos códigos analisados, sem alterar seu comportamento funcional externo. O processo foi orientado pelas boas práticas de engenharia de software, sendo executado manualmente por uma desenvolvedora com familiaridade com as tecnologias utilizadas nos projetos analisados. Para assegurar a consistência e a validade dos resultados, seguiu-se o seguinte fluxo de atividades, alinhado à *Metodologia de Aplicação de Técnicas de Refatoração* proposta por Almogahed *et al*. (2023):

1. Validação Funcional Inicial: Antes de qualquer modificação, cada sistema foi executado em seu ambiente original, com o intuito de registrar seu comportamento esperado e garantir uma base confiável para comparação posterior.

2. Aplicação da Refatoração Manual: As técnicas de refatoração, descritas na subseção 3.1.1, foram aplicadas com sua execução sendo realizada conforme os procedimentos estabelecidos por Fowler em seu repositório (Fowler, 2026).

3. Validação Funcional Pós-Refatoração: Após cada etapa de modificação, os sistemas foram novamente executados para confirmar que nenhuma alteração comprometeu o comportamento funcional original, assegurando a equivalência funcional.

4. Registro das Alterações: Todas as modificações foram documentadas, incluindo os arquivos alterados, trechos modificados, justificativas técnicas para cada intervenção e o tempo despendido em cada tarefa.

O processo de refatoração foi conduzido no ambiente de desenvolvimento Visual Studio Code, garantindo uniformidade nas condições de trabalho entre esta etapa e a aplicação da refatoração assistida por Inteligência Artificial Generativa.

2.4.1 Perfil do Desenvolvedor

A refatoração manual foi conduzida por uma desenvolvedora em nível júnior, com experiência prévia em desenvolvimento de software e familiaridade com as tecnologias empregadas nos projetos analisados.

A desenvolvedora possuía experiência em linguagens como JavaScript, TypeScript, PHP e C#, além de conhecimento sobre princípios de Engenharia de Software e técnicas de refatoração descritas por Fowler (2020). Essa caracterização foi importante para contextualizar os resultados obtidos pela abordagem manual e permitir comparações futuras envolvendo diferentes níveis de experiência profissional.

[1] A ferramenta Kite AI foi oficialmente descontinuada em novembro de 2021, conforme anunciado no site da empresa: https://kite.com/. Acesso em: 15 jun. 2025.

2.5 Aplicação da Refatoração com IA Generativa

A refatoração com suporte de Inteligência Artificial Generativa foi aplicada com o intuito de comparar seu desempenho à abordagem manual, mantendo o mesmo escopo, critérios de avaliação e ambiente de execução. Assim como na refatoração manual, buscou-se assegurar a equivalência funcional e a imparcialidade metodológica entre as abordagens.

Para essa etapa, foram utilizadas as ferramentas GitHub Copilot e Amazon Q, conforme justificado na seção 2.3. O processo foi conduzido no ambiente de desenvolvimento Visual Studio Code, adotando os mesmos projetos e configurações utilizados na refatoração manual. A execução seguiu as seguintes etapas, em conformidade com a metodologia empregada por DePalma *et al*. (2024), para a aplicação de refatoração com LLMs.

1. Preparação do Ambiente: As extensões oficiais das ferramentas foram instaladas e configuradas no Visual Studio Code, visando reproduzir um ambiente de desenvolvimento realista, reduzindo interferências externas que pudessem afetar os resultados.

2. Aplicação dos *Prompts*: Para garantir consistência e minimizar a influência subjetiva na geração de código, utilizaram-se *prompts* padronizados durante todo o processo de refatoração, descritos na subseção 2.5.1. Os *prompts* foram aplicados individualmente a cada arquivo selecionado para refatoração, com o propósito de reduzir vieses na formulação dos comandos e assegurar que as variações nos resultados decorressem das próprias ferramentas, e não de instruções humanas.

3. Aplicação das Sugestões: As alterações propostas pelas ferramentas foram incorporadas aos projetos mantendo-se a menor intervenção humana possível, conforme detalhado na subseção 2.5.1. Correções pontuais foram realizadas apenas quando identificados erros de compilação, falhas de execução ou inconsistências decorrentes das alterações geradas. Todas as intervenções realizadas durante o processo foram registradas para posterior análise e comparação entre as abordagens.

4. Validação Funcional e Registro: Assim como na abordagem manual, após cada refatoração, realizou-se uma validação funcional do sistema, assegurando que o comportamento externo permanecesse equivalente ao original. Todas as modificações foram registradas, incluindo o código gerado, eventuais correções adicionais, o tempo de execução e observações sobre a eficácia das sugestões fornecidas pelas IAs.

2.5.1 *Prompts* Utilizados

Para garantir a comparabilidade entre as ferramentas avaliadas e reduzir a influência de diferentes estratégias de interação, foi adotado um único *prompt* padronizado durante o processo de refatoração assistida por Inteligência Artificial Generativa. A utilização de um comando único teve como objetivo minimizar variáveis experimentais relacionadas à formulação

dos *prompts*, permitindo que as diferenças observadas nos resultados estivessem associadas principalmente às características das ferramentas analisadas, e não às instruções fornecidas pelo usuário. Embora comandos mais específicos possam direcionar a aplicação de determinadas técnicas de refatoração ou produzir resultados distintos, a adoção de um único *prompt* contribuiu para a padronização do experimento e para a isonomia metodológica entre as abordagens avaliadas.

Dessa forma, em todas as interações relacionadas à refatoração dos projetos, foi utilizado o seguinte *prompt* principal:

> "Refatore este código para melhorar manutenibilidade, compreensibilidade, reusabilidade e complexidade estrutural."

Durante a execução do experimento, observou-se que as ferramentas frequentemente propunham não apenas técnicas clássicas de refatoração, mas também modificações relacionadas à segurança, confiabilidade e robustez da aplicação. Embora essas alterações não se enquadrem estritamente na definição clássica de refatoração proposta por Fowler (2020), elas foram mantidas e registradas por representarem melhorias de qualidade frequentemente sugeridas por ferramentas de IA Generativa quando submetidas ao *prompt* utilizado.

Em alguns casos, as alterações propostas resultaram em erros de compilação, execução ou integração entre componentes do sistema. Nessas situações, a intervenção humana restringiu-se à solicitação de correção dos problemas identificados, utilizando um segundo *prompt* padronizado:

> "Resolva os seguintes erros: [erros identificados]."

Não foram fornecidas instruções adicionais relacionadas à arquitetura, implementação ou aplicação de novas técnicas de refatoração além das sugeridas pelas próprias ferramentas. As intervenções realizadas tiveram como único objetivo restaurar o funcionamento esperado dos sistemas para possibilitar a continuidade da análise.

Essa estratégia buscou aumentar a reprodutibilidade do experimento e assegurar condições equivalentes de utilização para todas as ferramentas avaliadas. Entretanto, reconhece-se que diferentes formulações de *prompts* podem influenciar significativamente os resultados obtidos. Por esse motivo, a investigação do impacto de estratégias alternativas de *prompting* constitui uma oportunidade para trabalhos futuros.

2.6 Critérios de Avaliação

Nesta seção são definidos os critérios utilizados para avaliar a eficácia das refatorações aplicadas, tanto manualmente quanto com o auxílio de IA Generativa. A avaliação fundamenta-se em métricas de qualidade de software associadas a atributos como manutenibilidade, compreensibilidade, complexidade e reusabilidade. Os atributos selecionados foram baseados em

estudos de Sommerville (2011) e na aplicação prática apresentada por Riemer (2024), com o objetivo de proporcionar uma análise comparativa clara entre as abordagens de refatoração.

Os atributos de qualidade analisados neste estudo foram: (i) manutenibilidade, que refere-se à facilidade de modificar, corrigir ou evoluir o sistema ao longo do tempo; (ii) compreensibilidade, associada à facilidade de entendimento do código por outros desenvolvedores; (iii) reusabilidade, que corresponde ao potencial de reutilização de funções, métodos ou componentes em diferentes partes do sistema; e (iv) complexidade estrutural, que refere-se ao grau de dificuldade inerente à organização do código.

A avaliação desses atributos foi realizada por meio de métricas objetivas extraídas da ferramenta SonarQube antes e após as refatorações. Complementarmente, foram conduzidas análises qualitativas para identificar aspectos não capturados diretamente pelas métricas automatizadas, tais como reorganização de responsabilidades, modularização do código e alterações estruturais promovidas pelas diferentes abordagens avaliadas.

2.6.1 Métricas Utilizadas

Para garantir a coerência entre os objetivos do estudo e as métricas adotadas, utilizou-se o framework GQM (Goal–Question–Metric), proposto por Basili *et al*. (1994), que estrutura a avaliação da qualidade de software a partir da definição de objetivos, formulação de perguntas e seleção de métricas adequadas para respondê-las. O Quadro 3 apresenta a estrutura de avaliação utilizada neste trabalho.

Quadro 3 – Estrutura de Avaliação segundo o Framework GQM (Goal–Question–Metric)

Objetivo (Goal)	Pergunta (Question)	Métricas (Metric)
Avaliar a manutenibilidade	O código tornou-se mais fácil de modificar e evoluir?	Número de Code Smells; débito técnico; percentual de Código Duplicado.
Avaliar a compreensibilidade	O código tornou-se mais fácil de entender?	Complexidade Cognitiva; tamanho médio dos métodos; observação qualitativa da organização estrutural e modularização.
Avaliar a reusabilidade	O código apresenta maior potencial de reutilização?	Redução de duplicações; quantidade de funções, métodos ou componentes extraídos e reutilizáveis identificados durante a análise qualitativa.
Avaliar a complexidade	O código tornou-se estruturalmente menos complexo?	Complexidade Ciclomática; complexidade Cognitiva.

Fonte: Autoria própria (2026).

As métricas quantitativas foram obtidas por meio da ferramenta SonarQube, permitindo comparar objetivamente os projetos antes e após a aplicação das refatorações. Complementarmente, foram realizadas análises qualitativas para identificar aspectos não capturados dire-

tamente pelas métricas automatizadas, como reorganização de responsabilidades, modularização do código, aplicação de técnicas de refatoração e potencial de reutilização de componentes. Essas análises foram utilizadas para complementar a interpretação dos resultados quantitativos e fornecer uma visão mais abrangente dos impactos produzidos por cada abordagem de refatoração.

2.6.2 Critérios de Comparação

A comparação entre as abordagens foi conduzida em três etapas complementares, permitindo analisar a evolução dos projetos ao longo do processo de refatoração e comparar os resultados obtidos por cada método.

1. Avaliação da refatoração manual: Inicialmente, foram comparadas as métricas coletadas antes e após a aplicação da refatoração manual. Essa etapa teve como objetivo identificar o impacto das técnicas de refatoração sobre os atributos de qualidade analisados.

2. Avaliação da refatoração assistida por IA Generativa: Na segunda etapa, foram comparadas as métricas obtidas antes e após a aplicação das ferramentas GitHub Copilot e Amazon Q Developer. A análise considerou os mesmos atributos e métricas utilizados na avaliação da refatoração manual, permitindo verificar as melhorias promovidas por cada ferramenta.

3. Comparação entre as abordagens: Por fim, realizou-se uma comparação direta entre os resultados finais obtidos pelas abordagens manual e assistidas por IA Generativa, contemplando análises quantitativas e qualitativas.

Essa estrutura permitiu comparar os impactos das diferentes abordagens sobre os atributos de qualidade definidos no estudo, buscando identificar:

- Qual abordagem promoveu maior redução de Code Smells?
- Qual abordagem obteve maior redução na duplicação de código?
- Qual abordagem apresentou maior redução da complexidade estrutural?
- Qual abordagem proporcionou melhores ganhos de modularização e potencial de reutilização de componentes?
- Qual abordagem produziu código mais organizado e compreensível?
- Qual abordagem demandou menor tempo de execução e menor quantidade de intervenções corretivas?

Os resultados obtidos são apresentados e discutidos no capítulo a seguir.

3 RESULTADOS

Este capítulo apresenta os resultados obtidos a partir da aplicação das refatorações manuais e com o uso de ferramentas de Inteligência Artificial Generativa nos projetos selecionados. O objetivo foi evidenciar os impactos dessas abordagens na qualidade do código, com base nas métricas definidas na metodologia.

Inicialmente, na seção 3.1, são descritas as execuções das refatorações, tanto manuais quanto assistidas por IA, destacando as principais intervenções realizadas em cada projeto. Posteriormente, a seção 3.2 realiza a análise dos resultados, na qual são discutidas as diferenças entre as abordagens de refatoração, considerando as métricas definidas e os contextos de cada projeto. Por fim, a seção 3.3 apresenta a discussão dos resultados, abordando a interpretação dos achados e as principais vantagens e desvantagens observadas em cada abordagem.

3.1 Execução das Refatorações

Esta seção apresenta a execução das refatorações realizadas nos projetos selecionados, abrangendo tanto a abordagem manual quanto a assistida por ferramentas de Inteligência Artificial Generativa.

Inicialmente, são descritas as modificações aplicadas manualmente, destacando as principais técnicas empregadas e as decisões tomadas ao longo do processo. Em seguida, são apresentados os resultados obtidos com o uso das ferramentas de IA, evidenciando as sugestões geradas e as alterações implementadas. O objetivo é fornecer uma visão mais detalhada das intervenções realizadas em cada projeto, permitindo uma análise comparativa mais aprofundada nas seções posteriores.

3.1.1 Refatoração Manual

A presente subseção apresenta os resultados da refatoração manual aplicada aos projetos desenvolvidos, com base nos princípios e técnicas conforme descrito por Fowler (2026). As intervenções realizadas tiveram como foco principal a melhoria da legibilidade, redução da complexidade cognitiva, eliminação de código morto, padronização de nomenclaturas e aprimoramento da organização estrutural.

As subseções a seguir apresentam a aplicação dessas refatorações nos projetos Memoletrando, Querida Floresta 2.0 e SpeakSnake, evidenciando as principais melhorias realizadas em cada tecnologia envolvida.

3.1.1.1 Memoletrando

A refatoração manual do sistema foi conduzida com base na análise de problemas estruturais identificados previamente, com foco na melhoria da legibilidade, redução de comple-

xidade e adequação a boas práticas de desenvolvimento. As alterações abrangeram todas as camadas da aplicação (Laravel, Angular e Ionic), priorizando a eliminação de código obsoleto, padronização de nomenclaturas e reorganização de responsabilidades, seguindo as boas práticas descritas por Fowler (2026).

No *back-end* em Laravel, as modificações concentraram-se principalmente na remoção de código morto em diversos módulos, na substituição de valores literais por constantes e variáveis semânticas e na melhoria da organização lógica de métodos. Técnicas como *Extract Function*, *Replace Magic Number/Literal*, *Rename Method/Variable* e *Replace Nested Conditional with Guard Clauses* foram amplamente aplicadas, especialmente em validações como CPF e CNPJ e em serviços centrais como ApiService e DataTableService. Além disso, houve reestruturação de classes consideradas excessivamente grandes (God Classes), por meio de *Extract Class* e *Split Phase*, reduzindo acoplamento e melhorando a modularidade.

No *front-end* Angular, a refatoração priorizou a simplificação de *templates* e componentes, com foco na redução de complexidade cognitiva e melhoria da clareza do código. Foram aplicadas técnicas como *Extract Method*, *Introduce Explaining Variable*, *Replace Loop with Pipeline* e *Encapsulate Variable*, além da eliminação de código não utilizado. Também houve melhorias na organização de serviços e no tratamento de requisições, com divisão de responsabilidades em métodos menores e mais coesos. Adicionalmente, foram realizadas melhorias de acessibilidade e padronização de *bindings*, contribuindo para maior qualidade estrutural do código.

Por fim, no Ionic, as alterações focaram tanto em aspectos estruturais quanto em design de código, incluindo a remoção de duplicações, reorganização de estilos e melhoria na separação entre lógica e apresentação. Técnicas como *Decompose Conditional*, *Extract Function*, *Encapsulate Field*, *Replace Primitive with Object* e *Separate Query from Modifier* foram utilizadas para tornar o código mais modular e compreensível. Também houve padronização de nomenclatura, eliminação de métodos redundantes e adoção de práticas modernas da linguagem, como uso de *const/let* e tipagem mais explícita. No geral, a refatoração manual resultou em um código mais organizado, com menor complexidade e melhor alinhado a boas práticas de engenharia de software.

O Quadro 4 apresenta um resumo das técnicas de refatoração manual aplicadas no sistema Memoletrando, bem como os principais arquivos e módulos em que essas técnicas foram empregadas.

3.1.1.2 Querida Floresta 2.0

A refatoração manual do projeto foi conduzida também com foco na melhoria da legibilidade, organização estrutural e redução da complexidade dos *scripts* em C#. Inicialmente, foram identificados trechos com alta complexidade ciclomática, uso excessivo de condicionais aninhadas, presença de valores mágicos e responsabilidades concentradas em métodos extensos. A partir desse diagnóstico, aplicaram-se técnicas clássicas de refatoração com o objetivo

Quadro 4 – Técnicas de Refatoração Aplicadas no Memoletrando

Técnica de Refatoração	Arquivos / Módulos Aplicados
Remove Dead Code	Múltiplos módulos Laravel, Angular e Ionic
Extract Function / Extract Method	ValidCPF.php; ValidCNPJ.php; ApiService.php; DataTableService.php; componentes Angular; serviços Ionic
Replace Magic Number / Literal	Arquivos de configuração Laravel; Helpers; Services; arquivos SCSS
Rename Method / Rename Variable	Componentes e serviços Angular; componentes Ionic; módulos Laravel
Replace Nested Conditional with Guard Clauses	Validadores (CPF/CNPJ); serviços Laravel, Angular e Ionic
Extract Class	ApiService.php; DataTableService.php
Decompose Conditional	Serviços e componentes Angular e Ionic
Replace Loop with Pipeline	Serviços Angular e Laravel
Encapsulate Variable / Field	Serviços e componentes em todas as camadas
Consolidate Conditional Expression	Templates Angular e serviços
Split Phase	ApiService; DataTableService; organização de serviços e estilos
Replace Primitive with Object	Modelos e serviços no Ionic
Separate Query from Modifier	Serviços Angular e Ionic
Introduce Explaining Variable / Comment	Diversos módulos para melhoria de legibilidade

Fonte: Autoria própria (2026).

de preservar o comportamento original do sistema enquanto se promoviam maior clareza e modularização do código.

O processo de refatoração manual concentrou-se em aprimorar a organização estrutural, segurança da informação, coesão das classes e redução de duplicação de código, preservando o comportamento original do sistema.

Nos *scripts* relacionados ao sistema de chuva (como *BaseRainScript.cs*, *RainCollision.cs* e *RainScript2D.cs*), a refatoração concentrou-se na decomposição de métodos extensos em funções menores e mais descritivas, bem como na substituição de números mágicos por constantes simbólicas. Também foram introduzidas variáveis explicativas para tornar condições mais compreensíveis e aplicadas *guard clauses* para reduzir níveis de aninhamento. Adicionalmente, houve remoção de código morto e encapsulamento de variáveis, contribuindo para uma estrutura mais limpa e de fácil manutenção.

Na camada de dados e integração (incluindo *Credentials.cs*, *GoogleSheetsController.cs*, *Player.cs* e *PlayerData.cs*), as modificações focaram na separação de responsabilidades e melhor organização arquitetural. Foram aplicadas técnicas como *Extract Class*, *Encapsulate Field* e *Move Class into Namespace*, além da substituição de literais por constantes nomeadas e

introdução de validações para evitar estados inválidos. Essas mudanças resultaram em maior controle sobre o acesso a dados sensíveis e melhor coesão entre componentes.

Nos controladores de cena e interface (abrangendo múltiplas cenas como *Scene1*, *Scene2*, *Scene3*, *Scene4*, *Scene5*, *Scene8* e *Scene9*), a refatoração priorizou a padronização de nomenclaturas, extração de métodos, eliminação de duplicações e organização em *namespaces*. Também foram aplicadas técnicas como *Replace Conditional with Polymorphism*, *Extract Method* e *Encapsulate Field*, além da introdução de constantes simbólicas e separação entre lógica de controle e efeitos colaterais. Essas alterações reduziram a complexidade estrutural e facilitaram a compreensão do fluxo de execução. O Quadro 5 apresenta um resumo das principais técnicas de refatoração aplicadas e os respectivos arquivos afetados.

3.1.1.3 SpeakSnake

A refatoração manual do projeto foi realizada com foco na modernização do código, melhoria da legibilidade e aumento da robustez, mantendo o comportamento original da aplicação. Foram identificados problemas como código morto, uso de práticas desatualizadas (por exemplo, `var` em JavaScript), dependências frágeis (como seleção fixa de índices) e ausência de validações em pontos críticos. A partir desse diagnóstico, foram aplicadas modificações estruturais e sintáticas para adequar o código a padrões mais atuais e reduzir riscos de execução.

Nos arquivos JavaScript, especialmente em *script.js*, a refatoração concentrou-se na substituição de construções antigas por práticas modernas, como o uso de `let` e `const`, além da correção de erros lógicos e melhoria na manipulação de estados. Também foram adicionadas verificações de existência de elementos do DOM, reduzindo a probabilidade de falhas em tempo de execução. A seleção de voz foi tornada mais robusta, eliminando dependência de índices fixos, e o fluxo de execução foi ajustado com *guard clauses*, garantindo interrupção correta do processamento em condições específicas. Além disso, houve remoção de código morto e limpeza de comentários desnecessários.

Na camada de interface e estilo (HTML e CSS), as alterações envolveram principalmente a organização estrutural e eliminação de redundâncias. No *style.css*, foram removidos duplicações e comentários inúteis, além da introdução de variáveis CSS para substituir valores mágicos. O código foi reorganizado em blocos lógicos (*reset*, layout, componentes), melhorando a manutenibilidade. Nos arquivos PHP de interface, como *game.php* e *index.php*, foram realizadas melhorias de acessibilidade (como uso de `aria-live`), tratamento seguro de dados (com `htmlspecialchars`) e reorganização de trechos condicionais com foco em clareza e segurança.

Na camada de *back-end* e integração (arquivos PHP como *Database.php*, *Login.php*, *register.php* e ações AJAX), a refatoração priorizou a separação de responsabilidades, encapsulamento e tratamento adequado de erros. Foram introduzidas classes para centralizar lógica de conexão e autenticação, substituindo abordagens procedurais. Também foram aplicadas técnicas como uso de *prepared statements*, substituição de códigos de erro por exceções e en-

Quadro 5 – Técnicas de Refatoração Aplicadas no Querida Floresta 2.0

Técnica de Refatoração	Arquivos / Módulos Aplicados
Extract Method / Extract Function	BaseRainScript.cs; RainScript2D.cs; Scene controllers diversos (Scene1 a Scene9); Canvas controllers
Introduce Explaining Variable	BaseRainScript.cs; Scene controllers; Canvas controllers
Replace Magic Number with Symbolic Constant	BaseRainScript.cs; RainCollision.cs; Scene controllers; Canvas controllers
Replace Nested Conditional with Guard Clauses	BaseRainScript.cs; Scene controllers; DragAndDrop scripts
Remove Dead Code	RainCollision.cs; PlayerData.cs; Scene controllers; métodos não utilizados
Encapsulate Field / Variable	Player.cs; PlayerData.cs; Credentials.cs; controllers diversos
Move Class into Namespace	Controllers de cena; scripts de UI; classes de dados e integração
Extract Class	PlayerData.cs; Credentials.cs
Rename Method / Rename Variable	Scene controllers; Canvas controllers; Narrator controllers
Consolidate Conditional Expression	BaseRainScript.cs; RainScript2D.cs; lógica de UI
Make Method Static	GoogleSheetsController.cs; Scene controllers
Introduce Assertion	Credentials.cs; Player.cs; controllers com dependências de Inspector
Replace Conditional with Polymorphism	DragAndDrop.cs
Extract Interface	DragAndDrop.cs
Encapsulate Collection	PlayerData.cs
Substitute Algorithm	PlayerData.cs (reset de dados com loop)
Separate Query from Modifier	Scene controllers (lógica de jogo e efeitos colaterais)
Inline Temp	Trechos diversos de controllers
Introduce Parameter Object	Credentials.cs; PlayerData.cs

Fonte: Autoria própria (2026).

capsulamento de dados de entrada em estruturas apropriadas. Adicionalmente, arquivos foram renomeados e organizados em *namespaces*, contribuindo para melhor modularização do sistema. O Quadro 6 apresenta um resumo das principais técnicas de refatoração aplicadas e os respectivos arquivos afetados no jogo SpeakSnake.

Quadro 6 – Técnicas de Refatoração Aplicadas no SpeakSnake

Técnica de Refatoração	Arquivos / Módulos Aplicados
Remove Dead Code	script copy.js; script.js; style.css; game.php; index.php
Extract Function / Extract Method	index.php; register.php; Login.php; arquivos de ações AJAX
Replace Magic Number / Literal	script.js; style.css
Rename Method / Rename Variable	Arquivos JavaScript e PHP diversos
Replace Nested Conditional with Guard Clauses	script.js; game.php; index.php; logout.php
Extract Class	Database.php; Login.php
Encapsulate Variable / Field	Database.php; Login.php; register.php
Replace Temp with Query	script.js; backend PHP
Introduce Assertion / Validation	script.js; register.php; Database.php
Replace Error Code with Exception	Database.php; Login.php; arquivos AJAX
Introduce Parameter Object	register.php; arquivos AJAX
Separate Query from Modifier	index.php; arquivos AJAX
Consolidate Duplicate Conditional Fragments	style.css
Replace Inline Code with Function Call	CSS e trechos PHP
Replace Include with Namespace Import	index.php; register.php; arquivos AJAX
Remove Closing PHP Tag	Arquivos AJAX
Move/Rename Files (organização estrutural)	conexao.php → Database.php; selecao.php → menu.php; arquivos AJAX renomeados.

Fonte: Autoria própria (2026).

3.1.2 Refatoração por IA Generativa

A segunda etapa do estudo consistiu na aplicação de ferramentas de Inteligência Artificial Generativa aos projetos selecionados, utilizando o GitHub Copilot e o Amazon Q Developer. O objetivo foi comparar os resultados obtidos por essas ferramentas com aqueles alcançados na refatoração manual, considerando os atributos de qualidade definidos na metodologia: manutenibilidade, compreensibilidade, reusabilidade e complexidade estrutural.

Para garantir a comparabilidade entre as abordagens, ambas as ferramentas receberam o mesmo *prompt* padronizado, descrito na subseção 2.5.1, e foram aplicadas aos mesmos arquivos previamente submetidos à refatoração manual. Além da análise das melhorias promovidas pelas ferramentas, foram registradas as intervenções necessárias para correção de erros de compilação, execução ou integração decorrentes das alterações geradas.

Durante a execução dos experimentos, observou-se que as ferramentas aplicaram tanto técnicas clássicas de refatoração, como extração de métodos, redução de duplicações, reorganização de responsabilidades e simplificação estrutural, quanto modificações voltadas à robustez, segurança e confiabilidade do sistema. Embora essas últimas não se enquadrem estritamente na definição clássica de refatoração proposta por Fowler (2020), elas foram mantidas e registradas por representarem alterações frequentemente sugeridas por ferramentas de Inteligência Artificial Generativa quando submetidas ao *prompt* utilizado neste estudo.

O Quadro 7 apresenta um resumo das principais modificações realizadas por cada ferramenta nos projetos analisados, distinguindo melhorias estruturais relacionadas à refatoração de alterações complementares voltadas à robustez e qualidade geral do software. Nas subseções seguintes, são detalhados os resultados obtidos em cada projeto, bem como as dificuldades encontradas, os erros gerados pelas ferramentas e os impactos observados nas métricas de qualidade avaliadas.

Quadro 7 – Resumo das Refatorações Realizadas com IA Generativa

Ferramenta	Projeto	Principais Modificações Realizadas
GitHub Copilot	Memoletrando	Extração de métodos, redução de duplicações, reorganização de trechos de código, melhoria da modularização e padronização estrutural. Adicionalmente, foram sugeridas melhorias de tipagem e tratamento de erros.
Amazon Q Developer	Memoletrando	Eliminação de código morto, reorganização de componentes, redução de duplicações e padronização estrutural. Também foram sugeridas modificações relacionadas à segurança da aplicação.
GitHub Copilot	Querida Floresta 2.0	Extração de funções, simplificação de estruturas condicionais, padronização de nomenclaturas, reorganização de *scripts* e redução da complexidade estrutural.
Amazon Q Developer	Querida Floresta 2.0	Reorganização arquitetural, centralização de responsabilidades em componentes específicos, redução de duplicações e melhoria da modularização do projeto.
GitHub Copilot	SpeakSnake	Redução de duplicações, reorganização de responsabilidades, modularização de *scripts* JavaScript e melhoria da legibilidade do código. Também foram sugeridas alterações relacionadas à validação de dados e robustez da aplicação.
Amazon Q Developer	SpeakSnake	Reorganização estrutural do projeto, padronização arquitetural, modularização de componentes e redução de duplicações. Adicionalmente, foram propostas melhorias relacionadas à configuração do ambiente e validações de entrada.

Fonte: Autoria própria (2026).

3.1.2.1 Memoletrando

A refatoração do jogo Memoletrando com o GitHub Copilot foi realizada em aproximadamente 2 horas e 15 minutos, utilizando uma abordagem iterativa baseada na aplicação do *prompt* padronizado sobre conjuntos de arquivos organizados por módulos. Essa estratégia permitiu que a ferramenta analisasse o contexto de cada parte do sistema antes de propor modificações, favorecendo intervenções graduais na estrutura do código.

As principais alterações realizadas pelo GitHub Copilot estiveram relacionadas à melhoria da manutenibilidade, compreensibilidade e redução da complexidade estrutural. Entre as modificações observadas destacam-se a extração de métodos auxiliares, reorganização de funções extensas em unidades menores e mais coesas, eliminação de código morto, remoção de *imports* não utilizados e redução de trechos duplicados. Também foram realizadas melhorias na organização dos componentes e na padronização de fluxos recorrentes da aplicação, contribuindo para uma estrutura mais modular e de mais fácil manutenção.

Além das modificações associadas às técnicas clássicas de refatoração, a ferramenta também sugeriu alterações relacionadas à robustez do sistema, incluindo aprimoramentos de tipagem, reorganização de fluxos assíncronos, tratamento de exceções e ajustes em validações. Essas modificações foram registradas separadamente, uma vez que não se enquadram estritamente na definição clássica de refatoração, mas influenciam atributos de qualidade do software.

De forma geral, o GitHub Copilot apresentou uma abordagem incremental, realizando alterações distribuídas entre diferentes módulos do projeto e demonstrando boa capacidade de identificar padrões repetitivos e oportunidades de modularização. Entretanto, algumas sugestões introduziram inconsistências que exigiram validações adicionais e intervenções para correção de erros de compilação e integração entre componentes.

A refatoração utilizando o Amazon Q Developer foi concluída em aproximadamente 1 hora e adotou uma estratégia mais abrangente, baseada na análise de conjuntos maiores de arquivos e na proposição de alterações estruturais de maior alcance. Diferentemente da abordagem incremental observada no GitHub Copilot, o Amazon Q Developer concentrou suas sugestões em reorganizações mais extensas da arquitetura do sistema.

No contexto das técnicas de refatoração, destacaram-se a eliminação de código morto, redução de duplicações, simplificação de estruturas condicionais, extração de métodos auxiliares e reorganização de componentes e serviços. A ferramenta também promoveu padronizações estruturais em diferentes partes da aplicação, contribuindo para a redução da complexidade cognitiva e para uma distribuição mais consistente das responsabilidades entre os componentes do sistema.

Assim como observado no GitHub Copilot, o Amazon Q Developer também sugeriu alterações que extrapolam o escopo tradicional da refatoração, incluindo modificações relacionadas à configuração da aplicação, validação de entradas, tratamento de exceções e outros aspectos

associados à robustez e confiabilidade do software. Essas alterações foram consideradas durante a análise, mas avaliadas separadamente das técnicas clássicas de refatoração.

De maneira geral, o Amazon Q Developer demonstrou uma abordagem mais sistemática e orientada à reorganização estrutural da aplicação, promovendo alterações de maior abrangência em comparação ao GitHub Copilot. Embora essas modificações tenham contribuído para a melhoria da organização do código, também demandaram validações adicionais e correções pontuais para garantir a compatibilidade das alterações com o comportamento esperado do sistema.

3.1.2.2 Querida Floresta 2.0

A refatoração do projeto Querida Floresta 2.0 com o GitHub Copilot foi realizada ao longo de aproximadamente 5 horas, utilizando a pasta `Project/Scripts` como principal contexto de análise. O processo ocorreu de forma incremental, com aplicação do *prompt* padronizado em arquivos específicos, estratégia que permitiu maior controle das alterações em um projeto Unity caracterizado pela forte interdependência entre *scripts*. Inicialmente, foram priorizados os componentes centrais da lógica do jogo e, posteriormente, os controladores de cena, interface e sistemas auxiliares.

As modificações realizadas pelo GitHub Copilot concentraram-se principalmente em melhorias relacionadas à manutenibilidade, compreensibilidade e redução da complexidade estrutural. Entre as alterações observadas destacam-se a padronização de nomenclaturas, extração de funções, decomposição de métodos extensos em unidades menores e mais coesas, reorganização de responsabilidades entre classes e simplificação de fluxos de execução relacionados ao gerenciamento de estados do jogo. Também foram identificadas reduções de duplicação de código em controladores de cena e componentes de interface.

Além das técnicas clássicas de refatoração, a ferramenta sugeriu modificações voltadas à robustez da aplicação, incluindo verificações adicionais de valores nulos, tratamento de exceções e ajustes no gerenciamento de recursos da Unity. Essas alterações foram registradas separadamente das atividades de refatoração, por não estarem diretamente associadas à definição clássica proposta por Fowler (2020).

De maneira geral, o GitHub Copilot apresentou uma abordagem gradual e controlada, favorecendo a compreensão do impacto de cada alteração e reduzindo riscos de incompatibilidade entre componentes. Embora essa estratégia tenha demandado maior acompanhamento durante o processo, ela contribuiu para uma integração mais previsível das modificações realizadas.

A refatoração utilizando o Amazon Q Developer também foi conduzida em aproximadamente 5 horas e empregou o mesmo *prompt* utilizado no GitHub Copilot. Diferentemente da abordagem incremental, a ferramenta iniciou o processo propondo alterações estruturais de maior abrangência, com foco na reorganização arquitetural e na redistribuição de responsabilidades entre os componentes do sistema.

Durante a execução do experimento, observou-se que o Amazon Q Developer apresentou dificuldades ocasionais na identificação do contexto correto dos arquivos analisados. Em algumas situações, a ferramenta deixou de considerar arquivos relevantes ou direcionou sugestões para componentes diferentes dos especificados. Para minimizar esse comportamento, foi necessário utilizar referências explícitas aos arquivos e realizar múltiplas interações até a obtenção de resultados consistentes.

As principais modificações realizadas estiveram relacionadas à reorganização estrutural do projeto. Destacam-se a centralização de responsabilidades em componentes específicos, a consolidação de funcionalidades repetidas, a extração de métodos auxiliares, a redução de duplicações e a reorganização de *scripts* visando melhorar a modularização do sistema. A ferramenta também promoveu simplificações em estruturas condicionais e reduções significativas no tamanho de alguns arquivos por meio da redistribuição de responsabilidades entre componentes.

Assim como ocorreu no GitHub Copilot, o Amazon Q Developer também sugeriu alterações complementares relacionadas à robustez da aplicação, incluindo validações adicionais e tratamento de exceções. Essas modificações foram consideradas durante a análise, mas avaliadas separadamente das técnicas de refatoração propriamente ditas.

Em síntese, o Amazon Q Developer apresentou uma abordagem mais orientada à reorganização global da arquitetura do projeto, promovendo alterações de maior abrangência estrutural do que aquelas observadas no GitHub Copilot. Apesar das dificuldades relacionadas ao controle de contexto e à necessidade de interações adicionais, a ferramenta demonstrou potencial para identificar oportunidades de modularização e reorganização arquitetural em larga escala. Entretanto, as modificações propostas exigiram validações contínuas e correções pontuais para assegurar a compatibilidade das alterações com o comportamento esperado da aplicação.

3.1.2.3 SpeakSnake

A refatoração do projeto SpeakSnake com o GitHub Copilot foi conduzida de forma incremental, utilizando a aplicação de um mesmo *prompt* em cada arquivo individualmente. O processo teve duração aproximada de 35 minutos e ocorreu por meio da análise contextual de cada componente do sistema, permitindo que a ferramenta sugerisse modificações localizadas e progressivas, voltadas à melhoria de segurança, confiabilidade, manutenibilidade e redução de complexidade.

As alterações promovidas pelo Copilot concentraram-se principalmente em melhorias de segurança e organização estrutural do código. Entre as principais modificações, destacam-se a substituição de consultas diretas ao banco de dados por *prepared statements*, a introdução de validações de entrada, melhorias no gerenciamento de sessões e recomendações para uso adequado de *hashing* de senhas. Também foram aplicadas técnicas de sanitização de saída para mitigação de vulnerabilidades como XSS. Apesar dessas melhorias, observou-se que algumas

práticas inseguras permaneceram inicialmente no código, como a manutenção de credenciais diretamente nos arquivos, exigindo ajustes complementares posteriores.

No aspecto de manutenibilidade, o Copilot reorganizou diferentes partes da aplicação, promovendo separação mais clara de responsabilidades entre PHP, HTML, CSS e JavaScript. Houve redução de duplicações, simplificação de lógicas condicionais e modularização de *scripts* JavaScript, além de melhorias na organização de variáveis, funções e validações. Essas alterações contribuíram para tornar o código mais legível, consistente e de mais fácil manutenção, ainda que preservando, em grande parte, a arquitetura original do sistema.

De maneira geral, o GitHub Copilot demonstrou eficiência na aplicação de melhorias pontuais e na correção de problemas evidentes distribuídos ao longo dos arquivos. A abordagem incremental favoreceu maior controle sobre as modificações realizadas, embora tenha limitado intervenções arquiteturais mais profundas e abrangentes no sistema.

A refatoração com o Amazon Q Developer foi realizada em aproximadamente 50 minutos, adotando uma abordagem mais abrangente e centralizada. Diferentemente do comportamento incremental observado no Copilot, a ferramenta realizou análises completas dos arquivos antes da aplicação das modificações, frequentemente propondo reestruturações mais amplas, incluindo reorganização significativa do código e criação de novos componentes.

No aspecto de segurança, o Amazon Q promoveu melhorias mais robustas e sistemáticas, como remoção de credenciais do código-fonte por meio do uso de variáveis de ambiente, implementação de validações rigorosas de entrada, padronização de respostas seguras e eliminação da exposição de erros sensíveis. Também foram incorporados mecanismos de sanitização de dados, controle adequado de sessões e uso consistente de *prepared statements*, abrangendo diferentes camadas da aplicação.

Em relação à organização estrutural e manutenibilidade, a ferramenta realizou modificações mais profundas, incluindo encapsulamento de funcionalidades em classes, criação de funções reutilizáveis, padronização de respostas JSON e separação mais clara de responsabilidades. Além disso, foram reorganizados arquivos CSS e JavaScript, promovendo modularização e maior uniformidade arquitetural ao longo do sistema.

Em síntese, o Amazon Q Developer apresentou uma abordagem mais orientada à reestruturação global do sistema, promovendo ganhos relevantes em segurança, padronização e organização arquitetural. Entretanto, o caráter mais abrangente das alterações reduziu a granularidade de controle sobre as mudanças quando comparado à estratégia incremental utilizada pelo GitHub Copilot, tornando necessária maior validação das modificações realizadas.

3.2 Análise Comparativa dos Resultados

A presente seção tem como objetivo analisar comparativamente os resultados obtidos a partir das diferentes abordagens de refatoração aplicadas aos projetos avaliados. A análise contempla tanto a refatoração manual quanto o uso de ferramentas de Inteligência Artificial Generativa, considerando métricas de qualidade de software relacionadas à segurança, confia-

bilidade, manutenibilidade, duplicação, tamanho e complexidade. Esta seção prioriza uma abordagem interpretativa dos resultados, enquanto as tabelas completas com os dados coletados encontram-se disponíveis no Apêndice A.

Inicialmente, são discutidos os efeitos das refatorações em cada projeto individualmente, permitindo observar como cada abordagem influencia a evolução das métricas em diferentes contextos. Em seguida, são realizadas comparações diretas entre a refatoração manual e as abordagens baseadas em IA, com o objetivo de identificar vantagens, limitações e padrões de comportamento associados a cada estratégia.

Posteriormente, a análise é aprofundada por meio da comparação entre as ferramentas de IA Generativa utilizadas, avaliando suas diferenças de desempenho em cenários distintos. Por fim, são apresentadas comparações entre os projetos, evidenciando como fatores como porte, arquitetura e complexidade inicial influenciam os resultados obtidos. Dessa forma, esta seção busca oferecer uma visão abrangente sobre a eficácia das diferentes abordagens de refatoração, contribuindo para a compreensão de seus impactos na qualidade do código e no processo de manutenção de software.

3.2.1 Análise Individual dos Projetos: Comparação Antes e Após as Refatorações

Esta seção apresenta a comparação entre os dados coletados antes e após a aplicação das técnicas de refatoração nos projetos analisados. O objetivo é evidenciar as mudanças observadas nas métricas de qualidade de código, permitindo identificar os impactos diretos das refatorações realizadas. A análise considera aspectos como manutenibilidade, compreensibilidade, reusabilidade e complexidade, possibilitando avaliar a efetividade das modificações aplicadas em cada projeto.

3.2.1.1 Memoletrando

A análise dos resultados obtidos no projeto Memoletrando evidencia diferenças significativas entre as abordagens de refatoração manual e assistida por Inteligência Artificial Generativa. De forma geral, a refatoração manual apresentou os melhores resultados globais, enquanto as ferramentas de IA demonstraram desempenhos distintos, com ganhos mais localizados em determinados critérios.

A refatoração manual promoveu melhorias expressivas em praticamente todas as métricas avaliadas. Em confiabilidade, houve eliminação completa dos *issues*, reduzindo-os de 143 para 0, além da remoção total do esforço estimado de correção. Na dimensão de manutenibilidade, a quantidade de *issues* caiu de 896 para apenas 5, evidenciando uma redução substancial de problemas estruturais e de código potencialmente problemático.

Também foram observados ganhos relevantes relacionados à duplicação de código. A densidade de duplicação foi reduzida de 9,1% para 5,2%, acompanhada da diminuição no número de linhas, blocos e arquivos duplicados. Esses resultados indicam maior reaproveitamento

de componentes e melhor consolidação de trechos redundantes, contribuindo para uma estrutura mais consistente e menos suscetível a erros futuros.

No aspecto estrutural, as alterações foram mais moderadas. Houve redução nas linhas de código, passando de 24.253 para 22.734, além de diminuição no número total de arquivos do sistema. A proporção de comentários permaneceu estável em aproximadamente 5,8%, indicando que as modificações realizadas não comprometeram a documentação existente.

Em relação à complexidade, a abordagem manual apresentou melhora tanto na complexidade ciclomática quanto na cognitiva. A primeira foi reduzida de 2.875 para 2.715, enquanto a segunda apresentou queda mais acentuada, passando de 1.245 para 844. Esses resultados sugerem um código mais compreensível, modular e de mais fácil manutenção.

A refatoração realizada com o GitHub Copilot apresentou resultados mais heterogêneos. Embora tenham sido observadas melhorias em duplicação e manutenibilidade, a ferramenta também introduziu regressões em aspectos importantes. Na dimensão de confiabilidade, por exemplo, houve aumento no número de *issues*, passando de 143 para 153, sem redução no esforço estimado de correção, o que sugere a introdução de novos problemas durante o processo de refatoração.

Em manutenibilidade, observou-se redução moderada de *issues*, de 896 para 808, acompanhada de pequena diminuição no esforço de correção. Entretanto, os valores permaneceram elevados quando comparados à abordagem manual, indicando que as melhorias estruturais promovidas foram limitadas.

A redução da duplicação foi um dos pontos positivos da ferramenta. A densidade caiu de 9,1% para 5,8%, demonstrando capacidade de consolidar trechos redundantes. Contudo, essa melhoria ocorreu simultaneamente ao crescimento expressivo da base de código, que passou de 24.253 para 29.374 linhas. Isso sugere que parte da redução percentual decorreu da expansão geral do sistema, e não exclusivamente da eliminação efetiva de duplicações.

Quanto à complexidade, os resultados mostraram comportamento misto. A complexidade ciclomática aumentou de 2.875 para 2.961, indicando maior complexidade estrutural do fluxo de execução, enquanto a complexidade cognitiva apresentou leve redução, passando de 1.305 para 1.262. Em síntese, o GitHub Copilot demonstrou capacidade de aplicar melhorias pontuais, porém sem manter consistência global nos critérios avaliados.

O Amazon Q Developer apresentou desempenho intermediário entre as abordagens analisadas. Em confiabilidade, houve leve piora, com aumento de *issues* de 143 para 147, mantendo-se o esforço de correção. Apesar disso, a ferramenta apresentou resultados mais consistentes nos critérios relacionados à organização estrutural e redução de complexidade.

Na dimensão de manutenibilidade, houve redução de *issues* de 896 para 716, acompanhada de diminuição no esforço de correção. Embora os resultados ainda permaneçam distantes dos obtidos manualmente, observa-se melhora estrutural mais significativa do que aquela alcançada pelo GitHub Copilot.

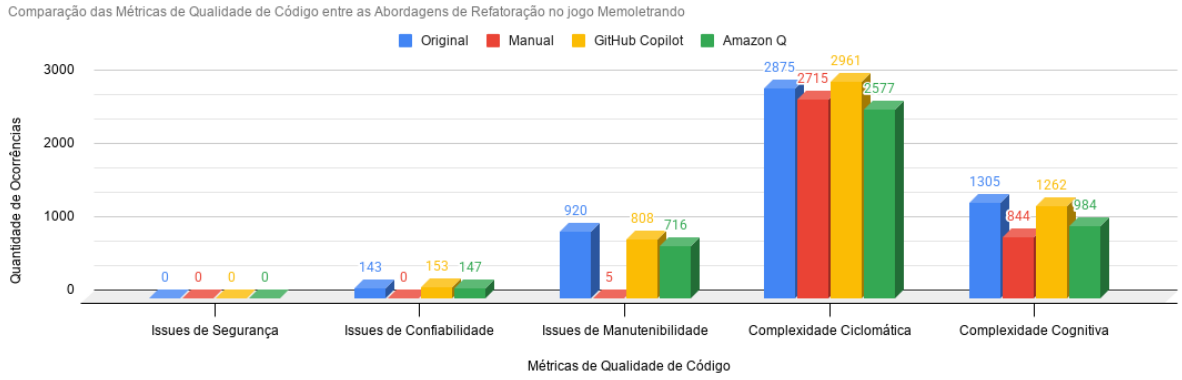
O principal destaque do Amazon Q foi a redução de duplicação de código. A densidade caiu de 8,9% para 4,0%, representando o melhor resultado entre todas as abordagens avalia-

das. Também foram observadas reduções expressivas em linhas, blocos e arquivos duplicados, evidenciando forte atuação da ferramenta na consolidação de código redundante.

Além disso, o Amazon Q promoveu redução geral no tamanho e na complexidade do sistema. Houve diminuição nas linhas de código, no número de declarações e funções, acompanhada da redução da complexidade ciclomática, de 2.875 para 2.577, e da complexidade cognitiva, de 1.305 para 984. Esses resultados indicam um sistema estruturalmente mais enxuto e menos complexo.

A Figura 2 apresenta a comparação das principais métricas de qualidade entre as abordagens analisadas. O gráfico evidencia visualmente a superioridade da refatoração manual na redução de *issues* de segurança, confiabilidade e manutenibilidade, além de apresentar o menor índice de complexidade cognitiva. Também é possível observar que o Amazon Q obteve desempenho intermediário, enquanto o GitHub Copilot apresentou resultados menos consistentes, especialmente em complexidade.

Figura 2 – Comparação das Métricas de Qualidade de Código entre as Abordagens de Refatoração no jogo Memoletrando

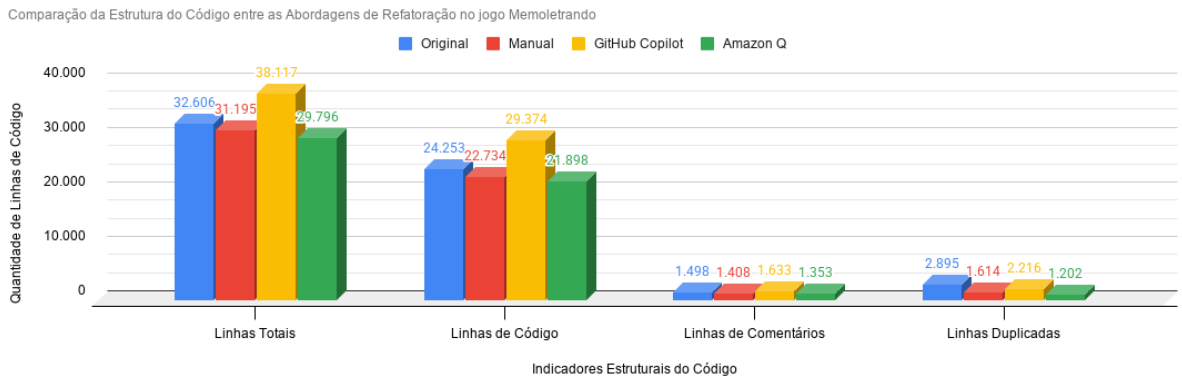


Fonte: Autoria própria (2026).

Complementarmente, a Figura 3 apresenta a comparação das métricas relacionadas à estrutura do código, incluindo linhas de código, linhas totais, linhas de comentário e linhas duplicadas. Observa-se que a abordagem manual reduziu o tamanho geral do sistema sem comprometer a proporção de comentários, enquanto o Amazon Q promoveu redução simultânea de linhas totais e duplicadas. Em contrapartida, o GitHub Copilot apresentou crescimento significativo na quantidade de linhas de código, evidenciando expansão estrutural da aplicação mesmo com redução percentual da duplicação.

De forma comparativa, observa-se que a refatoração manual foi a abordagem mais eficaz no projeto Memoletrando, promovendo melhorias abrangentes em confiabilidade, manutenibilidade, duplicação e complexidade. Entre as ferramentas de IA, o Amazon Q Developer apresentou os resultados mais consistentes, especialmente na redução de duplicação e simplificação estrutural, enquanto o GitHub Copilot demonstrou comportamento mais instável, com melhorias pontuais acompanhadas de regressões em critérios relevantes.

Figura 3 – Comparação da Estrutura do Código entre as Abordagens de Refatoração no jogo Memoletrando



Fonte: Autoria própria (2026).

3.2.1.2 Querida Floresta 2.0

Os resultados obtidos no projeto Querida Floresta 2.0 evidenciam diferenças expressivas entre as abordagens de refatoração analisadas. De maneira geral, a refatoração manual apresentou desempenho significativamente superior, promovendo melhorias consistentes em praticamente todos os indicadores de qualidade avaliados. Em contraste, as abordagens com Inteligência Artificial Generativa demonstraram comportamentos distintos, com ganhos pontuais acompanhados, em alguns casos, por regressões estruturais relevantes.

A refatoração manual foi a única abordagem capaz de eliminar completamente os *issues* relacionados à segurança, confiabilidade e manutenibilidade. Além da redução total dos problemas identificados, o esforço estimado de correção também foi reduzido a zero, indicando uma base de código mais estável, segura e de mais fácil evolução futura.

Os ganhos relacionados à duplicação de código também foram expressivos. A densidade de duplicação foi reduzida de 5,8% para 0,0%, eliminando completamente linhas, blocos e arquivos duplicados. Esse resultado sugere uma reestruturação eficiente baseada na centralização de responsabilidades e no reaproveitamento adequado de componentes, contribuindo para maior consistência e facilidade de manutenção.

Em termos estruturais, observou-se aumento moderado em métricas como linhas de código, quantidade de funções, classes e comentários. Contudo, esse crescimento foi acompanhado por uma elevação significativa na proporção de comentários, que passou de 3,7% para 7,5%, indicando maior nível de documentação e potencial melhoria na compreensibilidade do sistema.

No aspecto de complexidade, os resultados apresentaram comportamento misto. A complexidade ciclomática aumentou de 748 para 847, possivelmente em decorrência da introdução de novas abstrações e estruturas auxiliares. Entretanto, a complexidade cognitiva foi reduzida de forma significativa, passando de 471 para 307, indicando que o código se tornou mais compreensível e organizado do ponto de vista humano, mesmo com aumento estrutural moderado.

A refatoração conduzida com o GitHub Copilot apresentou melhorias parciais, porém acompanhadas de efeitos colaterais relevantes. Em segurança, os resultados permaneceram praticamente inalterados, enquanto na confiabilidade houve redução expressiva dos *issues*, passando de 65 para apenas 1. Esse comportamento demonstra a capacidade da ferramenta de corrigir problemas funcionais e inconsistências operacionais presentes no sistema.

Na dimensão de manutenibilidade, observou-se redução significativa no número de *issues*, de 540 para 196, acompanhada de diminuição no esforço estimado de correção. Apesar da melhora, os valores permaneceram elevados quando comparados à abordagem manual, indicando limitações na capacidade da ferramenta de promover uma reorganização estrutural mais profunda.

Por outro lado, a abordagem com Copilot resultou em crescimento expressivo da base de código. As linhas de código aumentaram de 3.764 para 4.777, acompanhadas por expansão nas declarações, comentários e arquivos do sistema. Embora a densidade de duplicação tenha apresentado leve redução, de 5,8% para 5,2%, houve aumento absoluto nas linhas duplicadas, indicando que parte da melhoria percentual decorreu do crescimento geral do sistema.

A complexidade também apresentou piora significativa. A complexidade ciclomática praticamente dobrou, passando de 748 para 1.399, enquanto a complexidade cognitiva aumentou de 471 para 1.058. Esses resultados indicam um código estruturalmente mais complexo e potencialmente mais difícil de compreender e manter. Assim, apesar dos ganhos em confiabilidade e manutenibilidade, o aumento expressivo da complexidade comprometeu a qualidade global da refatoração produzida pelo Copilot.

A refatoração realizada com o Amazon Q Developer apresentou o desempenho menos favorável entre as abordagens analisadas. Em segurança, houve melhoria pontual com eliminação do único *issue* existente. Entretanto, não foram observados avanços em confiabilidade, mantendo-se os mesmos 65 *issues* identificados na versão original do sistema.

Na dimensão de manutenibilidade, ocorreu aumento no número de *issues*, passando de 540 para 657. Embora tenha sido observada redução no esforço estimado de correção, o crescimento no volume total de problemas indica piora estrutural da aplicação.

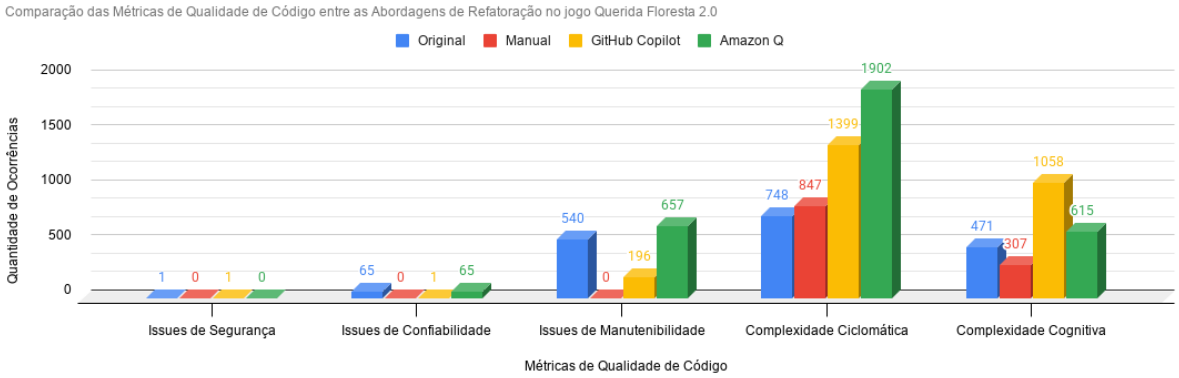
Os resultados relacionados à duplicação foram particularmente negativos. A densidade duplicada aumentou de 5,8% para 11,4%, acompanhada de crescimento expressivo em linhas, blocos e arquivos duplicados. Esse comportamento evidencia forte introdução de redundâncias e perda de padronização estrutural, aumentando a propensão a inconsistências futuras e dificultando a manutenção do sistema.

Além disso, o Amazon Q promoveu aumento significativo da complexidade do projeto. A complexidade ciclomática elevou-se de 748 para 1.902, enquanto a complexidade cognitiva aumentou de 471 para 615. Esses indicadores sugerem um sistema mais difícil de compreender, testar e evoluir. De forma geral, apesar de ganhos pontuais em segurança, a abordagem apresentou regressões relevantes nos demais critérios avaliados.

A Figura 4 apresenta a comparação das principais métricas de qualidade entre as abordagens analisadas. O gráfico evidencia visualmente a superioridade da refatoração manual,

especialmente na eliminação completa dos *issues* de segurança, confiabilidade e manutenibilidade. Também é possível observar o aumento expressivo da complexidade ciclomática e cognitiva nas abordagens automatizadas, sobretudo com o Amazon Q Developer, enquanto o GitHub Copilot apresentou desempenho intermediário, combinando melhorias funcionais com aumento considerável de complexidade estrutural.

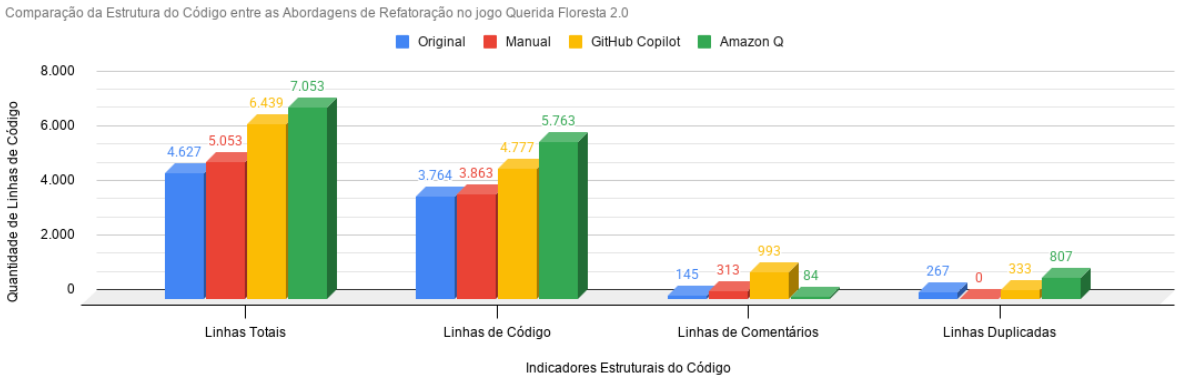
Figura 4 – Comparação das Métricas de Qualidade de Código entre as Abordagens de Refatoração no jogo Querida Floresta 2.0



Fonte: Autoria própria (2026).

Complementarmente, a Figura 5 apresenta a comparação das métricas relacionadas à estrutura do código, incluindo linhas de código, linhas totais, linhas de comentário e linhas duplicadas. Observa-se que a refatoração manual promoveu aumento moderado do sistema acompanhado por crescimento proporcional da documentação e eliminação total de duplicações. Já as abordagens automatizadas apresentaram expansão significativa da base de código, com aumento simultâneo de linhas totais e linhas duplicadas, reforçando os indícios de regressão estrutural observados anteriormente.

Figura 5 – Comparação da Estrutura do Código entre as Abordagens de Refatoração no jogo Querida Floresta 2.0



Fonte: Autoria própria (2026).

De forma comparativa, conclui-se que a refatoração manual foi amplamente superior no projeto Querida Floresta 2.0, sendo a única abordagem capaz de promover melhorias consis-

tentes em todos os critérios analisados. O GitHub Copilot apresentou resultados intermediários, com avanços relevantes em confiabilidade e manutenibilidade, porém comprometidos pelo aumento expressivo da complexidade e do tamanho do sistema. Já o Amazon Q Developer apresentou desempenho insatisfatório na maior parte dos indicadores, com regressões significativas em duplicação, complexidade e organização estrutural.

3.2.1.3 SpeakSnake

Os resultados obtidos no projeto SpeakSnake evidenciam diferenças claras entre as abordagens de refatoração analisadas. De forma geral, a refatoração manual apresentou desempenho significativamente superior, promovendo melhorias consistentes em todos os indicadores avaliados. As abordagens assistidas por Inteligência Artificial Generativa, por sua vez, demonstraram comportamentos mais instáveis, combinando avanços pontuais com regressões estruturais relevantes.

A refatoração manual promoveu melhorias expressivas nos critérios de segurança, confiabilidade, manutenibilidade, duplicação e complexidade. Todos os *issues* relacionados à segurança e confiabilidade foram eliminados, reduzindo também o esforço estimado de correção a zero. Na dimensão de manutenibilidade, o número de *issues* foi reduzido de 115 para apenas 2, indicando uma reorganização estrutural significativa e um código substancialmente mais preparado para manutenção e evolução futura.

Os resultados relacionados à duplicação foram igualmente relevantes. A densidade de duplicação caiu de 12,6% para 0,0%, eliminando completamente linhas, blocos e arquivos duplicados. Esse comportamento sugere uma refatoração orientada à reutilização adequada de componentes e centralização de responsabilidades, reduzindo redundâncias e aumentando a consistência do sistema.

Em relação à estrutura do projeto, observou-se leve crescimento em métricas como linhas de código, funções, classes e arquivos. Entretanto, esse aumento não representou perda de qualidade estrutural, estando provavelmente associado à modularização do sistema e à separação mais adequada de responsabilidades entre componentes.

No aspecto de complexidade, a refatoração manual apresentou melhorias consistentes. A complexidade ciclomática foi reduzida de 171 para 154, enquanto a complexidade cognitiva caiu de 167 para 89. Esses resultados indicam um código mais simples, compreensível e de manutenção menos onerosa.

A abordagem com GitHub Copilot apresentou resultados intermediários. Em segurança, houve eliminação do único *issue* existente, enquanto na confiabilidade observou-se redução parcial dos problemas, passando de 12 para 9 *issues*. Na dimensão de manutenibilidade, a ferramenta também apresentou melhora significativa, reduzindo os *issues* de 115 para 30.

Apesar desses avanços, os resultados estruturais mostraram limitações importantes. A duplicação aumentou de forma relevante, com crescimento da densidade de 12,6% para 14,3%, além do aumento no número absoluto de linhas duplicadas. Esse comportamento indica que a

ferramenta introduziu redundâncias durante o processo de refatoração, comprometendo parte dos ganhos obtidos em outros critérios.

As métricas de complexidade também apresentaram piora. A complexidade ciclomática aumentou de 171 para 188, enquanto a complexidade cognitiva cresceu de 167 para 187. Esses indicadores sugerem um código estruturalmente mais difícil de compreender e manter. Embora tenha havido redução pontual de *issues*, o aumento simultâneo da complexidade e duplicação comprometeu a qualidade global da refatoração realizada pelo Copilot.

A refatoração conduzida com o Amazon Q Developer apresentou os resultados menos favoráveis entre as abordagens analisadas. Em segurança, houve piora, com aumento dos *issues* de 1 para 3 e crescimento do esforço estimado de correção, indicando introdução de novas vulnerabilidades. Na confiabilidade, observou-se apenas melhora discreta, com redução de 12 para 9 *issues*.

Na dimensão de manutenibilidade, a situação também apresentou regressão. O número de *issues* aumentou de 115 para 119, mantendo-se praticamente o mesmo esforço estimado de correção. Esse comportamento sugere que as alterações realizadas não contribuíram para melhorar a organização estrutural do sistema e, em alguns casos, introduziram novos problemas.

Os resultados relacionados à duplicação foram particularmente negativos. A densidade duplicada aumentou de 12,6% para 16,6%, acompanhada por crescimento significativo em linhas, blocos e arquivos duplicados. Paralelamente, houve expansão considerável do tamanho geral do sistema, indicando aumento simultâneo de redundância e volume de código.

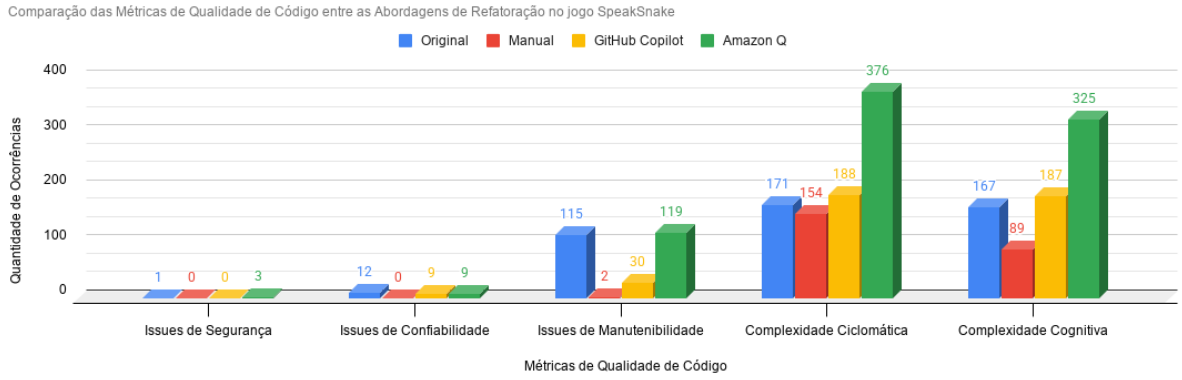
A complexidade também apresentou crescimento expressivo. A complexidade ciclomática mais que dobrou, passando de 171 para 376, enquanto a complexidade cognitiva aumentou de 167 para 325. Esses resultados indicam um sistema substancialmente mais difícil de compreender, testar e evoluir após a refatoração automatizada com Amazon Q Developer.

A Figura 6 apresenta a comparação das principais métricas de qualidade entre as abordagens aplicadas ao projeto SpeakSnake. O gráfico evidencia visualmente a superioridade da refatoração manual, especialmente na eliminação dos *issues* de segurança e confiabilidade, na redução dos problemas de manutenibilidade e nos menores índices de complexidade ciclomática e cognitiva. Também é possível observar que as abordagens automatizadas apresentaram aumento de complexidade, sobretudo no caso do Amazon Q Developer.

Complementarmente, a Figura 7 apresenta a comparação das métricas estruturais do sistema, incluindo linhas de código, linhas totais, linhas de comentário e linhas duplicadas. Observa-se que a abordagem manual promoveu reorganização estrutural acompanhada de eliminação total das duplicações, enquanto o GitHub Copilot apresentou aumento moderado de redundância. Já o Amazon Q Developer demonstrou crescimento significativo tanto no tamanho do sistema quanto na quantidade de linhas duplicadas, reforçando os indícios de piora estrutural observados nas demais métricas.

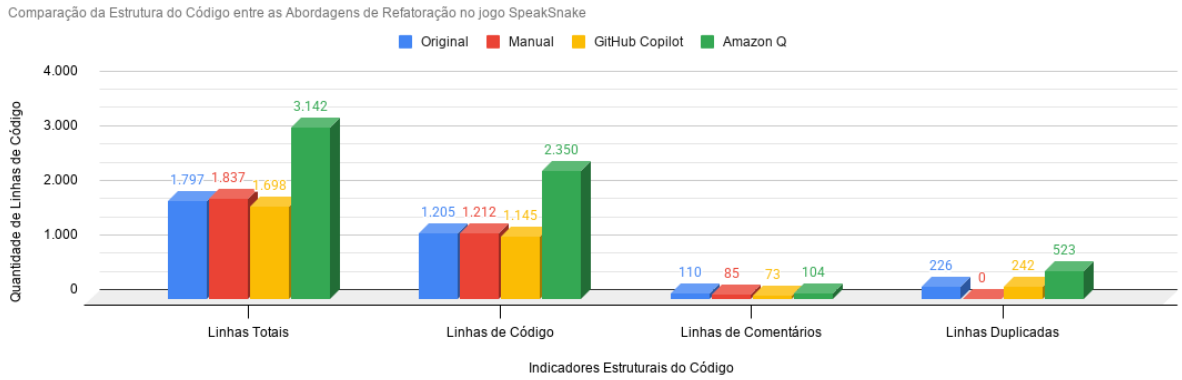
De forma comparativa, conclui-se que a refatoração manual foi novamente a abordagem mais eficaz, promovendo melhorias consistentes em todos os critérios avaliados. O GitHub Copilot apresentou desempenho intermediário, obtendo avanços em segurança e redução de

Figura 6 – Comparação das Métricas de Qualidade de Código entre as Abordagens de Refatoração no jogo SpeakSnake



Fonte: Autoria própria (2026).

Figura 7 – Comparação da Estrutura do Código entre as Abordagens de Refatoração no jogo SpeakSnake



Fonte: Autoria própria (2026).

issues, porém acompanhado de aumento de complexidade e duplicação. Já o Amazon Q Developer apresentou regressões significativas em múltiplos indicadores, especialmente em complexidade, redundância e organização estrutural do sistema.

3.2.2 Comparação entre Refatoração Manual e IA Generativa

Nesta seção, é apresentada uma comparação entre os resultados obtidos por meio da refatoração manual e aqueles alcançados com o uso de ferramentas de Inteligência Artificial Generativa. A análise busca identificar diferenças em termos de qualidade do código, eficiência no processo de refatoração e aderência às boas práticas de desenvolvimento. Dessa forma, pretende-se avaliar em que medida as abordagens automatizadas conseguem se aproximar dos resultados obtidos por intervenção humana.

3.2.2.1 Memoletrando

A comparação entre a refatoração manual e as abordagens baseadas em IA evidencia diferenças consistentes na qualidade dos resultados obtidos. Em relação ao GitHub Copilot, observa-se inicialmente equivalência no aspecto de segurança, com ausência de *issues* em ambas as abordagens. No entanto, essa similaridade não se sustenta nos demais critérios.

Na confiabilidade, a refatoração manual elimina completamente os problemas, enquanto o Copilot mantém 153 *issues* e um esforço de correção elevado, indicando que a abordagem automatizada não foi capaz de tratar adequadamente as falhas existentes. Esse comportamento se repete na manutenibilidade, em que a abordagem manual apresenta apenas 5 *issues*, contrastando com 808 na solução automatizada, além de uma diferença expressiva no esforço de correção (25 minutos contra mais de 6 dias). Esses resultados apontam para uma estrutura de código significativamente mais organizada e sustentável na refatoração manual.

No que se refere à duplicação, os valores percentuais são relativamente próximos (5,2% na manual e 5,8% com Copilot), porém a análise absoluta revela maior volume de redundâncias na abordagem automatizada. Esse cenário é influenciado pelo aumento expressivo no tamanho do sistema gerado pelo Copilot, o que sugere que a redução percentual não decorre necessariamente de uma refatoração mais eficiente, mas sim da expansão do código. Essa interpretação é reforçada pela análise de complexidade, na qual o Copilot apresenta valores superiores tanto na complexidade ciclomática quanto na cognitiva, indicando um sistema mais difícil de compreender e manter.

Na comparação com o Amazon Q Developer, observa-se um padrão semelhante. Embora ambas as abordagens apresentem equivalência em segurança, a refatoração manual novamente se destaca na confiabilidade, eliminando completamente os *issues*, enquanto a solução automatizada mantém 147 ocorrências. A diferença também é expressiva na manutenibilidade, com 5 *issues* na abordagem manual contra 716 na solução baseada em IA, além de maior esforço de correção.

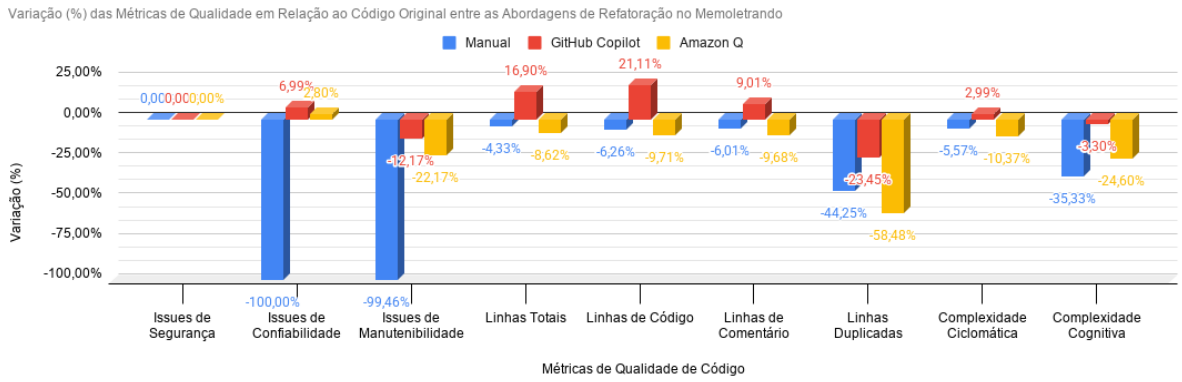
Por outro lado, o Amazon Q Developer apresenta desempenho superior em duplicação, com densidade de 4,0%, inferior aos 5,2% da refatoração manual, além de menores valores absolutos de redundância. Esse resultado indica maior eficiência na eliminação de código duplicado. Adicionalmente, a solução automatizada gera um sistema mais enxuto, com menor número de linhas de código e menor complexidade ciclomática.

Entretanto, esses ganhos pontuais não se refletem de forma consistente nos demais indicadores. A complexidade cognitiva permanece superior à da refatoração manual, sugerindo maior dificuldade de compreensão do código. Além disso, a manutenção de altos níveis de *issues* compromete a qualidade geral da solução.

A Figura 8 apresenta a variação percentual das métricas de qualidade do projeto Memoletrando em relação ao código original, considerando as abordagens de refatoração manual, GitHub Copilot e Amazon Q Developer. Observa-se que, no contexto do Memoletrando, a refatoração manual apresentou os resultados mais consistentes na redução de problemas estruturais,

principalmente em confiabilidade e manutenibilidade, enquanto as ferramentas de IA apresentaram melhorias mais relacionadas à reorganização e padronização do código.

Figura 8 – Variação (%) das Métricas de Qualidade em Relação ao Código Original entre as Abordagens de Refatoração no Memoletrando



Fonte: Autoria própria (2026).

De forma geral, observa-se que a refatoração manual apresenta desempenho superior em aspectos críticos como confiabilidade e manutenibilidade, garantindo maior estabilidade e qualidade estrutural do sistema. As abordagens baseadas em IA, embora apresentem vantagens específicas, como melhorias em duplicação no caso do Amazon Q Developer, não demonstram consistência suficiente para superar a abordagem manual.

3.2.2.2 Querida Floresta 2.0

A comparação entre a refatoração manual e as abordagens baseadas em IA evidencia diferenças marcantes na qualidade dos resultados. Em relação ao GitHub Copilot, observa-se inicialmente que, embora a ferramenta tenha reduzido significativamente os problemas em relação ao estado inicial, ela não alcança o nível de qualidade obtido pela abordagem manual. Enquanto a refatoração manual elimina completamente os *issues* de segurança, confiabilidade e manutenibilidade, o Copilot ainda mantém ocorrências residuais, com destaque para os 196 *issues* de manutenibilidade e esforço de correção associado.

Essa diferença torna-se ainda mais evidente ao analisar a duplicação e a complexidade. A refatoração manual elimina completamente qualquer redundância, enquanto o Copilot mantém uma densidade de duplicação de 5,2%, além de apresentar crescimento no tamanho do sistema, com aumento em linhas de código, declarações e comentários. Esse aumento estrutural influencia diretamente a complexidade, que se eleva de forma expressiva tanto na dimensão ciclomática quanto na cognitiva, indicando um código mais difícil de compreender e manter. Em contraste, a abordagem manual mantém níveis significativamente mais baixos nesses indicadores, reforçando sua eficiência na simplificação do sistema.

Na comparação com o Amazon Q Developer, observa-se um cenário ainda mais desfavorável para a abordagem automatizada. Embora exista equivalência no aspecto de segurança,

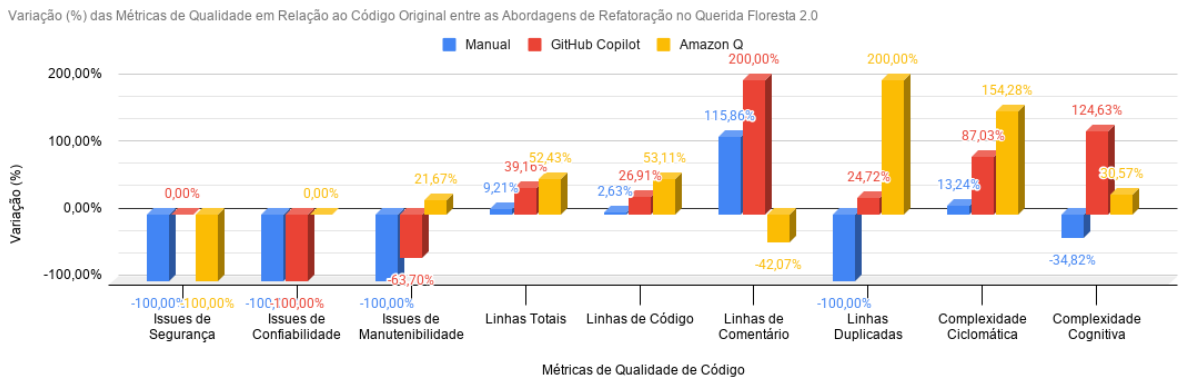
a confiabilidade não apresenta evolução, mantendo os mesmos 65 *issues* do estado inicial. Na manutenibilidade, há aumento no número de *issues*, totalizando 657, o que indica regressão na qualidade estrutural do código, mesmo com redução no esforço individual de correção.

Os indicadores de duplicação e tamanho reforçam essa tendência. Diferentemente da refatoração manual, que elimina completamente redundâncias, o Amazon Q Developer aumenta significativamente a densidade de duplicação para 11,4%, acompanhado por crescimento expressivo em linhas, blocos e arquivos duplicados. Esse comportamento sugere introdução de código redundante e menor controle estrutural. Paralelamente, o sistema gerado apresenta aumento considerável em praticamente todas as métricas de tamanho, indicando expansão não necessariamente associada a melhorias qualitativas.

A complexidade também se eleva de forma substancial, especialmente na dimensão ciclomática, que atinge valores muito superiores aos observados na refatoração manual, além de aumento na complexidade cognitiva. Esses resultados indicam um sistema mais difícil de compreender, evoluir e manter.

A Figura 9 apresenta a variação percentual das métricas de qualidade do projeto Querida Floresta 2.0 em relação ao código original. Os resultados mostram diferenças significativas entre as abordagens analisadas. No contexto do Querida Floresta 2.0, percebe-se que a refatoração manual manteve maior equilíbrio entre redução de problemas e controle da complexidade estrutural. As ferramentas de IA, embora tenham promovido reorganizações importantes, introduziram aumentos expressivos em métricas estruturais, evidenciando limitações no controle global da arquitetura do sistema.

Figura 9 – Variação (%) das Métricas de Qualidade em Relação ao Código Original entre as Abordagens de Refatoração no Querida Floresta 2.0



Fonte: Autoria própria (2026).

De forma geral, observa-se que a refatoração manual apresenta desempenho superior, sendo a única abordagem capaz de promover melhorias consistentes em todos os indicadores analisados, incluindo eliminação completa de *issues* e duplicações, além de controle efetivo da complexidade. O GitHub Copilot apresenta desempenho intermediário, com melhorias parciais, porém limitadas pelo aumento de complexidade e expansão do sistema. Já o Amazon

Q Developer demonstra o desempenho mais limitado, com regressões em múltiplos aspectos estruturais.

3.2.2.3 SpeakSnake

A comparação entre a refatoração manual e as abordagens baseadas em IA evidencia diferenças significativas na qualidade do código resultante. Em relação ao GitHub Copilot, observa-se equivalência apenas no critério de segurança, já que ambas as abordagens não apresentaram *issues*. No entanto, essa similaridade não se mantém nos demais indicadores.

Na dimensão de confiabilidade, a abordagem manual elimina completamente os problemas identificados, enquanto o Copilot mantém 9 *issues* e esforço de correção associado. Considerando que o código original possuía 12 ocorrências, a solução automatizada promove apenas uma melhoria parcial. Situação semelhante ocorre na manutenibilidade: a refatoração manual reduz os *issues* para 2, ao passo que o Copilot registra 30, além de maior esforço de manutenção, indicando limitações na organização estrutural do código gerado.

As diferenças tornam-se mais evidentes na análise de duplicação. A abordagem manual remove integralmente as redundâncias, enquanto o Copilot aumenta a densidade de duplicação para 14,3%, mantendo linhas e blocos repetidos. Isso sugere que a refatoração automatizada introduziu trechos redundantes no sistema. Embora o código produzido pelo Copilot apresente redução em algumas métricas de tamanho, tal diminuição não resulta em ganhos qualitativos relevantes.

A análise de complexidade reforça esse comportamento. A refatoração manual reduz tanto a complexidade ciclomática quanto a cognitiva, enquanto o Copilot eleva ambos os indicadores, produzindo um código mais difícil de compreender e manter. Dessa forma, apesar das melhorias pontuais, a solução gerada com GitHub Copilot não alcança o mesmo nível de qualidade estrutural obtido manualmente.

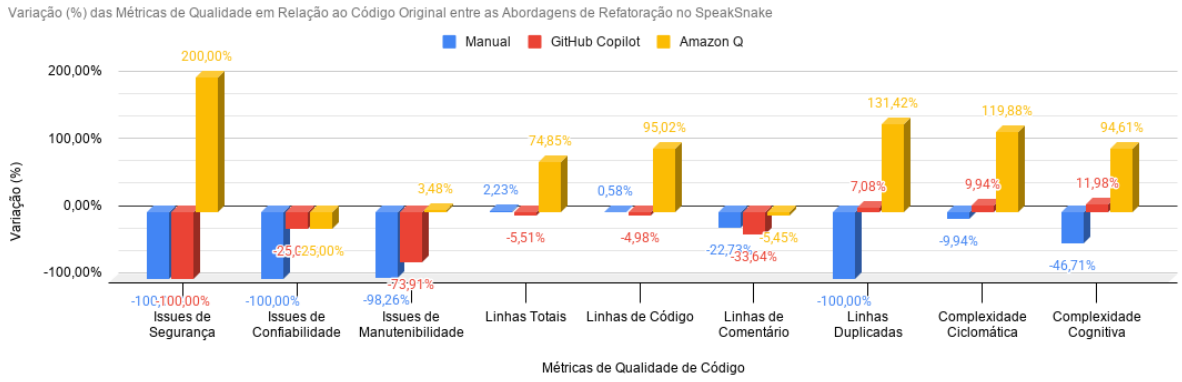
Na comparação com o Amazon Q Developer, as diferenças são ainda mais expressivas. Enquanto a refatoração manual elimina completamente os *issues* de segurança, a abordagem automatizada introduz novos problemas, elevando o total para 3. Na confiabilidade, o Amazon Q mantém 9 *issues*, contrastando novamente com a eliminação total alcançada pela abordagem manual.

Na manutenibilidade, a discrepância é ainda maior: a refatoração manual reduz os *issues* para 2, enquanto o Amazon Q apresenta 119 ocorrências, indicando elevado volume de problemas estruturais e maior custo de manutenção. A duplicação também se agrava na solução automatizada, com aumento da densidade para 16,6% e crescimento significativo de linhas e blocos duplicados, evidenciando forte introdução de redundâncias.

Além disso, o sistema gerado pelo Amazon Q apresenta aumento substancial em praticamente todas as métricas de tamanho. Esse crescimento é acompanhado por forte elevação da complexidade ciclomática e cognitiva, ambas mais que dobrando em relação ao código original, o que torna o sistema significativamente mais difícil de compreender e evoluir.

A Figura 10 apresenta a variação percentual das métricas de qualidade do projeto SpeakSnake em relação ao código original. Os resultados reforçam o padrão observado nos demais projetos analisados, com a refatoração manual demonstrando maior controle sobre a redução de complexidade e de problemas estruturais, enquanto as abordagens com IA Generativa priorizaram modularização e reorganização arquitetural, frequentemente acompanhadas pelo aumento do volume e da complexidade do código.

Figura 10 – Variação (%) das Métricas de Qualidade em Relação ao Código Original entre as Abordagens de Refatoração no SpeakSnake



Fonte: Autoria própria (2026).

De modo geral, a refatoração manual apresentou o melhor desempenho, sendo a única abordagem capaz de promover melhorias consistentes em todos os indicadores analisados, incluindo eliminação de *issues*, remoção de duplicações e redução da complexidade. O GitHub Copilot obteve resultados intermediários, com avanços parciais comprometidos pelo aumento de redundância e complexidade. Já o Amazon Q Developer apresentou o desempenho mais limitado, com regressões em múltiplos aspectos de qualidade.

3.2.3 Comparação entre Ferramentas de IA Generativa

Esta seção tem como objetivo comparar o desempenho das ferramentas de Inteligência Artificial Generativa utilizadas no estudo, considerando os resultados obtidos na refatoração dos diferentes projetos. A análise foca na qualidade das sugestões geradas, na capacidade de adaptação ao contexto do código e na efetividade das modificações propostas, permitindo identificar pontos fortes e limitações de cada ferramenta.

3.2.3.1 Memoletrando

A comparação entre as ferramentas de IA generativa, GitHub Copilot e Amazon Q Developer, evidencia diferenças consistentes no desempenho das abordagens. No aspecto de segurança, ambas apresentam comportamento equivalente, sem ocorrência de *issues* e com esforço de correção nulo, não havendo distinção relevante nesse critério.

Entretanto, ao analisar a confiabilidade, observa-se uma leve vantagem do Amazon Q Developer, que apresenta 147 *issues*, em comparação aos 153 identificados no GitHub Copilot. Embora a diferença seja pequena e não impacte o esforço de correção, ela indica uma tendência de melhor desempenho do Amazon Q na redução de falhas funcionais.

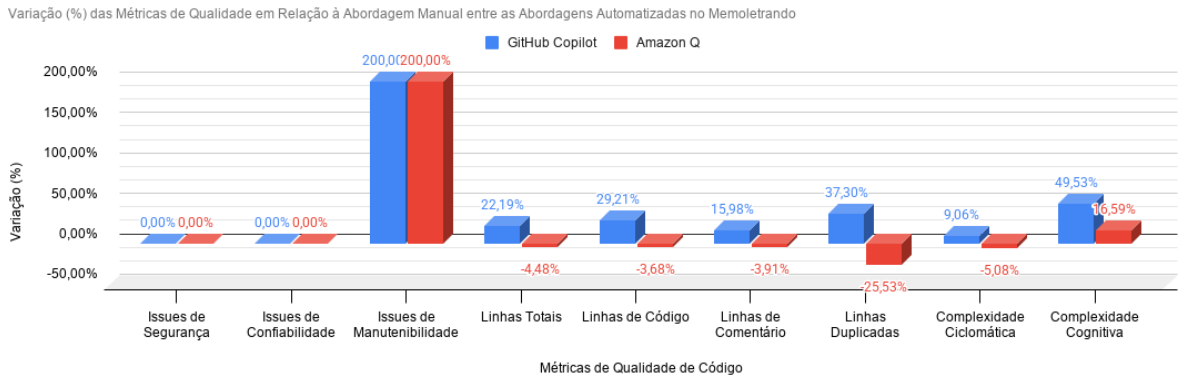
Na manutenibilidade, essa diferença torna-se mais evidente. O Amazon Q Developer apresenta menor número de *issues* (716 contra 808), além de menor esforço estimado de correção, sugerindo uma estrutura de código relativamente mais organizada. Ainda assim, ambas as ferramentas mantêm níveis elevados de problemas, indicando limitações na capacidade de promover melhorias estruturais mais profundas.

A análise de duplicação reforça a vantagem do Amazon Q Developer. A ferramenta apresenta densidade de duplicação de 4,0%, inferior aos 5,8% do GitHub Copilot, além de menores valores absolutos de redundância. Esse resultado sugere maior eficiência na consolidação de trechos repetidos e melhor reaproveitamento de código.

Diferenças semelhantes são observadas no tamanho e na complexidade do sistema. O código gerado pelo Amazon Q Developer é mais enxuto, com menor volume de linhas, declarações e funções, além de apresentar níveis inferiores de complexidade ciclomática e cognitiva. Em contraste, o GitHub Copilot gera um sistema mais extenso e com maior complexidade, o que pode impactar negativamente a compreensão e a manutenção do código.

A Figura 11 apresenta a variação percentual das métricas de qualidade do projeto Memoletrando em relação à abordagem manual, comparando os resultados obtidos com o GitHub Copilot e o Amazon Q Developer. Observa-se que ambas as ferramentas apresentaram comportamento semelhante em relação aos *issues* em comparação à abordagem manual. Entretanto, os resultados sugerem que o GitHub Copilot priorizou a incrementação de melhorias, enquanto o Amazon Q Developer concentrou-se em simplificação arquitetural e redução de redundâncias.

Figura 11 – Variação (%) das Métricas de Qualidade em Relação à Abordagem Manual entre as Abordagens Automatizadas no Memoletrando



Fonte: Autoria própria (2026).

Em síntese, observa-se que o Amazon Q Developer apresenta desempenho superior ao GitHub Copilot na maioria dos indicadores analisados, especialmente em duplicação, tamanho e complexidade. No entanto, os altos níveis de *issues* em ambas as abordagens indicam que,

apesar das diferenças entre as ferramentas, nenhuma delas foi capaz de promover melhorias estruturais abrangentes.

3.2.3.2 Querida Floresta 2.0

A análise comparativa entre GitHub Copilot e Amazon Q Developer no projeto Querida Floresta 2.0 evidencia um comportamento heterogêneo entre as ferramentas, sem predominância absoluta em todos os critérios. No aspecto de segurança, o Amazon Q Developer apresenta melhor desempenho ao eliminar o único *issue* existente, enquanto o GitHub Copilot mantém uma ocorrência residual.

Em contrapartida, na confiabilidade, o cenário se inverte de forma significativa. O Copilot reduz os *issues* para apenas 1, enquanto o Amazon Q mantém 65, com esforço de correção elevado, indicando limitações na resolução de falhas funcionais. Esse padrão também se reflete na manutenibilidade, em que o Copilot apresenta 196 *issues*, valor substancialmente inferior aos 657 observados no Amazon Q Developer, sugerindo uma estrutura de código mais organizada e com menor acúmulo de problemas técnicos.

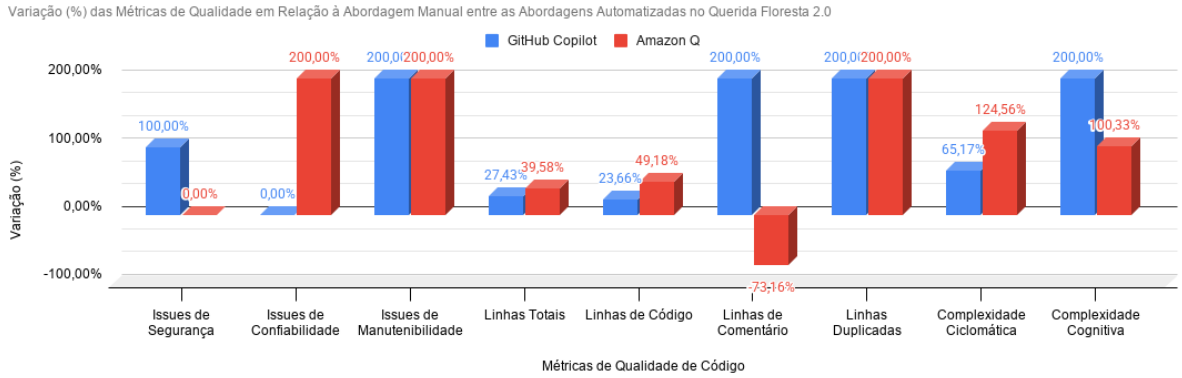
A análise de duplicação reforça essa tendência. O GitHub Copilot apresenta densidade de 5,2%, significativamente inferior aos 11,4% do Amazon Q Developer, além de menores valores absolutos de redundância. Esse resultado indica maior eficiência na consolidação de código e menor propensão à inconsistência. Por outro lado, o Amazon Q demonstra introdução relevante de trechos duplicados, o que pode impactar negativamente a manutenção do sistema.

Em relação ao tamanho e à complexidade, observa-se um cenário mais equilibrado. O Amazon Q Developer gera um sistema mais extenso, com maior número de linhas, funções e classes, além de maior complexidade ciclomática. No entanto, apresenta menor complexidade cognitiva em comparação ao Copilot, o que pode indicar melhor organização lógica em partes do código. Já o GitHub Copilot, embora produza um sistema relativamente mais compacto e com maior volume de comentários, apresenta maior complexidade cognitiva, sugerindo maior dificuldade de compreensão global.

A Figura 12 apresenta a comparação das variações percentuais das métricas de qualidade no projeto Querida Floresta 2.0 entre as ferramentas GitHub Copilot e Amazon Q Developer. O GitHub Copilot apresentou aumento expressivo em linhas de comentário, linhas duplicadas e complexidade cognitiva. O Amazon Q Developer, por sua vez, apresentou aumento em linhas duplicadas e complexidade ciclomática, com destaque para a redução acentuada nas linhas de comentário.

Em síntese, observa-se que o GitHub Copilot apresenta desempenho superior em aspectos estruturais críticos, como confiabilidade, manutenibilidade e duplicação, enquanto o Amazon Q Developer se destaca pontualmente em segurança e complexidade cognitiva. Ainda assim, os resultados indicam que ambas as ferramentas apresentam limitações relevantes, não sendo capazes de promover melhorias consistentes em todos os critérios avaliados.

Figura 12 – Variação (%) das Métricas de Qualidade em Relação à Abordagem Manual entre as Abordagens Automatizadas no Querida Floresta 2.0



Fonte: Autoria própria (2026).

3.2.3.3 SpeakSnake

A comparação entre GitHub Copilot e Amazon Q Developer no projeto SpeakSnake evidencia diferenças consistentes entre as ferramentas, com vantagem mais clara para o Copilot na maioria dos critérios analisados. No aspecto de segurança, o GitHub Copilot apresenta melhor desempenho, com ausência de *issues* e esforço de correção nulo, enquanto o Amazon Q Developer introduz novos problemas, totalizando 3 ocorrências. Em relação à confiabilidade, ambas as ferramentas apresentam resultados equivalentes, com 9 *issues* e esforço de correção semelhante, não havendo distinção relevante nesse critério.

Na manutenibilidade, o GitHub Copilot demonstra vantagem significativa, com 30 *issues* frente a 119 do Amazon Q Developer, além de menor esforço de correção. Esse resultado indica uma estrutura de código relativamente mais organizada e com menor acúmulo de problemas técnicos, enquanto a solução do Amazon Q apresenta maior volume de inconsistências, o que pode impactar negativamente a manutenção a longo prazo.

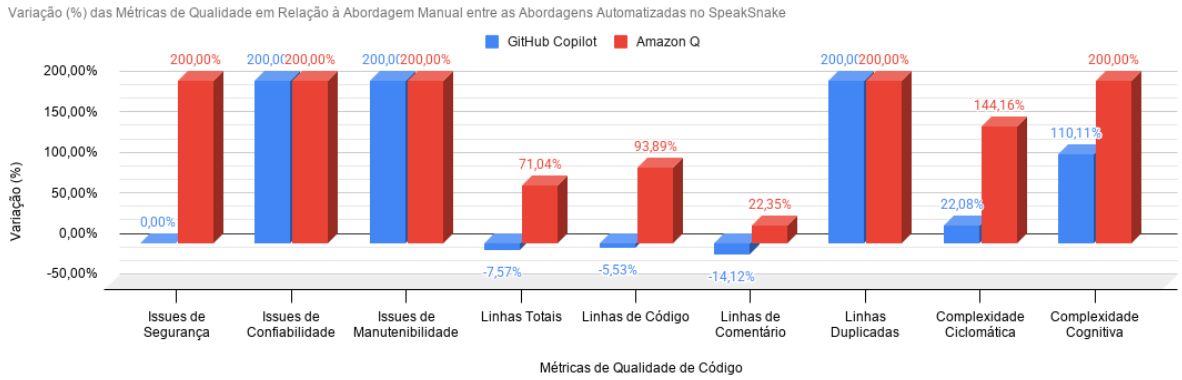
Em relação à duplicação, ambas as abordagens apresentam níveis elevados, porém o Copilot mantém melhor desempenho, com densidade de 14,3% contra 16,6% do Amazon Q Developer, além de menores valores absolutos de redundância. Esse cenário indica que, embora nenhuma das ferramentas tenha sido eficaz na eliminação de código duplicado, o Copilot introduziu menos redundâncias.

A análise de tamanho e complexidade reforça essa diferença. O Amazon Q Developer gera um sistema significativamente maior e mais complexo, com aumento expressivo em métricas estruturais e elevação acentuada da complexidade ciclomática e cognitiva. Em contraste, o GitHub Copilot produz um sistema mais enxuto e com níveis inferiores de complexidade, ainda que não ideais, favorecendo a compreensão e a manutenção.

A Figura 13 apresenta a comparação entre as ferramentas GitHub Copilot e Amazon Q Developer no projeto SpeakSnake, evidenciando um comportamento bastante distinto entre as ferramentas analisadas. O GitHub Copilot apresentou redução nas linhas totais, linhas de

código e linhas de comentário, indicando uma abordagem voltada à simplificação incremental do código e à remoção de redundâncias. Já o Amazon Q Developer apresentou crescimento expressivo em praticamente todas as métricas estruturais analisadas.

Figura 13 – Variação (%) das Métricas de Qualidade em Relação à Abordagem Manual entre as Abordagens Automatizadas no SpeakSnake



Fonte: Autoria própria (2026).

Em síntese, observa-se que o GitHub Copilot apresenta desempenho superior ao Amazon Q Developer na maioria dos critérios avaliados, especialmente em segurança, manutenibilidade e controle de complexidade. Por outro lado, a ausência de melhorias substanciais em duplicação e os níveis ainda elevados de *issues* em ambas as abordagens indicam limitações relevantes no uso de IA para refatoração nesse contexto.

3.2.4 Comparação entre Projetos por Ferramenta de IA Generativa

Nesta seção, é analisado o desempenho das ferramentas de Inteligência Artificial Generativa em diferentes projetos, considerando as particularidades de cada contexto. O objetivo é verificar como fatores como linguagem de programação, estrutura do código e complexidade do sistema influenciam os resultados obtidos. Essa análise permite compreender a consistência e a aplicabilidade das ferramentas em cenários distintos.

3.2.4.1 GitHub Copilot

A análise do desempenho do GitHub Copilot nos diferentes projetos evidencia variações relevantes associadas ao porte e à complexidade dos sistemas. No projeto Memoletrando, a ferramenta apresenta desempenho limitado em diversos indicadores de qualidade. Apesar da ausência de *issues* de segurança, os elevados valores de confiabilidade (153 *issues*) e manutenibilidade (808 *issues*), acompanhados de alto esforço de correção, indicam um acúmulo significativo de problemas estruturais.

Esse cenário é reforçado pelos indicadores de tamanho e complexidade. O sistema gerado apresenta grande volume de código, com altos valores de linhas, arquivos e funções, além

de níveis elevados de complexidade ciclomática e cognitiva. A duplicação, embora moderada em termos percentuais (5,8%), apresenta valores absolutos expressivos, refletindo a escala do sistema. Em conjunto, esses resultados sugerem que, em projetos maiores, o Copilot tende a produzir soluções mais extensas, porém com maior dificuldade de controle estrutural.

No projeto Querida Floresta 2.0, observa-se uma melhora significativa no desempenho. Os *issues* de segurança e confiabilidade são praticamente inexistentes, enquanto a manutenibilidade apresenta valores mais moderados (196 *issues*), indicando menor acúmulo de problemas. A duplicação também se mantém em níveis controlados (5,2%), com baixos valores absolutos, refletindo um código mais enxuto.

Em termos de tamanho e complexidade, os valores são intermediários, inferiores aos observados em Memoletrando, mas ainda com complexidade relevante, especialmente na dimensão ciclomática. Esse comportamento indica que, embora o Copilot se adapte melhor a projetos de menor escala, ainda enfrenta limitações na produção de código estruturalmente simples e de fácil compreensão.

No projeto SpeakSnake, o GitHub Copilot apresenta seu melhor desempenho relativo. A ausência de *issues* de segurança, aliada aos baixos valores de confiabilidade (9 *issues*) e manutenibilidade (30 *issues*), indica um código mais estável e com menor débito técnico. O sistema gerado também é o menor entre os projetos analisados, o que contribui para valores absolutos reduzidos em diversas métricas.

Entretanto, mesmo nesse cenário mais favorável, a duplicação permanece elevada (14,3%), indicando limitações na eliminação de redundâncias. A complexidade, embora inferior em termos absolutos, ainda se mostra relevante quando considerada em relação ao porte do projeto, especialmente na dimensão cognitiva.

De forma geral, observa-se que o desempenho do GitHub Copilot pode estar diretamente relacionado ao porte do projeto: quanto menor o sistema, melhores tendem a ser os resultados em termos de *issues* e complexidade. Por outro lado, a ferramenta apresenta dificuldades em manter consistência estrutural em sistemas maiores, além de limitações recorrentes no controle de duplicação. Esses resultados indicam que, embora o Copilot possa ser útil em cenários de menor escala, seu uso em projetos mais complexos requer maior supervisão para garantir a qualidade do código gerado.

3.2.4.2 Amazon Q Developer

A análise do desempenho do Amazon Q Developer nos diferentes projetos revela um comportamento sensível ao porte e à complexidade dos sistemas, com resultados heterogêneos entre os cenários avaliados. No projeto Memoletrando, a ferramenta apresenta desempenho intermediário. Embora não haja *issues* de segurança, os níveis de confiabilidade (147 *issues*) e manutenibilidade (716 *issues*) permanecem elevados, acompanhados de esforço significativo de correção, indicando limitações na redução de problemas estruturais.

Por outro lado, a duplicação destaca-se como um ponto positivo nesse contexto. O Memoletrando apresenta a menor densidade de duplicação entre os projetos analisados com o Amazon Q (4,0%), sugerindo maior eficiência na eliminação de redundâncias. Ainda assim, o sistema gerado é extenso, com altos valores de linhas, funções e arquivos, além de níveis elevados de complexidade ciclomática e cognitiva. Esse conjunto de resultados indica que, em sistemas maiores, a ferramenta tende a controlar melhor a redundância, mas não necessariamente a complexidade ou o acúmulo de *issues*.

No projeto Querida Floresta 2.0, o desempenho torna-se menos favorável. Apesar da ausência de *issues* de segurança, os indicadores de confiabilidade (65 *issues*) e manutenibilidade (657 *issues*) permanecem elevados para o porte do sistema, sugerindo menor eficiência na adaptação da ferramenta a contextos menores. Esse comportamento é reforçado pelos níveis de duplicação, que aumentam para 11,4%, indicando maior introdução de redundâncias.

Além disso, observa-se crescimento do sistema em termos de tamanho e complexidade, com elevação da complexidade ciclomática. A complexidade cognitiva, embora moderada, não compensa o aumento geral da estrutura, indicando que a organização do código não evolui de forma proporcional ao seu tamanho.

No projeto SpeakSnake, o desempenho do Amazon Q Developer é ainda mais limitado. Observa-se aumento de *issues* de segurança (3), além de valores de confiabilidade (9 *issues*) e manutenibilidade (119 *issues*) elevados em relação ao porte reduzido do sistema. Esse resultado sugere que, em projetos menores, a ferramenta tende não apenas a manter problemas existentes, mas também a introduzir novas inconsistências.

Esse comportamento é acompanhado por aumento significativo da duplicação, que atinge 16,6%, além de crescimento expressivo no tamanho do sistema. A complexidade também se eleva de forma acentuada, tanto na dimensão ciclomática quanto na cognitiva, resultando em um código mais difícil de compreender e manter.

De forma geral, observa-se que o Amazon Q Developer apresentou melhor desempenho relativo em sistemas maiores, especialmente no controle de duplicação. Entretanto, sua eficácia diminui à medida que o porte do projeto reduz, sendo acompanhada por aumento de redundâncias, complexidade e *issues*. Esses resultados indicam que a ferramenta carece de consistência entre diferentes contextos, exigindo maior supervisão para garantir qualidade, sobretudo em projetos menores, onde seu uso pode resultar em regressões estruturais relevantes.

3.3 Discussão dos Resultados

A análise comparativa dos resultados evidenciou diferenças relevantes entre a refatoração manual e as abordagens assistidas por Inteligência Artificial Generativa. De modo geral, a refatoração manual apresentou resultados mais consistentes nos atributos de qualidade avaliados. As ferramentas de IA, por sua vez, demonstraram comportamento mais variável, influenciado pelas características do projeto analisado, pela tecnologia utilizada e pela estratégia adotada por cada ferramenta para interpretar e modificar o código.

Os resultados sugerem que a abordagem manual se beneficiou da capacidade humana de compreender simultaneamente aspectos arquiteturais, estruturais e contextuais da aplicação. Essa característica favoreceu decisões mais precisas relacionadas à modularização, reorganização de responsabilidades, eliminação de duplicações e simplificação da estrutura do código. Como consequência, observou-se redução mais consistente de *code smells*, duplicações e complexidade cognitiva nos três projetos analisados.

Outro aspecto observado foi a previsibilidade das alterações realizadas manualmente. Mesmo quando houve aumento do volume de código em determinados módulos, esse crescimento esteve associado principalmente à aplicação de técnicas de refatoração voltadas à melhoria da organização estrutural, como extração de métodos, encapsulamento de responsabilidades e decomposição de componentes excessivamente complexos.

As abordagens baseadas em Inteligência Artificial Generativa também produziram melhorias relevantes, porém com resultados menos homogêneos. Em diversos casos, as ferramentas conseguiram reduzir duplicações, reorganizar componentes e simplificar trechos específicos do código. Entretanto, também foram observadas situações em que as modificações resultaram em aumento do tamanho da base de código, introdução de redundâncias ou crescimento da complexidade estrutural.

Parte desse comportamento pode estar relacionada à estratégia metodológica adotada neste estudo, baseada na utilização de um único *prompt* genérico para todas as interações. Embora essa decisão tenha sido importante para garantir a comparabilidade entre as ferramentas, ela também pode ter incentivado respostas mais abrangentes e menos direcionadas às técnicas clássicas de refatoração, levando as IAs a propor alterações adicionais relacionadas à robustez, validação de dados e reorganização funcional da aplicação.

A comparação entre as ferramentas revelou diferenças importantes de comportamento. O GitHub Copilot apresentou uma abordagem predominantemente incremental, realizando alterações graduais e geralmente mais alinhadas à estrutura existente dos projetos. Essa característica contribuiu para resultados mais previsíveis e para menor necessidade de adaptações estruturais posteriores. Em termos gerais, o Copilot apresentou melhor desempenho nos atributos de manutenibilidade e compreensibilidade, promovendo melhorias consistentes na organização do código e na redução localizada da complexidade.

O Amazon Q Developer, por outro lado, demonstrou uma abordagem mais abrangente e orientada à reorganização estrutural. Em diversos momentos, a ferramenta propôs alterações arquiteturais mais extensas, incluindo centralização de responsabilidades, reorganização de componentes e redistribuição de funcionalidades entre módulos. Essa característica resultou em ganhos pontuais de modularização e reusabilidade, especialmente em projetos de maior porte, mas também aumentou a necessidade de validação e correção das alterações realizadas.

As características dos projetos analisados também influenciaram os resultados obtidos. O Memoletrando, por apresentar maior volume de código e arquitetura multicamadas envolvendo Angular, TypeScript, Laravel e Ionic, representou o cenário mais desafiador para as fer-

ramentas de IA. Nesse contexto, observou-se maior incidência de alterações extensas e maior necessidade de validação das modificações geradas.

No projeto Querida Floresta 2.0, desenvolvido com Unity e C#, as ferramentas demonstraram desempenho intermediário. A forte interdependência entre *scripts* e o compartilhamento de responsabilidades entre componentes dificultaram a realização de alterações estruturais amplas sem impactos em outras partes da aplicação. Ainda assim, foram observados ganhos relacionados à modularização e reorganização do código.

Já o SpeakSnake apresentou os resultados mais favoráveis para as ferramentas de IA. Sua arquitetura mais simples e menor volume de código facilitaram a identificação de duplicações e oportunidades de reorganização estrutural. Apesar disso, ainda foram necessárias revisões para correção de inconsistências introduzidas automaticamente.

Outro resultado relevante refere-se à necessidade de intervenção humana. Embora as ferramentas tenham automatizado parte significativa das modificações, todas geraram alterações que demandaram validação contínua, correção de erros de compilação, ajustes de integração e revisão funcional. Esse comportamento reforça que, no contexto analisado, a utilização de IA Generativa ainda depende de supervisão humana para garantir a qualidade e a consistência das modificações realizadas.

Apesar dessas limitações, as ferramentas demonstraram vantagens importantes relacionadas à produtividade. Em todos os projetos analisados, observou-se redução do esforço necessário para identificação inicial de oportunidades de melhoria e geração rápida de alternativas de implementação. As ferramentas mostraram-se particularmente úteis em tarefas repetitivas, reorganizações localizadas e identificação de padrões recorrentes de código.

De forma geral, os resultados indicam que a refatoração manual apresentou os melhores resultados globais nos atributos de qualidade avaliados, especialmente em manutenibilidade, compreensibilidade e controle da complexidade estrutural. Entre as ferramentas analisadas, o GitHub Copilot apresentou desempenho mais consistente nesses atributos, enquanto o Amazon Q Developer se destacou nas tarefas de reorganização estrutural mais ampla e modularização de componentes. O Quadro 8 sintetiza os principais resultados observados ao longo das análises comparativas.

Assim, os resultados sugerem que a Inteligência Artificial Generativa pode atuar como um importante mecanismo de apoio ao processo de refatoração, especialmente para acelerar atividades operacionais e auxiliar na identificação de oportunidades de melhoria. Entretanto, as decisões relacionadas à arquitetura, à organização estrutural e à aplicação criteriosa das técnicas de refatoração continuam dependendo da análise humana. Nesse contexto, os resultados apontam para a adoção de uma abordagem híbrida, combinando a capacidade analítica do desenvolvedor com a produtividade proporcionada pelas ferramentas de IA Generativa.

Quadro 8 – Resumo Comparativo das Abordagens de Refatoração

Critério	Abordagem	Principais Resultados Observados
Manutenibilidade	Refatoração Manual	Apresentou melhorias mais consistentes e previsíveis na organização estrutural do código, com maior controle sobre modularização, coesão e redução de *code smells*.
Manutenibilidade	IA Generativa	Promoveu melhorias relevantes de forma automatizada, especialmente na reorganização estrutural. O GitHub Copilot apresentou resultados mais consistentes nesse atributo.
Compreensibilidade	Refatoração Manual	Produziu código mais organizado e legível por meio da decomposição de métodos extensos, simplificação de fluxos lógicos e redução da complexidade cognitiva.
Compreensibilidade	IA Generativa	Contribuiu para a reorganização do código e padronização de nomenclaturas. O GitHub Copilot destacou-se por realizar alterações mais alinhadas à estrutura original dos projetos.
Reusabilidade	Refatoração Manual	Favoreceu a extração de métodos, redução de duplicações e reorganização de responsabilidades, ampliando o potencial de reutilização de componentes e funções.
Reusabilidade	IA Generativa	Promoveu a criação e reorganização de componentes reutilizáveis, especialmente em propostas de modularização mais amplas.
Complexidade Estrutural	Refatoração Manual	Redução gradual e controlada da complexidade ciclomática e cognitiva, com simplificação de condicionais e melhor distribuição de responsabilidades entre componentes.
Complexidade Estrutural	IA Generativa	Obteve resultados variáveis. Em alguns casos reduziu duplicações e simplificou estruturas, porém também gerou aumento de complexidade.
Produtividade	Refatoração Manual	Demandou maior tempo de execução e esforço analítico, especialmente em projetos maiores e com arquitetura mais complexa.
Produtividade	IA Generativa	Reduziu significativamente o tempo necessário para identificação de oportunidades de melhoria e geração de soluções, automatizando tarefas repetitivas e operacionais.
Intervenção Humana	Refatoração Manual	Todo o processo decisório foi conduzido pelo desenvolvedor, proporcionando maior previsibilidade e controle sobre as modificações.
Intervenção Humana	IA Generativa	Apesar do elevado nível de automação, as ferramentas exigiram supervisão para validação funcional, correção de erros de compilação e resolução de regressões.

Fonte: Autoria própria (2026).

4 CONCLUSÃO

Este trabalho apresentou uma análise comparativa entre a refatoração manual e a refatoração assistida por ferramentas de Inteligência Artificial Generativa, avaliando seus impactos nos atributos de qualidade de software relacionados à manutenibilidade, compreensibilidade, reusabilidade e complexidade estrutural. A investigação foi conduzida em três projetos desenvolvidos com diferentes tecnologias, sendo elas Angular/TypeScript integrado ao Laravel e Ionic, Unity/C# e JavaScript/HTML/CSS, permitindo analisar o comportamento das abordagens em contextos arquiteturais distintos.

Os resultados obtidos indicam que a refatoração manual apresentou desempenho mais consistente nos atributos avaliados. A atuação do desenvolvedor responsável pela execução das refatorações, que possuía experiência prévia nas tecnologias utilizadas e conhecimento das técnicas de refatoração adotadas neste estudo, possibilitou a realização de modificações mais alinhadas à arquitetura e ao contexto de cada sistema. Como consequência, foram observadas reduções mais consistentes de *code smells*, duplicações e complexidade estrutural, além de melhorias na organização, modularização e compreensibilidade do código.

As ferramentas de Inteligência Artificial Generativa também contribuíram para a melhoria da qualidade dos sistemas analisados, especialmente na identificação de oportunidades de reorganização estrutural, redução de duplicações e automatização de tarefas repetitivas. Entretanto, os resultados apresentaram maior variabilidade entre projetos e ferramentas. Em alguns cenários, as alterações propostas resultaram em aumento do volume de código, introdução de novas abstrações e necessidade de correções adicionais para garantir a compatibilidade das modificações com o comportamento esperado dos sistemas.

Entre as ferramentas avaliadas, o GitHub Copilot apresentou resultados mais consistentes nos atributos de manutenibilidade e compreensibilidade, demonstrando uma abordagem incremental e mais aderente à estrutura original dos projetos. O Amazon Q Developer destacou-se principalmente em atividades relacionadas à reorganização estrutural e à modularização de componentes, contribuindo para ganhos de reusabilidade em alguns cenários. Contudo, também apresentou maior variabilidade nos resultados e maior necessidade de validação das alterações propostas.

Outro aspecto relevante observado foi a influência das características dos projetos nos resultados obtidos. Sistemas maiores e com arquiteturas mais complexas demandaram maior esforço de análise e apresentaram maior desafio para as ferramentas de IA, enquanto projetos menores e menos complexos favoreceram a identificação de oportunidades de melhoria e a aplicação das modificações sugeridas.

Os resultados também evidenciaram que a utilização de ferramentas de Inteligência Artificial Generativa para refatoração ainda depende de supervisão humana. Embora as ferramentas tenham acelerado a identificação de oportunidades de melhoria e a geração de soluções, foram necessárias validações funcionais, correções de erros de compilação e ajustes de integração em diferentes momentos do experimento. Dessa forma, a automação mostrou-se mais

adequada como mecanismo de apoio ao desenvolvedor do que como substituição integral do processo de refatoração.

Diante dos resultados observados, conclui-se que a refatoração manual apresentou os melhores resultados globais nos atributos de qualidade analisados, especialmente em manutenibilidade, compreensibilidade e controle da complexidade estrutural. As ferramentas de Inteligência Artificial Generativa demonstraram potencial como apoio ao processo de refatoração, contribuindo para a produtividade e para a identificação de oportunidades de melhoria, mas ainda dependem de supervisão humana para garantir a consistência das alterações realizadas.

4.1 Trabalhos Futuros

Os resultados obtidos neste estudo abrem oportunidades para novas investigações sobre a aplicação de Inteligência Artificial Generativa no processo de refatoração de software. Recomenda-se ampliar a análise para um número maior de projetos, contemplando diferentes domínios de aplicação, arquiteturas e linguagens de programação, de modo a aumentar a capacidade de generalização dos resultados.

Outra possibilidade consiste na avaliação de diferentes perfis de desenvolvedores durante a execução da refatoração manual. A comparação entre profissionais com níveis distintos de experiência, como desenvolvedores juniores, plenos e seniores, pode contribuir para compreender em que medida o conhecimento técnico influencia os resultados obtidos e como essa variação se compara ao desempenho das ferramentas de IA Generativa.

Também se destaca a necessidade de investigar o impacto de diferentes estratégias de *prompting* sobre os resultados das ferramentas avaliadas. Neste trabalho foi adotado um único *prompt* padronizado para garantir a comparabilidade entre as abordagens; entretanto, comandos mais específicos, orientados a técnicas de refatoração particulares ou a determinados atributos de qualidade, podem produzir resultados significativamente distintos. Dessa forma, estudos futuros podem explorar a influência da formulação dos *prompts* na qualidade das refatorações geradas.

Por fim, recomenda-se a avaliação de outras ferramentas e modelos de Inteligência Artificial Generativa, incluindo versões futuras das soluções analisadas e novas plataformas disponibilizadas ao mercado. Comparações envolvendo diferentes modelos podem contribuir para identificar quais tecnologias apresentam melhor desempenho em atributos específicos, como manutenibilidade, compreensibilidade, reusabilidade e redução da complexidade estrutural.

REFERÊNCIAS

ALMANASRA, S.; SUWAIS, K. Analysis of ChatGPT-Generated Codes Across Multiple Programming Languages. **IEEE Access**, [*S.l.*], v. 13, p. 23580–23596, fev. 2025.

ALMOGAHED, A. *et al.* A Refactoring Classification Framework for Efficient Software Maintenance. **IEEE Access**, [*S.l.*], v. 11, p. 78904–78917, jul. 2023.

ALOMAR, E. A. Deciphering Refactoring Branch Dynamics in Modern Code Review: An Empirical Study on Qt. **Information and Software Technology**, [*S.l.*], v. 177, p. 107596, jan. 2025.

ALOMAR, E. A. *et al.* On Preserving the Behavior in Software Refactoring: A Systematic Mapping Study. **Information and Software Technology**, [*S.l.*], v. 140, p. 106675, jul. 2021.

ASARE, O.; NAGAPPAN, M.; ASOKAN, N. Is GitHub's Copilot as Bad as Humans at Introducing Vulnerabilities in Code? **Empirical Software Engineer**, v. 28, n. 6, p. 129, set. 2023.

BASILI, V. R.; CALDIERA, G.; ROMBACH, H. D. The Goal Question Metric Approach. *In*: MARCINIAK, J. J. (Ed.). **Encyclopedia of Software Engineering**. Nova Iorque: John Wiley & Sons, 1994. p. 528–532.

CHENG, R. *et al.* "It Would Work For Me Too": How Online Communities Shape Software Developers' Trust in AI-Powered Code Generation Tools. **ACM Transactions on Interactive Intelligent Systems**, v. 14, n. 2, p. 11:1–11:39, maio 2024.

DEPALMA, K. *et al.* Exploring ChatGPT's Code Refactoring Capabilities: An Empirical Study. **Expert Systems with Applications**, [*S.l.*], v. 249, p. 123602, mar. 2024.

DSE. **Repositório do Projeto de Desenvolvimento de Software Educacional**. Ponta Grossa, 2026. Disponível em: https://sites.google.com/view/lesic-softwareeducacional. Acesso em: 29 abr. 2025.

FOWLER, M. **Catalog of Refactorings**, . 2026. Disponível em: https://refactoring.com/catalog. Acesso em: 28 mai. 2025.

FOWLER, M. **Refatoração: Aperfeiçoando o Design de Códigos Existentes**. 2. ed. São Paulo: Novatec Editora, 2020.

GOUVEIA, L. B. Universidade Fernando Pessoa **Inteligência Artificial (IA) e IA Generativa para Geração de Código**. Porto, Portugal: , 2024. (Tecnologia, Redes e Sociedade, 01/2024). Relatório Interno. Disponível em: http://hdl.handle.net/10284/13392. Acesso em: 29 mar. 2025.

HAQUE, M. A. LLMs: A Game-Changer for Software Engineers? **BenchCouncil Transactions on Benchmarks, Standards and Evaluations**, v. 5, n. 1, p. 100204, maio 2025.

HUANG, Y. *et al.* Your Code Secret Belongs to Me: Neural Code Completion Tools Can Memorize Hard-Coded Credentials. **Proceedings of the ACM on Software Engineering**, v. 1, n. FSE, p. 111:2515–111:2537, maio 2024.

LENARDUZZI, V. *et al.* A Critical Comparison on Six Static Analysis Tools: Detection, Agreement, and Precision. **Journal of Systems and Software**, [*S.l.*], v. 198, p. 111575, dez. 2023.

LESIC. **Laboratório de Engenharia de Software e Inteligência Computacional**. Ponta Grossa, 2026. Disponível em: https://dainf.pg.utfpr.edu.br/lesic/site/. Acesso em: 30 abr. 2025.

LIRA, W.; NETO, P. S.; OSORIO, L. Uma Análise do Uso de Ferramentas de Geração de Código por Alunos de Computação. *In*: SIMPÓSIO BRASILEIRO DE EDUCAÇÃO EM COMPUTAÇÃO (EDUCOMP). 4., 2024, Porto Alegre. **Anais [...]** Porto Alegre: Sociedade Brasileira de Computação, 2024. p. 63–71. Evento Online.

NORING, C. *et al.* **AI-Assisted Programming for Web and Machine Learning: Improve Your Development Workflow with ChatGPT and GitHub Copilot**. Birmingham, Inglaterra: Packt Publishing, 2024.

REBAI, S. *et al.* Recommending Refactorings via Commit Message Analysis. **Information and Software Technology**, [*S.l.*], v. 126, p. 106332, maio 2020.

RIEMER, Y. **Aplicação de Métricas Indiretas na Ferramenta Refactoring and Measurement Tool (RMT)**. 2024. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) — Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2024.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Prentice Hall, 2011.

APÊNDICES

APÊNDICE A – TABELAS COMPLETAS DE MÉTRICAS DE QUALIDADE

Este apêndice apresenta as tabelas completas contendo os dados quantitativos utilizados para avaliar o impacto das refatorações aplicadas nos projetos analisados. Foram consideradas métricas de qualidade de código coletadas antes e após as refatorações, tanto manuais quanto com o uso de ferramentas de Inteligência Artificial Generativa, possibilitando uma base para comparações posteriores. As métricas foram obtidas por meio da ferramenta SonarQube, empregada na indústria e na academia para análise estática de código, garantindo padronização no processo de coleta e confiabilidade nos resultados obtidos.

Inicialmente, foram registrados os dados correspondentes ao estado original dos projetos, estabelecendo uma linha de base para avaliação. Em seguida, foram realizadas novas coletas após a aplicação das refatorações manuais e, posteriormente, após as refatorações assistidas por IA Generativa, mantendo-se o mesmo conjunto de métricas e condições de análise em todas as etapas. As subseções a seguir apresentam os dados coletados para cada projeto e abordagem, evidenciando as variações observadas nas métricas. Esses dados servirão de base para a análise comparativa apresentada na seção 3.2.

A.1 Dados Iniciais

Antes da aplicação das refatorações, foram coletadas as métricas iniciais de qualidade de código dos três projetos selecionados, utilizando a ferramenta SonarQube. Essas métricas abrangem aspectos relacionados à segurança, confiabilidade, manutenibilidade e duplicação de código, além de indicadores complementares, como número de linhas de código e complexidade. Os dados obtidos nesta etapa estabelecem uma linha de base para a avaliação dos impactos das refatorações realizadas, permitindo a comparação com os resultados posteriores.

A.1.1 Memoletrando

O projeto Memoletrando apresenta um total de 22.830 linhas de código, distribuídas entre as tecnologias Angular, Ionic e Laravel. Os dados obtidos indicam ausência de problemas de segurança identificados na análise inicial. Em contrapartida, foram registradas ocorrências relevantes nas dimensões de confiabilidade e, principalmente, de manutenibilidade, além de uma taxa significativa de duplicação de código. Também se destacam os valores associados à complexidade ciclomática e cognitiva, bem como o volume geral do sistema, evidenciando características estruturais importantes do projeto em seu estado original.

A Tabela 4 apresenta as métricas coletadas no projeto Memoletrando.

Tabela 4 – Métricas Iniciais do Memoletrando

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	143
	Esforço de Correção	1d 2h
Manutenibilidade	Issues	896
	Esforço de Correção	7d 4h
Duplicação	Densidade	9,1%
	Linhas Duplicadas	2.834
	Blocos Duplicados	137
	Arquivos Duplicados	61
Tamanho	Linhas de Código (LOC)	22.830
	Linhas	32.003
	Declarações (Statements)	5.380
	Funções	1.834
	Classes	352
	Arquivos	563
	Linhas de Comentário	1.514
	Comentários (%)	6,2%
Complexidade	Ciclomática	2.731
	Cognitiva	1.245

Fonte: Autoria própria (2026).

A.1.2 Querida Floresta 2.0

O projeto Querida Floresta 2.0 apresenta o maior volume de código entre os casos analisados, totalizando 49.137 linhas. No entanto, uma parcela significativa desse volume corresponde a dependências externas e arquivos de configuração, não refletindo diretamente a lógica principal da aplicação. Dessa forma, a análise de métricas foi concentrada no diretório `Project/Scripts`, que reúne os componentes centrais do sistema.

Os resultados indicam a presença de um número reduzido de problemas de segurança e uma quantidade moderada de ocorrências relacionadas à confiabilidade e à manutenibilidade. Em relação à duplicação de código, observa-se uma densidade geral relativamente baixa, embora haja variações entre diferentes partes do sistema. A Tabela 5 apresenta as métricas coletadas para o diretório analisado.

Tabela 5 – Métricas Iniciais do Querida Floresta 2.0

Categoria	Métrica	Valor
Segurança	Issues	1
	Esforço de Correção	30min
Confiabilidade	Issues	65
	Esforço de Correção	5h 25min
Manutenibilidade	Issues	540

(continua)

Tabela 5 – Métricas Iniciais do Querida Floresta 2.0

(continuação)

Categoria	Métrica	Valor
	Esforço de Correção	5d 6h
Duplicação	Densidade	5,8%
	Linhas Duplicadas	267
	Blocos Duplicados	9
	Arquivos Duplicados	7
Tamanho	Linhas de Código (LOC)	3.764
	Linhas	4.627
	Declarações (Statements)	1.621
	Funções	435
	Classes	65
	Arquivos	65
	Linhas de Comentário	146
	Comentários (%)	3,7%
Complexidade	Ciclomática	748
	Cognitiva	471

Fonte: Autoria própria (2026).

A.1.3 SpeakSnake

O projeto SpeakSnake, embora apresente o menor volume de código entre os casos analisados, totalizando 1.205 linhas de código principais, apresentou problemas nas dimensões de segurança, confiabilidade e manutenibilidade, além de uma taxa relativamente elevada de duplicação de código em comparação ao porte do sistema. Também se destacam os valores associados à complexidade ciclomática e cognitiva, bem como a distribuição geral dos elementos estruturais do projeto.

A Tabela 6 apresenta os dados coletados nesta etapa.

Tabela 6 – Métricas Iniciais do SpeakSnake

Categoria	Métrica	Valor
Segurança	Issues	1
	Esforço de Correção	45min
Confiabilidade	Issues	12
	Esforço de Correção	58min
Manutenibilidade	Issues	115
	Esforço de Correção	1d
Duplicação	Densidade	12,6%
	Linhas Duplicadas	226
	Blocos Duplicados	8
	Arquivos Duplicados	2
Tamanho	Linhas de Código (LOC)	1.205
	Linhas	1.797
	Declarações (Statements)	649
	Funções	64

(continua)

Tabela 6 – Métricas Iniciais do SpeakSnake

(continuação)

Categoria	Métrica	Valor
	Classes	0
	Arquivos	11
	Linhas de Comentário	110
	Comentários (%)	8,4%
Complexidade	Ciclomática	171
	Cognitiva	167

Fonte: Autoria própria (2026).

A.2 Dados Pós-Refatoração Manual

Após a aplicação das refatorações manuais, foram realizadas novas coletas das métricas de qualidade de código dos projetos, utilizando a ferramenta SonarQube. Essa etapa teve como objetivo registrar os valores obtidos após as modificações estruturais, mantendo as mesmas condições e critérios adotados na coleta inicial. As métricas consideradas abrangem aspectos relacionados à segurança, confiabilidade, manutenibilidade, duplicação de código, tamanho e complexidade, permitindo a observação das variações ocorridas após a intervenção manual.

As subseções a seguir apresentam os dados coletados para cada projeto após a refatoração manual.

A.2.1 Memoletrando

Após a aplicação da refatoração manual no projeto Memoletrando, foram coletadas novamente as métricas de qualidade de código por meio da ferramenta SonarQube. Os dados obtidos indicam alterações nos indicadores de confiabilidade e manutenibilidade, bem como na densidade de duplicação de código. Também são observadas variações nos valores associados à complexidade e à estrutura geral do sistema, incluindo número de linhas, funções, classes e arquivos. Esses resultados refletem as modificações realizadas durante o processo de refatoração manual.

A Tabela 7 apresenta as métricas coletadas após a refatoração, permitindo a futura comparação com os dados do estado inicial do projeto.

Tabela 7 – Métricas Pós-Refatoração Manual do Memoletrando

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	0
	Esforço de Correção	0
Manutenibilidade	Issues	5

(continua)

Tabela 7 – Métricas Pós-Refatoração Manual do Memoletrando

(continuação)

Categoria	Métrica	Valor
	Esforço de Correção	25min
Duplicação	Densidade	5,2%
	Linhas Duplicadas	1.614
	Blocos Duplicados	76
	Arquivos Duplicados	37
Tamanho	Linhas de Código (LOC)	22.734
	Linhas	31.195
	Declarações (Statements)	5.300
	Funções	1.949
	Classes	346
	Arquivos	536
	Linhas de Comentário	1.408
	Comentários (%)	5,8%
Complexidade	Ciclomática	2.715
	Cognitiva	844

Fonte: Autoria própria (2026).

A.2.2 Querida Floresta 2.0

Após a aplicação da refatoração manual no projeto Querida Floresta 2.0, foram coletadas novamente as métricas de qualidade de código por meio da ferramenta SonarQube, mantendo-se o mesmo escopo de análise no diretório `Project/Scripts` e as mesmas condições adotadas na etapa inicial. Os dados obtidos indicam alterações nos indicadores dessas dimensões, incluindo a ausência de ocorrências nas categorias de segurança, confiabilidade e manutenibilidade, bem como variações na densidade de duplicação de código. Também são observadas mudanças nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas durante o processo de refatoração manual.

A Tabela 8 apresenta as métricas coletadas após a refatoração.

Tabela 8 – Métricas Pós-Refatoração Manual do Querida Floresta 2.0

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	0
	Esforço de Correção	0
Manutenibilidade	Issues	0
	Esforço de Correção	0
Duplicação	Densidade	0,0%
	Linhas Duplicadas	0
	Blocos Duplicados	0
	Arquivos Duplicados	0
Tamanho	Linhas de Código (LOC)	3.863

(continua)

Tabela 8 – Métricas Pós-Refatoração Manual do Querida Floresta 2.0

(continuação)

Categoria	Métrica	Valor
	Linhas	5.053
	Declarações (Statements)	1.283
	Funções	592
	Classes	73
	Arquivos	67
	Linhas de Comentário	313
	Comentários (%)	7,5%
Complexidade	Ciclomática	847
	Cognitiva	307

Fonte: Autoria própria (2026).

A.2.3 SpeakSnake

Após a aplicação da refatoração manual no projeto SpeakSnake, foram coletadas novamente as métricas de qualidade de código por meio da ferramenta SonarQube, mantendo-se os mesmos critérios e condições adotados na etapa inicial. Os dados obtidos indicam alterações nos indicadores dessas dimensões, incluindo a ausência de ocorrências nas categorias de segurança e confiabilidade, bem como variações nos níveis de manutenibilidade e na densidade de duplicação de código. Também são observadas mudanças nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas durante o processo de refatoração manual.

A Tabela 9 apresenta as métricas coletadas após a refatoração manual no SpeakSnake, permitindo a posterior comparação com os dados do estado inicial do projeto.

Tabela 9 – Métricas Pós-Refatoração Manual do SpeakSnake

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	0
	Esforço de Correção	0
Manutenibilidade	Issues	2
	Esforço de Correção	10min
Duplicação	Densidade	0,0%
	Linhas Duplicadas	0
	Blocos Duplicados	0
	Arquivos Duplicados	0
Tamanho	Linhas de Código (LOC)	1.212
	Linhas	1.837
	Declarações (Statements)	497
	Funções	72
	Classes	11
	Arquivos	17

(continua)

Tabela 9 – Métricas Pós-Refatoração Manual do SpeakSnake

(continuação)

Categoria	Métrica	Valor
	Linhas de Comentário	85
	Comentários (%)	6,6%
Complexidade	Ciclomática	154
	Cognitiva	89

Fonte: Autoria própria (2026).

A.3 Dados Pós-Refatoração por IA Generativa

Após a aplicação das refatorações com o auxílio de ferramentas de Inteligência Artificial Generativa, foram realizadas novas coletas das métricas de qualidade de código por meio da ferramenta SonarQube, mantendo-se os mesmos critérios e condições adotados nas etapas anteriores. As refatorações foram conduzidas com o uso das ferramentas GitHub Copilot e Amazon Q Developer, conforme descrito na seção 3.1. As métricas consideradas abrangem aspectos relacionados à segurança, confiabilidade, manutenibilidade, duplicação de código, tamanho e complexidade, permitindo a observação das variações ocorridas após a aplicação das sugestões automatizadas.

As subseções a seguir apresentam, de forma objetiva, os dados coletados para cada projeto e ferramenta utilizada, servindo como base para as análises comparativas desenvolvidas nas seções posteriores.

A.3.1 Memoletrando

Após a aplicação da refatoração no projeto Memoletrando com o auxílio da ferramenta GitHub Copilot, foram coletadas novamente as métricas de qualidade de código por meio do SonarQube. Os dados obtidos indicam variações nos indicadores dessas dimensões, incluindo mudanças nos níveis de manutenibilidade e confiabilidade, bem como na densidade de duplicação de código. Também são observadas alterações nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas com o suporte da ferramenta.

A Tabela 10 apresenta as métricas coletadas após a refatoração, permitindo a comparação com os dados do estado inicial do projeto e com os resultados das demais abordagens.

Tabela 10 – Métricas Pós-Refatoração pelo GitHub Copilot do Memoletrando

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	153

(continua)

Tabela 10 – Métricas Pós-Refatoração pelo GitHub Copilot do Memoletrando

(continuação)

Categoria	Métrica	Valor
	Esforço de Correção	1d 3h
Manutenibilidade	Issues	808
	Esforço de Correção	6d 1h
Duplicação	Densidade	5,8%
	Linhas Duplicadas	2.216
	Blocos Duplicados	114
	Arquivos Duplicados	54
Tamanho	Linhas de Código (LOC)	29.374
	Linhas	38.117
	Declarações (Statements)	5.879
	Funções	1.936
	Classes	372
	Arquivos	586
	Linhas de Comentário	1.633
	Comentários (%)	5,3%
Complexidade	Ciclomática	2.961
	Cognitiva	1.262

Fonte: Autoria própria (2026).

Após a aplicação da refatoração no projeto Memoletrando com o auxílio da ferramenta Amazon Q Developer, foram coletadas novamente as métricas de qualidade de código por meio do SonarQube. Os dados obtidos indicam variações nos indicadores dessas dimensões, incluindo mudanças nos níveis de manutenibilidade e confiabilidade, bem como na densidade de duplicação de código. Também são observadas alterações nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas com o suporte da ferramenta.

A Tabela 11 apresenta as métricas coletadas após a refatoração, permitindo a futura comparação com os dados do estado inicial do projeto e com os resultados das demais abordagens.

Tabela 11 – Métricas Pós-Refatoração pelo Amazon Q Developer do Memoletrando

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	147
	Esforço de Correção	1d 3h
Manutenibilidade	Issues	716
	Esforço de Correção	5d 6h
Duplicação	Densidade	4,0%
	Linhas Duplicadas	1.202
	Blocos Duplicados	79
	Arquivos Duplicados	35
Tamanho	Linhas de Código (LOC)	21.898

(continua)

Tabela 11 – Métricas Pós-Refatoração pelo Amazon Q Developer do Memoletrando (continuação)

Categoria	Métrica	Valor
	Linhas	29.796
	Declarações (Statements)	4.812
	Funções	1.767
	Classes	365
	Arquivos	580
	Linhas de Comentário	1.353
	Comentários (%)	5,8%
Complexidade	Ciclomática	2.577
	Cognitiva	984

Fonte: Autoria própria (2026).

A.3.2 Querida Floresta 2.0

Após a aplicação da refatoração no projeto Querida Floresta 2.0 com o auxílio da ferramenta GitHub Copilot, foram coletadas novamente as métricas de qualidade de código por meio do SonarQube, mantendo-se o mesmo escopo de análise no diretório `Project/Scripts` e as mesmas condições adotadas na etapa inicial. Os dados obtidos indicam variações nos indicadores dessas dimensões, incluindo alterações nos níveis de segurança, confiabilidade e manutenibilidade, bem como na densidade de duplicação de código. Também são observadas mudanças nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas com o suporte da ferramenta.

A Tabela 12 apresenta as métricas coletadas após a refatoração automatizada no projeto pelo GitHub Copilot.

Tabela 12 – Métricas Pós-Refatoração pelo GitHub Copilot do Querida Floresta 2.0

Categoria	Métrica	Valor
Segurança	Issues	1
	Esforço de Correção	30min
Confiabilidade	Issues	1
	Esforço de Correção	5min
Manutenibilidade	Issues	196
	Esforço de Correção	1d 5h
Duplicação	Densidade	5,2%
	Linhas Duplicadas	333
	Blocos Duplicados	9
	Arquivos Duplicados	5
Tamanho	Linhas de Código (LOC)	4.777
	Linhas	6.439
	Declarações (Statements)	2.094
	Funções	558
	Classes	66
	Arquivos	65

(continua)

Tabela 12 – Métricas Pós-Refatoração pelo GitHub Copilot do Querida Floresta 2.0 (continuação)

Categoria	Métrica	Valor
	Linhas de Comentário	993
	Comentários (%)	17,2%
Complexidade	Ciclomática	1.399
	Cognitiva	1.058

Fonte: Autoria própria (2026).

Após a aplicação da refatoração no projeto Querida Floresta 2.0 com o auxílio da ferramenta Amazon Q Developer, foram coletadas novamente as métricas de qualidade de código por meio do SonarQube, mantendo-se o mesmo escopo de análise no diretório `Project/Scripts` e as mesmas condições adotadas na etapa inicial. Os dados obtidos indicam variações nos indicadores dessas dimensões, incluindo alterações nos níveis de confiabilidade e manutenibilidade, bem como na densidade de duplicação de código. Também são observadas mudanças nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas com o suporte da ferramenta.

A Tabela 13 apresenta, de forma detalhada, as métricas coletadas após a refatoração, permitindo a posterior comparação com os dados do estado inicial do projeto e com os resultados das demais abordagens.

Tabela 13 – Métricas Pós-Refatoração pelo Amazon Q Developer do Querida Floresta 2.0

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	65
	Esforço de Correção	5h 25min
Manutenibilidade	Issues	657
	Esforço de Correção	1d 3h
Duplicação	Densidade	11,4%
	Linhas Duplicadas	807
	Blocos Duplicados	19
	Arquivos Duplicados	14
Tamanho	Linhas de Código (LOC)	5.763
	Linhas	7.053
	Declarações (Statements)	2.093
	Funções	846
	Classes	75
	Arquivos	71
	Linhas de Comentário	84
	Comentários (%)	1,4%
Complexidade	Ciclomática	1.902
	Cognitiva	615

Fonte: Autoria própria (2026).

A.3.3 SpeakSnake

Após a aplicação da refatoração no projeto SpeakSnake com o auxílio da ferramenta GitHub Copilot, foram coletadas novamente as métricas de qualidade de código por meio do SonarQube. Os dados obtidos indicam variações nos indicadores dessas dimensões, incluindo alterações nos níveis de segurança, confiabilidade e manutenibilidade, bem como na densidade de duplicação de código. Também são observadas mudanças nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas com o suporte da ferramenta.

A Tabela 14 apresenta as métricas coletadas após a refatoração, permitindo a comparação com os dados do estado inicial do projeto e com os resultados das demais abordagens.

Tabela 14 – Métricas Pós-Refatoração pelo GitHub Copilot do SpeakSnake

Categoria	Métrica	Valor
Segurança	Issues	0
	Esforço de Correção	0
Confiabilidade	Issues	9
	Esforço de Correção	45min
Manutenibilidade	Issues	30
	Esforço de Correção	2d 43min
Duplicação	Densidade	14,3%
	Linhas Duplicadas	242
	Blocos Duplicados	8
	Arquivos Duplicados	2
Tamanho	Linhas de Código (LOC)	1.145
	Linhas	1.698
	Declarações (Statements)	588
	Funções	69
	Classes	0
	Arquivos	11
	Linhas de Comentário	73
	Comentários (%)	6,0%
Complexidade	Ciclomática	188
	Cognitiva	187

Fonte: Autoria própria (2026).

Após a aplicação da refatoração no projeto SpeakSnake com o auxílio da ferramenta Amazon Q Developer, foram coletadas novamente as métricas de qualidade de código por meio do SonarQube. Os dados obtidos indicam variações nos indicadores dessas dimensões, incluindo alterações nos níveis de segurança, confiabilidade e manutenibilidade, bem como na densidade de duplicação de código. Também são observadas mudanças nos valores associados ao tamanho do sistema e às métricas de complexidade, refletindo as modificações realizadas com o suporte da ferramenta.

A Tabela 15 apresenta, de forma detalhada, as métricas coletadas após a refatoração, permitindo a futura comparação com os dados do estado inicial do projeto e com os resultados das demais abordagens.

Tabela 15 – Métricas Pós-Refatoração pelo Amazon Q Developer do SpeakSnake

Categoria	Métrica	Valor
Segurança	Issues	3
	Esforço de Correção	2h 15min
Confiabilidade	Issues	9
	Esforço de Correção	47min
Manutenibilidade	Issues	119
	Esforço de Correção	1d
Duplicação	Densidade	16,6%
	Linhas Duplicadas	523
	Blocos Duplicados	18
	Arquivos Duplicados	4
Tamanho	Linhas de Código (LOC)	2.350
	Linhas	3.142
	Declarações (Statements)	1.136
	Funções	155
	Classes	0
	Arquivos	19
	Linhas de Comentário	104
	Comentários (%)	4,2%
Complexidade	Ciclomática	376
	Cognitiva	325

Fonte: Autoria própria (2026).