

Universidade Federal do Piauí
Centro de Ciências da Natureza
Programa de Pós-Graduação em Ciência da Computação

Investigando ferramentas de apoio à programação baseadas em IA: do cenário controlado ao ambiente de produção

Luiz Fernando Mendes Osorio

Número de Ordem PPGCC: M151

Teresina-PI, Agosto de 2025

Luiz Fernando Mendes Osorio

Investigando ferramentas de apoio à programação baseadas em IA: do cenário controlado ao ambiente de produção

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Universidade Federal do Piauí – UFPI

Centro de Ciências da Natureza

Programa de Pós-Graduação em Ciência da Computação

Orientador: Pedro de Alcântara dos Santos Neto

Coorientador: Guilherme Amaral Avelino

Teresina-PI

Agosto de 2025

FICHA CATALOGRÁFICA
Universidade Federal do Piauí
Sistema de Bibliotecas UFPI - SIBi/UFPI
Biblioteca Setorial do CCN

O83i Osório, Luíz Fernando Mendes.
Investigando ferramentas de apoio à programação baseadas em IA: do cenário controlado ao ambiente de produção / Luíz Fernando Mendes Osório. -- 2025.
99 f.: il. color.

Dissertação (Mestrado) - Universidade Federal do Piauí. Centro de Ciências da Natureza. Programa de Pós-Graduação em Ciências da Computação, Teresina, 2025.
"Orientador: Prof. Dr. Pedro de Alcântara dos Santos Neto. Coorientador: Prof. Dr. Guilherme Amaral Avelino".

1. Inteligência Artificial - IA. 2. Copilot. 3. Geração de Códigos. I. Santos Neto, Pedro de Alcântara dos. II. Avelino, Guilherme Amaral. III. Titulo.

CDD 006.3

Bibliotecária: Caryne Maria da Silva Gomes - CRB3/1461

Luiz Fernando Mendes Osorio

Investigando ferramentas de apoio à programação baseadas em IA: do cenário controlado ao ambiente de produção

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da UFPI (área de concentração: Sistemas de Computação), como parte dos requisitos necessários para a obtenção do Título de Mestre em Ciência da Computação.

Trabalho aprovado. Teresina-PI, Agosto de 2025:

Documento assinado digitalmente
gov.br
PEDRO DE ALCANTARA DOS SANTOS NETO
Data: 04/09/2025 04:30:44-0300
Verifique em https://validar.iti.gov.br

Pedro de Alcântara dos Santos Neto
Orientador

Documento assinado digitalmente
gov.br
GUILHERME AMARAL AVELINO
Data: 03/09/2025 16:40:12-0300
Verifique em https://validar.iti.gov.br

Guilherme Amaral Avelino
Co-Orientador

Documento assinado digitalmente
gov.br
RAIMUNDO SANTOS MOURA
Data: 01/09/2025 15:19:21-0300
Verifique em https://validar.iti.gov.br

Raimundo Santos Moura

Documento assinado digitalmente
gov.br
MARCO TULIO DE OLIVEIRA VALENTE
Data: 27/08/2025 13:18:06-0300
Verifique em https://validar.iti.gov.br

Marco Tulio Valente
Membro Externo

Teresina-PI
Agosto de 2025

Dedico este trabalho aos meus pais, Rosa e Carlos,
por todo o amor, apoio e incentivo incondicional ao longo da minha jornada. Sem a paciência, compreensão e força que me transmitiram, este momento não seria possível.

Agradecimentos

Agradeço aos meus pais, Rosa e Carlos, por todo o amor, apoio e incentivo incondicional, que sempre foram fundamentais em cada passo dessa jornada.

Ao meu marido, Rodrigo Coelho, pelo amor, compreensão e companheirismo constante. Sua presença ao meu lado, em cada desafio e conquista, tornou essa caminhada muito mais leve e significativa.

Aos meus irmãos Carlos Mendes e Rafael, pela amizade e por sempre estarem ao meu lado, compartilhando os desafios e as conquistas.

Agradeço ao meu orientador, Prof. Dr. Pedro Alcântara, e ao meu coorientador, Prof. Dr. Guilherme Avelino, por todos os conselhos, pela paciência e pela orientação valiosa ao longo desse período. Sua dedicação e apoio foram essenciais para a realização deste trabalho.

Aos meus amigos, por estarem comigo nos momentos de descontração e por sempre me motivarem a seguir em frente, mesmo nos momentos mais difíceis.

Aos professores, que contribuíram com seus ensinamentos e orientações, ajudando a expandir meus conhecimentos e a fortalecer minha formação ao longo da trajetória acadêmica. Um agradecimento especial ao Prof. José Pereira, que, ainda no Ensino Fundamental, me incentivou a trilhar os caminhos que me trouxeram até aqui e que, mesmo após seu falecimento, permanece vivo em minha memória e gratidão.

A todos que, de alguma forma, contribuíram para a realização deste trabalho, meu sincero agradecimento.

"O espírito humano
precisa prevalecer sobre
a tecnologia."
(Albert Einstein)

Resumo

Este trabalho investigou o impacto do uso de ferramentas de geração de código assistido por Inteligência Artificial (IA), com foco na avaliação do GitHub Copilot, a partir de uma abordagem metodológica estruturada em três etapas: (i) uma fase de investigação, com levantamento da literatura e identificação dos principais desafios e indicadores utilizados na avaliação de ferramentas de geração de código; (ii) uma etapa de coleta de dados, com a realização de um estudo empírico com estudantes e um estudo de caso com desenvolvedores profissionais; e (iii) uma fase de análise, composta por procedimentos quantitativos e qualitativos que permitiram responder às questões de pesquisa e interpretar os efeitos do uso da ferramenta em diferentes contextos de desenvolvimento de software.

Os resultados revelam que o Copilot pode reduzir o tempo de execução das tarefas e o ritmo de alterações (LOC/h), entre estudantes e profissionais em tarefas rotineiras, respectivamente. No entanto, não foram observadas melhorias significativas na qualidade funcional do código, reforçando a necessidade de uso crítico das sugestões geradas pela ferramenta. A análise qualitativa complementou esses achados ao indicar que a percepção dos participantes foi majoritariamente positiva, embora tenham sido apontadas limitações técnicas e operacionais. Assim, este estudo contribui com evidências empíricas sobre o uso do Copilot, destacando tanto seu potencial como suporte à redução do esforço de desenvolvimento quanto a importância de estratégias formativas para sua adoção consciente em ambientes educacionais e profissionais.

Palavras-chave: Geração de código assistido por IA. Copilot. Avaliação empírica. Esforço de desenvolvimento.

Abstract

This study investigated the impact of using artificial intelligence (AI)-assisted code generation tools, focusing on GitHub Copilot, through a methodological approach structured in three stages: (i) an investigation phase, including a literature review and the identification of key challenges and indicators used to evaluate code generation tools; (ii) a data collection phase, consisting of an empirical study with students and a case study with professional developers; and (iii) an analysis phase, involving quantitative and qualitative procedures to answer the research questions and interpret the effects of using the tool in different software development contexts.

The results reveal that Copilot can significantly reduce task execution time and development effort, especially among undergraduate students and professionals performing routine tasks. However, no significant improvements were observed in the functional quality of the code, reinforcing the need for critical use of the tool's suggestions. The qualitative analysis supported these findings, showing that participants' perceptions were mostly positive, although technical and operational limitations were noted. Thus, this study provides empirical evidence on the use of Copilot, highlighting both its potential as a support tool to reduce development effort and the importance of formative strategies for its conscious adoption in educational and professional environments.

Keywords: AI-assisted code generation. Copilot. Empirical evaluation. Development effort.

Lista de ilustrações

Figura 1 – Fluxo de atividades na metodologia do estudo.
Fonte: autor. 30
Figura 2 – Exemplo de uma tarefa no sistema de gerenciamento da STI, evidenciando a associação com commits no repositório Git. 37
Figura 3 – Distribuição Normalizada Dos Indicadores por Uso do Copilot 42
Figura 4 – Distribuição do Tempo Gasto por Turma e Uso do Copilot 46
Figura 5 – Distribuição do Número de Testes Falhos por Turma e Uso do Copilot 46
Figura 6 – Problema #1.
Fonte: autor. 87
Figura 7 – Problema #2.
Fonte: autor. 88
Figura 8 – Problema #3.
Fonte: autor. 88
Figura 9 – Problema #4.
Fonte: autor. 89

Lista de tabelas

Tabela 1 – Comparação de Trabalhos Relacionados sobre Ferramentas de Geração de Código Assistido por IA. 20
Tabela 2 – Associação entre o ID do problema e o método removido nos sistemas. 33
Tabela 3 – Exemplo de distribuição dos problemas por participantes. 34
Tabela 4 – Distribuição das respostas por questão e uso do Copilot 40
Tabela 5 – Estatísticas Descritivas do Tempo Gasto (em minutos), Separadas pelo Uso do Copilot 41
Tabela 6 – Estatísticas Descritivas do Número de Testes Falhos, Separadas pelo Uso do Copilot 41
Tabela 7 – Estatísticas Descritivas do Tempo Gasto (min), por Turma e Uso do Copilot 45
Tabela 8 – Estatísticas Descritivas do Número de Testes Falhos, por Turma e Uso do Copilot 45
Tabela 9 – Distribuição dos Participantes por Turma 47
Tabela 10 – Tempo de Experiência com Programação 48
Tabela 11 – Autoavaliação das Habilidades de Programação (Escala de Likert) . . . 48
Tabela 12 – Distribuição do Esforço no Aprendizado das Tecnologias (Escala de Likert) 48
Tabela 13 – Avaliação Geral das Oficinas 49
Tabela 14 – Avaliação da Quantidade de Oficinas 50
Tabela 15 – Impacto Percebido do Formato On-line 50
Tabela 16 – Avaliação das Gravações das Oficinas 50
Tabela 17 – Desafios Relatados no Uso do GitHub Copilot 51
Tabela 18 – Confiança para Manutenção com o uso do Copilot 52
Tabela 19 – Quantidade de tarefas por desenvolvedor, tipo e período (com e sem Copilot, corte trimestral) 57
Tabela 20 – Ritmo de alterações no código (LOC/h) por desenvolvedor e período . 58
Tabela 21 – Ritmo de alterações no código (LOC/h) da equipe por período (recorte trimestral) 59
Tabela 22 – Ritmo de alterações (LOC/h) por tipo de tarefa 60
Tabela 23 – Resumo das respostas fechadas dos desenvolvedores participantes . . . 62

Lista de abreviaturas e siglas

API	Application Programming Interface
IDE	Integrated Development Environment (Ambiente de Desenvolvimento Integrado)
IA	Inteligência Artificial
LoC	Lines of Code (Linhas de Código)
LLM	Large Language Model (Modelo de Linguagem de Grande Escala)
VSCode	Visual Studio Code

Sumário

1 INTRODUÇÃO 1
1.1 Contexto e Motivação 1
1.2 Problema de Pesquisa, Objetivos e Escopo 3
1.3 Questões de Pesquisa 4
1.4 Estrutura do Trabalho 5

2 REFERENCIAL TEÓRICO 7
2.1 Evolução das Ferramentas de Geração de Código 7
2.1.1 Ferramentas de Geração de Código 7
2.1.2 A Era dos Modelos de Linguagem e a Integração de IA 8
2.1.3 Geração de Código via GitHub Copilot 8
2.2 Avaliação de Ferramentas de Geração de Código 9
2.2.1 Indicadores de Avaliação 9
2.2.1.1 Qualidade do Código 9
2.2.1.2 Eficiência 10
2.2.1.3 Usabilidade 10
2.2.1.4 Outros Indicadores Adicionais 11
2.3 Estudos Empíricos em Engenharia de *Software* 12
2.4 Considerações Finais 13

3 TRABALHOS RELACIONADOS 15
3.1 Considerações Finais 26

4 METODOLOGIA 29
4.1 Investigação 29
4.2 Coleta de Dados 31
4.2.1 Estudo Empírico com Estudantes 31
4.2.1.1 Treinamento Inicial dos Participantes 32
4.2.1.2 Sessões do Experimento 33
4.2.1.3 Dados Coletados 34
4.2.2 Estudo de Caso 35
4.2.2.1 Dados Coletados 36
4.3 Análise dos Dados 36
4.4 Resultados Alcançados 37
4.5 Considerações Finais 38

5 RESULTADOS . . . 39
5.1 Introdução . . . 39
5.2 Estudo Empírico com Estudantes . . . 39
5.2.1 Análise dos Resultados . . . 39
5.2.2 Análise Exploratória dos Dados . . . 40
5.2.2.1 Estatísticas Descritivas . . . 40
5.2.3 Análise Visual dos Indicadores Normalizados . . . 41
5.2.3.1 Análise de Normalidade e Teste de Hipótese . . . 43
5.2.3.2 Cliff's Delta . . . 43
5.2.4 Análise Comparativa entre Turmas . . . 44
5.2.5 Análise Qualitativa . . . 47
5.2.5.1 Perfil dos Participantes . . . 47
5.2.5.2 Esforço no Aprendizado das Tecnologias . . . 48
5.2.5.3 Percepção das Oficinas . . . 49
5.2.5.4 Desafios ao Usar o Copilot . . . 50
5.2.5.5 Confiança na Manutenção com Copilot . . . 51
5.2.6 Discussão dos Resultados do Estudo Empírico . . . 52
5.3 Estudo de Caso em Ambiente Profissional . . . 53
5.3.1 Caracterização dos Desenvolvedores Participantes . . . 55
5.3.2 Descrição das Tarefas Analisadas . . . 55
5.3.3 Preparação dos Dados para Análise do Esforço de Desenvolvimento . . . 57
5.3.4 Ritmo de Alterações por Desenvolvedor . . . 58
5.3.5 Ritmo de Alterações Agregado da Equipe . . . 59
5.3.6 Análise do Ritmo de Alterações por Tipo de Tarefa . . . 59
5.3.7 Análise Qualitativa das Percepções sobre o Uso do GitHub Copilot . . . 60
5.3.8 Discussão dos Resultados do Estudo de Caso . . . 62
5.4 Desafios na Condução do Estudo de Caso . . . 63
5.5 Lições Aprendidas . . . 64
5.6 Ameaças à Validade . . . 66
5.7 Considerações Finais . . . 67

6 CONCLUSÕES E TRABALHOS FUTUROS . . . 69
6.1 Conclusões . . . 69
6.2 Trabalhos Futuros . . . 70

REFERÊNCIAS . . . 73

APÊNDICES 79

APÊNDICE A – QUESTIONÁRIO SOLUÇÃO DESENVOLVIDA . 81

APÊNDICE B – QUESTIONÁRIO FINAL 83

APÊNDICE C – TRECHOS DO CÓDIGO A SEREM RECRIADOS 87

APÊNDICE D – QUESTIONÁRIO APLICADO NO ESTUDO DE CASO COM DESENVOLVEDORES PROFISSIONAIS 91

APÊNDICE E – TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO. 95

1 Introdução

Neste capítulo são apresentados os elementos que fundamentam esta pesquisa, estabelecendo o contexto e a motivação para o estudo das ferramentas de geração de código assistido por Inteligência Artificial - IA no desenvolvimento de *software*. Em seguida, o problema de pesquisa é delineado, junto aos objetivos geral e específicos, que orientam a investigação. As questões de pesquisa, formuladas para guiar a análise dos fenômenos observados, também são detalhadas. O capítulo encerra com uma visão geral que antecipa a estrutura dos próximos capítulos e o desenvolvimento do trabalho.

1.1 Contexto e Motivação

O software assumiu um papel central na vida moderna, transformando desde as interações pessoais até os processos mais complexos de negócios e serviços públicos. Em nosso dia a dia, softwares controlam a forma como nos comunicamos, trabalhamos, nos divertimos e até cuidamos da saúde. Dispositivos móveis, computadores, eletrodomésticos inteligentes, carros autônomos, sistemas de pagamento e inúmeras outras ferramentas e tecnologias estão todos interligados e operam através de softwares cada vez mais sofisticados. Esse cenário demonstra como o software se tornou indispensável para o funcionamento da sociedade moderna, sendo um elemento essencial para a continuidade e evolução das atividades econômicas e sociais (KITCHIN; DODGE, 2014; HERBSLEB; MOITRA, 2001).

Com essa importância crescente, surge uma demanda por softwares mais robustos, eficientes e confiáveis. Organizações de todos os setores precisam desenvolver sistemas que sejam escaláveis, capazes de processar grandes volumes de dados e que ofereçam segurança e desempenho superior. E além disso, esses softwares precisam ser entregues rapidamente para atender à competitividade do mercado e às expectativas dos consumidores (PENG et al., 2023). O tempo de desenvolvimento, antes espaçado e linear, é agora acelerado por metodologias, que integram continuamente novas funcionalidades e atualizações ao longo do ciclo de vida do software (SHAH, 2019). A necessidade de inovação constante coloca pressão sobre as equipes de desenvolvimento para criar e aprimorar softwares em um ritmo muito mais veloz e com margens de erro reduzidas.

Esse cenário também intensifica a importância da manutenção e atualização dos softwares existentes, que precisam acompanhar a rápida evolução tecnológica e as mudanças nas regulamentações e nas necessidades dos usuários (PRESSMAN, 2009; SOMMERVILLE, 2016). Softwares precisam não apenas ser desenvolvidos, mas mantidos de forma constante para garantir a funcionalidade e a relevância a longo prazo. Ferramentas como inteligência artificial e automação vêm sendo cada vez mais aplicadas para auxiliar nessa tarefa,

permitindo que as equipes se concentrem em desafios mais estratégicos. Nesse contexto, surgem também iniciativas que ampliam o acesso ao desenvolvimento de aplicações, como o *NoCodeGPT*, uma interface de *no-code* que permite a criação de aplicações web a partir de modelos de linguagem, facilitando a atuação até mesmo de usuários com pouca experiência em programação (MONTEIRO et al., 2025).

Assim, o crescimento da importância do software exige que organizações e profissionais do setor inovem constantemente em suas práticas de desenvolvimento e manutenção. A capacidade de entregar soluções cada vez mais rápidas e alinhadas às demandas de um mundo digitalizado tornou-se um diferencial competitivo essencial. O futuro aponta para uma sociedade ainda mais interligada digitalmente, onde o desenvolvimento de softwares eficientes, ágeis e seguros será um dos principais pilares de sustentação (PENG et al., 2023; DAKHEL et al., 2023).

Em consonância com essa aceleração da necessidade de software, pode-se constatar o desenvolvimento de ferramentas voltadas ao apoio à programação (DAKHEL et al., 2023). Entre as recentes inovações estão as ferramentas de geração automática de código, que foram impulsionadas pelos avanços nos modelos de linguagem de grande escala (LLM, do inglês *Large Language Models*) (VASWANI et al., 2023). Um exemplo comercial dessas ferramentas é o GitHub Copilot, um assistente de programação baseado em inteligência artificial, criado pela OpenAI em parceria com o GitHub (CHEN et al., 2021). O Copilot é construído com base no Codex, um modelo de linguagem especializado para programação, e utiliza técnicas sofisticadas de processamento de linguagem natural para sugerir não apenas linhas de código, mas também blocos funcionais completos, ajustando-se ao contexto do código em que está sendo aplicado (GITHUB, 2021). Essa tecnologia tem o potencial de impactar positivamente o desempenho dos desenvolvedores, oferecendo sugestões que podem acelerar o fluxo de trabalho, otimizar o tempo de desenvolvimento e aumentar a eficiência geral das equipes.

A adoção de ferramentas de geração de código tem o potencial de impactar significativamente não só a produtividade de desenvolvedores experientes, mas também a formação de novos programadores, ao facilitar o aprendizado e aumentar a eficiência (LIRA; NETO; OSORIO, 2024).

Este estudo justifica-se pela necessidade de preencher uma lacuna na literatura acadêmica quanto à eficácia prática dessas ferramentas em contextos reais de desenvolvimento. Embora muitos estudos abordem suas capacidades técnicas, poucos avaliam empiricamente seu impacto direto no desempenho dos desenvolvedores (DAKHEL et al., 2023; MOON et al., 2023; XUE; NIU, 2023; PURYEAR; SPRINT, 2022). A presente pesquisa contribui para a área de Engenharia de Software ao fornecer evidências empíricas sobre os benefícios e limitações das ferramentas de geração de código, auxiliando desenvolvedores, gestores e educadores na tomada de decisões mais informadas sobre a adoção e o uso estratégico

dessas tecnologias. Ao explorar como o uso dessas ferramentas pode ser adaptado de acordo com o nível de experiência dos desenvolvedores, o estudo avança tanto o conhecimento acadêmico quanto a prática profissional, oferecendo *insights* relevantes para maximizar seus benefícios.

1.2 Problema de Pesquisa, Objetivos e Escopo

Com base na motivação apresentada, o problema de pesquisa desta dissertação consiste na criação de evidências empíricas sobre a eficácia das ferramentas de geração de código em cenários reais de desenvolvimento. Embora essas ferramentas sejam amplamente promovidas como recursos inovadores para aumentar o desempenho (PENG et al., 2023), há lacunas significativas na literatura no que se refere à sua aplicação prática. O problema central é a necessidade de validar se essas ferramentas realmente contribuem para a otimização do processo de programação, especialmente considerando desenvolvedores com diferentes níveis de experiência. Além disso, torna-se essencial identificar não apenas os benefícios, mas também as limitações dessas ferramentas, garantindo que seu uso seja embasado em resultados concretos e adequados ao contexto real de desenvolvimento de *software* (NGUYEN; NADI, 2022).

O objetivo geral deste estudo é avaliar se as ferramentas de geração de código efetivamente auxiliam os desenvolvedores a programar de forma mais eficaz. Para alcançar esse objetivo, os seguintes objetivos específicos foram definidos:

- Identificar, a partir da literatura disponível, os principais benefícios, limitações e desafios associados ao uso de ferramentas de geração de código em diferentes contextos de desenvolvimento de *software*.
- Identificar indicadores amplamente utilizados para avaliar a eficácia e o impacto de ferramentas de geração de código por IA no desenvolvimento de software.
- Desenvolver e conduzir estudos empíricos com desenvolvedores de diferentes níveis de experiência, visando avaliar o impacto dessas ferramentas no desempenho ao longo do processo de desenvolvimento de *software*.

Escopo do Estudo: Este estudo foca especificamente no uso do GitHub Copilot em cenários reais de desenvolvimento de *software*, com ênfase na linguagem de programação Java. O público-alvo inclui desenvolvedores de diferentes níveis de experiência, desde iniciantes até profissionais experientes. A pesquisa se concentra na avaliação do desempenho, excluindo outros aspectos como colaboração em equipe ou impacto na formação de conhecimento a longo prazo. O escopo também está restrito a um conjunto de tarefas de programação pré-definidas, sendo a análise qualitativa e quantitativa limitada ao uso

individual da ferramenta, sem considerar integrações com outras plataformas ou contextos de desenvolvimento.

1.3 Questões de Pesquisa

A definição de questões de pesquisa é um passo fundamental em qualquer estudo acadêmico, pois determina o foco da investigação e orienta o desenvolvimento da pesquisa. De acordo com Wohlin e Aurum (2015), a formulação adequada das questões permite que o estudo seja guiado para abordar problemas relevantes, além de possibilitar uma análise estruturada dos fenômenos observados. No contexto deste trabalho, as questões de pesquisa foram elaboradas para investigar o impacto das ferramentas de geração de código em cenários reais de desenvolvimento de *software*. Assim, foram definidas três questões que orientam este estudo:

QP1: Quais são as oportunidades e os desafios existentes na literatura sobre o uso de ferramentas de geração de código em cenários de desenvolvimento de *software*?

> *Justificativa*: Esta questão é essencial para identificar o que já foi explorado sobre o uso dessas ferramentas no desenvolvimento de *software*, bem como as lacunas que ainda precisam ser abordadas. A resposta a esta questão proporcionará uma base sólida para entender o contexto teórico e prático das ferramentas de geração de código.

QP2: Quais indicadores são comumente utilizados para avaliar a eficácia e o impacto de ferramentas de geração de código por IA no desenvolvimento de software?

> *Justificativa*: Essa questão visa identificar os principais indicadores adotados na literatura para avaliar ferramentas de geração de código por IA, o que permitirá definir critérios para mensurar sua eficácia e impacto no processo de desenvolvimento.

QP3: Como o uso de ferramentas de geração de código impacta o desempenho de desenvolvedores com diferentes níveis de experiência?

> *Justificativa*: Esta questão foca no objetivo central desta pesquisa, que é avaliar empiricamente a eficácia das ferramentas de geração de código. Através de estudos empíricos, será possível identificar de que maneira essas ferramentas influenciam o desempenho de desenvolvedores com diferentes níveis de experiência.

1.4 Estrutura do Trabalho

Este trabalho está organizada da seguinte forma: no Capítulo 2, é apresentado o referencial teórico, que descreve a evolução das ferramentas de geração de código, os indicadores usados para avaliar seu desempenho e os critérios para a construção de trabalhos empíricos. Em seguida, o Capítulo 3 discute os trabalhos relacionados, fornecendo uma visão abrangente das pesquisas já realizadas sobre o tema. O Capítulo 4 detalha o planejamento e a execução do estudo, descrevendo o desenho experimental, o estudo de caso e os métodos utilizados para coleta e análise de dados. No Capítulo 5, são apresentados os dados obtidos, com análises comparativas e estatísticas entre as condições com e sem o uso de ferramentas de geração de código. Por fim, o Capítulo 6 encerra o trabalho destacando as principais descobertas, as implicações práticas dos achados, as limitações do estudo e as sugestões para futuras pesquisas, incluindo o estudo de caso previsto em contexto corporativo.

2 Referencial Teórico

O presente capítulo apresenta os conceitos e formas de uso de ferramentas de geração de código, descrevendo a evolução das ferramentas de geração de código, indicadores usados para avaliar o desempenho destas e critérios para a construção de trabalhos empíricos.

2.1 Evolução das Ferramentas de Geração de Código

A evolução das ferramentas de geração de código reflete o progresso constante das técnicas de automação no desenvolvimento de *software*, buscando otimizar o processo de programação e reduzir o esforço humano em tarefas repetitivas. Desde as primeiras tentativas de automatizar a geração de código com ferramentas como compiladores e assistentes de codificação simples, até as atuais soluções baseadas em inteligência artificial, essas ferramentas passaram por várias gerações de inovações.

2.1.1 Ferramentas de Geração de Código

As primeiras ferramentas de geração de código surgiram no contexto do desenvolvimento dos compiladores, com o objetivo de traduzir linguagens de alto nível para código executável em linguagens de máquina. Pioneiras como o Lex e o Yacc, criadas nos anos 1970 (LEVINE; MASON; BROWN, 1992), representaram avanços fundamentais ao permitir a geração automática de parsers e analisadores sintáticos com base em especificações formais (ORTIN et al., 2022; LEVINE; MASON; BROWN, 1992). Essas ferramentas foram marcos significativos na evolução das linguagens de programação e estabeleceram uma base sólida para inovações subsequentes.

Com o passar dos anos, as necessidades dos desenvolvedores evoluíram, e as ferramentas de suporte à codificação passaram a incorporar funcionalidades mais sofisticadas para automatizar etapas do processo de programação. Em ambientes de desenvolvimento integrado (IDEs), por exemplo, funcionalidades de autocompletar se tornaram indispensáveis, facilitando a escrita de código por meio da sugestão automática de identificadores, como tipos, métodos e nomes de campos, com base em prefixos e contexto (HUSEIN; ABURAJOUH; CATAL, 2025; LEE et al., 2013). Um exemplo é o IntelliSense (MICROSOFT, 2024a), disponível no Visual Studio Code (MICROSOFT, 2024b), que vai além do autocompletar básico, oferecendo sugestões contextuais e inteligentes que melhoram a eficiência e a produtividade.

Além do autocompletar, as IDEs modernas passaram a incluir recursos avançados de navegação de código. Ferramentas para localização de declarações de elementos (como

a navegação até a declaração de um método a partir de uma chamada) e para exibir todas as referências a um elemento específico tornaram-se comuns, facilitando o entendimento e a modificação de códigos complexos. Algumas abordagens experimentais investigam o uso de informações dinâmicas do programa para aprimorar ainda mais a navegação e auxiliar na manutenção de *software* (HUSEIN; ABURAJOUH; CATAL, 2025; PRATHER et al., 2023). Essas inovações não só aceleram o desenvolvimento como também minimizam erros e promovem a qualidade e a consistência do código produzido.

2.1.2 A Era dos Modelos de Linguagem e a Integração de IA

Com o avanço dos modelos de linguagem de grande escala - LLMs, como o GPT-3 (BROWN et al., 2020) e, posteriormente, o Codex (CHEN et al., 2021), desenvolvidos pela OpenAI, houve uma transformação significativa nas ferramentas de geração de código. Essas ferramentas trouxeram a capacidade de interpretar o contexto do código em desenvolvimento e sugerir automaticamente trechos de código, funções completas ou até mesmo a resolução de problemas complexos (GITHUB, 2021).

Diferente das gerações anteriores, essas ferramentas de IA baseadas em LLMs aprendem a partir de vastos conjuntos de dados e são capazes de gerar código que não apenas segue padrões sintáticos, mas também leva em consideração boas práticas de programação e soluções previamente bem-sucedidas (LIRA; NETO; OSORIO, 2024; PENG et al., 2023; DAKHEL et al., 2023). Isso marcou um novo capítulo na automação do desenvolvimento de *software*, com uma ênfase na adaptação do código às necessidades imediatas do desenvolvedor, aumentando tanto a eficiência quanto a qualidade (PENG et al., 2023; DAKHEL et al., 2023).

2.1.3 Geração de Código via GitHub Copilot

O GitHub Copilot é uma ferramenta de geração de código assistido por inteligência artificial desenvolvida pela OpenAI em colaboração com o GitHub (GITHUB, 2021). Baseada inicialmente no modelo Codex, que por sua vez se originou do GPT-3 (CHEN et al., 2021) e, a partir de novembro de 2023, passou a utilizar o modelo GPT-4 (GitHub, 2023), essa ferramenta utiliza técnicas de aprendizado profundo para sugerir trechos de código e soluções completas, ajudando desenvolvedores a programar de forma mais eficiente. A base tecnológica do Copilot é fundamentada nos conceitos estabelecidos no trabalho de Vaswani et al. (2023), que introduziu o mecanismo de *attention* e as *transformers*, responsáveis por um salto significativo na capacidade dos modelos de linguagem de processar e gerar texto com base em contexto.

Diferente de abordagens tradicionais de autocompletar, que se limitam a sugestões baseadas em padrões simples, o Copilot é capaz de entender o contexto do código em

desenvolvimento, fornecendo sugestões que levam em conta não apenas a sintaxe, mas também o propósito funcional do código (NGUYEN; NADI, 2022). O modelo Codex, utilizado pelo Copilot, foi treinado em uma vasta quantidade de código-fonte disponível publicamente, o que lhe permite sugerir desde pequenas correções até métodos e classes completas em diversas linguagens de programação (CHEN et al., 2021).

De maneira geral, estudos empíricos têm demonstrado a eficácia do Copilot em distintos contextos de desenvolvimento de *software*. Diversos autores avaliam a ferramenta sob perspectivas variadas e com o uso de diferentes metodologias. Conforme será discutido no Capítulo 3, esses estudos indicam o potencial do Copilot para transformar a abordagem dos desenvolvedores em relação à programação, especialmente ao oferecer suporte em tarefas repetitivas e padronizadas.

2.2 Avaliação de Ferramentas de Geração de Código

As ferramentas de geração de código têm recebido atenção crescente na Engenharia de Software pela promessa de auxiliar desenvolvedores a aumentar a eficiência e a qualidade do código produzido (NGUYEN; NADI, 2022; BATISTA et al., 2024). Contudo, avaliar o impacto dessas ferramentas requer uma análise criteriosa de vários aspectos do código gerado, que vão desde a qualidade e segurança do código até a usabilidade e impacto sobre o desenvolvedor.

A literatura aponta uma série de indicadores que têm sido utilizados para avaliar diferentes dimensões das ferramentas de geração de código. Esses indicadores servem para verificar a eficácia e identificar os desafios associados ao uso dessas tecnologias. A seguir, são apresentados os principais indicadores identificados na literatura, categorizados conforme a área de avaliação.

2.2.1 Indicadores de Avaliação

Estes indicadores examinam diversos aspectos do código gerado pelas ferramentas de IA, incluindo qualidade, eficiência e usabilidade. Abaixo são explorados os principais indicadores, encontrados nos trabalhos avaliados:

2.2.1.1 Qualidade do Código

- **Validade do Código**: A capacidade do código de ser compilado sem erros é fundamental. O estudo de Corso et al. (2024) destaca a importância da validade do código como um requisito essencial para o uso prático de ferramentas de IA no desenvolvimento.

- **Correção do Código**: Para que o código seja funcional, é necessário que produza os resultados esperados. O trabalho de Yetiştiren et al. (2023) explora a correção do código gerado utilizando testes unitários e o *HumanEval Dataset* como referência.
- **Segurança do Código**: A segurança é essencial, e o código gerado não deve conter vulnerabilidades conhecidas. Estudos como de Dakhel et al. (2023) discutem a importância de manter a segurança no código gerado por IA. O estudo Nguyen e Nadi (2022) também menciona a análise de problemas de segurança em sugestões do Copilot.
- **Confiabilidade do Código**: A robustez do código em diferentes condições de uso é um aspecto crítico. Yetiştiren et al. (2023) apresentam a confiabilidade como um critério de qualidade fundamental para o código gerado.
- **Manutenibilidade do Código**: A facilidade de compreensão, modificação e manutenção do código ao longo do tempo é essencial. Indicadores como complexidade ciclomática e número de linhas de código são analisados em Dakhel et al. (2023).
- **Complexidade Ciclomática (McCabe)**: A complexidade impacta diretamente a legibilidade e a manutenção do código. Corso et al. (2024) discutem o uso da complexidade de McCabe para avaliar a qualidade do código gerado.
- **Code Smells**: "Code smells", como código duplicado ou métodos muito longos, são analisados por Yetiştiren et al. (2023) para identificar problemas potenciais na manutenibilidade do código.

2.2.1.2 Eficiência

- **Tempo de Conclusão da Tarefa**: O tempo que um desenvolvedor gasta para concluir uma tarefa usando a ferramenta é um indicador direto de eficiência, medido em Kazemitabaar et al. (2023) e Peng et al. (2023).
- **Tempo de Execução do Código**: O desempenho do código gerado, em termos de tempo de execução, é avaliado por Nejjar et al. (2024) para entender a eficiência do código gerado.
- **Número de Linhas de Código (LOC)**: O tamanho do código, medido em número de linhas, reflete sua concisão e complexidade. O estudo de Corso et al. (2024) investiga a relação entre LOC e eficiência do código.

2.2.1.3 Usabilidade

- **Taxa de Aceitação**: A proporção de sugestões aceitas pelos desenvolvedores mede a relevância das sugestões geradas.

- **Feedback do Usuário**: A coleta de feedback subjetivo permite compreender a experiência dos desenvolvedores e apontar oportunidades de melhoria. Estudos como Prather et al. (2023) e Davila et al. (2024) se concentram na percepção dos usuários.

- **Facilidade de Uso**: Uma interface intuitiva e funcional melhora a experiência com a ferramenta. O estudo de Yetiştiren et al. (2023) destaca a facilidade de uso como um critério importante para a adoção da ferramenta.

2.2.1.4 Outros Indicadores Adicionais

Além das métricas de qualidade, eficiência e usabilidade, a literatura sugere indicadores adicionais que complementam a análise:

- **Diversidade do Código**: A capacidade da ferramenta de gerar diferentes abordagens para uma mesma tarefa permite explorar soluções alternativas. Dakhel et al. (2023) investigam essa métrica comparando soluções geradas por IA e desenvolvedores humanos.

- **Compreensibilidade do Código**: Avalia a facilidade de compreensão do código gerado. Nejjar et al. (2024) discutem a importância de comentários no código e documentação.

- **Impacto no Aprendizado**: Em ambientes educacionais, é importante avaliar se a ferramenta facilita o aprendizado. O trabalho de (KAZEMITABAAR et al., 2023) analisa o impacto da IA no aprendizado de conceitos e técnicas de programação.

- **Pass@k**: Mede a precisão das k primeiras sugestões geradas. (DAKHEL et al., 2023) explora o uso desse indicador para avaliar a eficácia das sugestões.

- **Similaridade com o Código de Referência**: Compara o código gerado com implementações de referência, utilizando métricas como CodeBLEU e similaridade de Levenshtein, como observado em (CORSO et al., 2024).

- **Reprodutibilidade**: Avalia a capacidade da ferramenta de gerar soluções consistentes em diferentes tentativas, analisada em (DAKHEL et al., 2023).

- **Nível de Granularidade**: Refere-se à granularidade das sugestões, como tokens, linhas ou blocos de código, descrito em (PURYEAR; SPRINT, 2022).

- **Resposta Recebida**: Em (DAKHEL et al., 2023), essa métrica contabiliza o número de vezes que o Copilot gera código em resposta a um problema.

- **Optimalidade do Código**: Avalia se a ferramenta sugere algoritmos eficientes para um problema, como observado em (DAKHEL et al., 2023).

- **Auto-correção**: Analisa a capacidade das ferramentas de corrigirem seus próprios erros durante o processo de geração de código. Esse aspecto é explorado em (BATISTA et al., 2024), onde são comparadas as capacidades de auto-correção do Copilot, ChatGPT e Gemini.

2.3 Estudos Empíricos em Engenharia de *Software*

Os estudos empíricos em Engenharia de *Software* são fundamentais para a compreensão e avaliação de práticas, técnicas e ferramentas utilizadas no desenvolvimento de *software* (SHULL; SINGER; SJØBERG, 2008; WOHLIN; AURUM, 2015; ZHANG et al., 2018). Esses estudos buscam coletar dados e evidências a partir de situações do mundo real, permitindo a validação de teorias e modelos propostos na literatura (ZHANG et al., 2018). Por meio de abordagens empíricas, pesquisadores e profissionais conseguem avaliar o impacto de diversas variáveis no processo de desenvolvimento, como a produtividade dos desenvolvedores, a qualidade do código gerado e a eficácia das ferramentas utilizadas (SHULL; SINGER; SJØBERG, 2008).

De acordo com Wohlin e Aurum (2015), os estudos empíricos podem ser classificados em diferentes tipos: como estudos de caso, experimentos controlados, *surveys* e análises de métricas . Cada tipo de estudo oferece uma perspectiva única e contribui para a geração de conhecimento na área. Por exemplo, experimentos controlados permitem a manipulação de variáveis para observar seus efeitos diretos, enquanto *surveys* podem oferecer uma visão abrangente sobre práticas e opiniões de profissionais do setor (ZHANG et al., 2018).

A realização de estudos empíricos é especialmente relevante na era atual, onde as ferramentas de geração de código e as tecnologias de inteligência artificial estão se tornando cada vez mais comuns no desenvolvimento de *software* (SHULL; SINGER; SJØBERG, 2008). Ao investigar a eficácia dessas ferramentas em ambientes reais, os estudos empíricos fornecem dados valiosos que podem orientar decisões de adoção e implementação, além de contribuir para o aprimoramento das práticas de desenvolvimento.

Através da análise crítica de dados coletados, é possível identificar padrões, melhores práticas e áreas de melhoria, permitindo que as equipes de desenvolvimento otimizem seus processos e produtos (WOHLIN; AURUM, 2015; ZHANG et al., 2018). Portanto, a integração de abordagens empíricas no campo da Engenharia de *Software* é essencial para garantir que as inovações sejam não apenas teoricamente válidas, mas também práticas e eficazes em cenários reais (SHULL; SINGER; SJØBERG, 2008).

2.4 Considerações Finais

Neste capítulo, foi apresentada uma análise sobre as ferramentas de geração de código e sua relação com a produtividade no desenvolvimento de *software*. A evolução dessas ferramentas, desde as primeiras tentativas com compiladores até as soluções modernas baseadas em inteligência artificial, evidencia um processo contínuo em otimizar o processo de programação e reduzir a carga de trabalho manual dos desenvolvedores.

As funcionalidades de autocompletar, navegação facilitada e as capacidades inovadoras das ferramentas contemporâneas demonstram como as tecnologias têm se aprimorado para atender às necessidades do processo de desenvolvimento, promovendo maior eficiência e qualidade no código gerado.

Além disso, a análise da qualidade do código reforça a importância de características como legibilidade, manutenibilidade e robustez, que são essenciais para a sustentabilidade e evolução dos sistemas ao longo do tempo. A interação entre a qualidade do código e o uso de ferramentas de geração de código é um aspecto que merece atenção, uma vez que códigos de alta qualidade não apenas atendem aos requisitos funcionais, mas também facilitam a colaboração e a adaptação a novas demandas.

Neste contexto, foram apresentados os principais trabalhos ligados à avaliação de ferramentas de geração de código, em especial, o GitHub Copilot. O capítulo descreveu trabalhos relacionados e algumas métricas que podem ser usadas no estudo. Diante da variedade de indicadores de avaliação presentes na literatura, este estudo optou por focar em dois principais: **Tempo de Conclusão da Tarefa** e **Correção do Código**. Estes indicadores foram selecionados por sua relevância e aplicabilidade para medir o desempenho e a precisão do código gerado por ferramentas de IA em diferentes contextos de desenvolvimento. A seleção de indicadores apropriados permite uma análise mais focalizada dos efeitos do uso de ferramentas de geração de código sobre o processo de desenvolvimento e os resultados obtidos, fornecendo uma base sólida para avaliar sua utilidade e impacto no ambiente de desenvolvimento de software.

Por fim, os estudos empíricos em Engenharia de *Software* foram destacados como um componente importante para validar práticas e teorias, permitindo uma avaliação mais precisa do impacto das ferramentas e técnicas na produtividade e na qualidade do *software*. À medida que a tecnologia avança e novas metodologias surgem, a integração de abordagens empíricas se torna relevante para garantir que as inovações não apenas sejam teoricamente fundamentadas, mas também eficazes na prática.

3 Trabalhos Relacionados

Este capítulo apresenta os principais trabalhos relacionados ao uso de ferramentas de geração de código baseadas em IA, com foco no GitHub Copilot e modelos de linguagem de grande porte (LLMs). Os estudos analisados utilizam diferentes metodologias, incluindo experimentos controlados, estudos de caso, análises de código e *surveys*, com o objetivo de avaliar métricas como produtividade, correção, qualidade e usabilidade. Essa revisão oferece subsídios teóricos para a construção da presente pesquisa e fundamenta a escolha dos métodos e indicadores utilizados.

O estudo conduzido por Ngwenyama, Kanita e Rowe (2025) avaliou se a IA generativa contribui simultaneamente para ganhos de produtividade e bem-estar humano no ambiente de trabalho. A pesquisa foi realizada com desenvolvedores de uma grande empresa de serviços na França, ao longo de um período de 10 semanas, analisando o uso do GitHub Copilot com base em métodos estatísticos como CHAID e ANOVA. Os resultados indicaram que o Copilot aumentou a produtividade, reduziu a frustração e melhorou a percepção de bem-estar no trabalho. O bem-estar (*flourishing*) se destacou como o fator mais relevante para a satisfação dos desenvolvedores, influenciado positivamente pelo uso da ferramenta. O estudo destaca que os benefícios da IA dependem de como os gestores integram esses ganhos, sendo necessário evitar uma ênfase excessiva na produtividade que possa comprometer o bem-estar individual.

O trabalho de Bakal et al. (2025) apresentou uma avaliação sistemática da implementação do GitHub Copilot na empresa Zoominfo, adotando uma metodologia em quatro fases. Foram combinadas métricas quantitativas — como taxa de aceitação de sugestões (33%) e de linhas de código geradas (20%) — com análises qualitativas, por meio de questionários de satisfação. Os desenvolvedores relataram economia de tempo estimada em 20% e contribuíram com centenas de milhares de linhas de código de produção usando a ferramenta. Apesar dos ganhos, os autores destacam limitações como a ausência de lógica de domínio e a necessidade frequente de ajustes no código sugerido. O estudo conclui que o Copilot é particularmente útil para automatizar tarefas repetitivas, como testes unitários, e recomenda investigações futuras sobre o impacto a longo prazo na produtividade e nas habilidades dos desenvolvedores.

Já o estudo de Stray et al. (2025) investigou o uso de ferramentas de IA generativa, como GitHub Copilot e ChatGPT, em fluxos de trabalho reais de desenvolvimento. A pesquisa foi conduzida em uma organização ágil do setor de Fintech, por meio de um estudo de caso qualitativo com observações por 25 dias e 14 entrevistas. Observou-se que a IA foi utilizada em todas as sessões de programação solo e na maioria das sessões de

programação em par. Em atividades solo, os desenvolvedores relataram maior eficiência e redução de estresse, enquanto em sessões em par, a preferência pela interação humana persistiu em tarefas que exigem raciocínio compartilhado. Os participantes relataram uma economia média de 23 minutos por dia, embora tenham apontado limitações, como imprecisão nas respostas. O estudo conclui que o uso combinado da IA com a expertise humana pode otimizar o trabalho, especialmente quando se trata de tarefas rotineiras e de baixa complexidade.

O estudo conduzido por Corso et al. (2024), foi realizada uma avaliação de assistentes de código baseados em IA na geração de métodos Java. O estudo comparou o desempenho de quatro assistentes (Copilot, ChatGPT, Bard e Tabnine) na criação de implementações para métodos Java, utilizando um conjunto de dados com métodos extraídos do GitHub e acompanhados de comentários Javadoc para fornecer contexto. As implementações originais foram removidas, e os assistentes foram solicitados a gerar o código correspondente. A corretude sintática e semântica das soluções foi analisada, assim como a complexidade de McCabe, eficiência, tamanho do código e similaridade com o código original. Os resultados indicaram que o Copilot teve a maior taxa de sucesso (32%) na geração de métodos corretos, e o estudo sugere que a combinação de recomendações de múltiplos assistentes pode ser uma abordagem promissora.

Em um estudo de caso conduzido por Davila et al. (2024), foi investigada a adoção de assistentes de programação baseados em IA na indústria brasileira, com foco em uma empresa do setor agroindustrial. A pesquisa incluiu 72 funcionários da empresa, que responderam a um questionário sobre o uso e as percepções em relação a ferramentas de IA como o GitHub Copilot e o ChatGPT. Os autores concluíram que as ferramentas de IA são amplamente utilizadas para acelerar buscas, digitação e lembrança de sintaxes, embora uma dificuldade comum seja a falta de contexto nas sugestões geradas, levando os profissionais a elaborar descrições mais detalhadas para contornar essa limitação.

O estudo de Gu et al. (2024) examinou a eficácia dos modelos de linguagem de grande porte (LLMs) na geração de código para domínios específicos, utilizando dados de repositórios GitHub de desenvolvimento web e jogos. Foram utilizados diferentes tipos de prompts, como assinatura de função, importação de biblioteca, uso de APIs e docstrings, para avaliar o desempenho dos LLMs, incluindo o ChatGPT. A avaliação foi conduzida com as métricas BLEU e CodeBLEU, que medem a similaridade do código gerado em relação ao código de referência dos repositórios. Os resultados indicaram que os LLMs apresentam desempenho inferior na geração de código específico de domínio em comparação com código de domínio geral, mas a inclusão de prompts com conhecimento de domínio ajudou a melhorar a qualidade das soluções geradas.

Em Koyanagi et al. (2024), os autores analisaram o impacto das diferentes linguagens naturais (Inglês, Japonês e Chinês) nas sugestões de código do GitHub Copilot, buscando

entender como o idioma influencia a geração de código. Foram utilizadas 756 questões de diferentes níveis de dificuldade do AtCoder, todas em Python, para avaliar a precisão do código sugerido pelo Copilot em cada idioma. Sem treinamento prévio dos participantes na ferramenta ou na linguagem, o estudo revelou que o Copilot pode ser utilizado para gerar código a partir de diferentes linguagens naturais, com melhor desempenho para o inglês. Como desafios, o desempenho do Copilot foi considerado inconsistente, sendo o chinês o idioma com os piores resultados e observando-se uma diminuição na precisão das respostas para questões mais complexas. Isso indica que a ferramenta pode ter limitações em contextos multilíngues e em tarefas que exigem maior compreensão do problema.

Em Liu et al. (2024), os autores investigaram a eficácia do ChatGPT na geração de código, bem como os problemas de qualidade associados ao código gerado. A metodologia envolveu a coleta de 2.033 tarefas de programação do LeetCode e a análise do código gerado pelo ChatGPT em linguagens como Java e Python, utilizando ferramentas de análise estática, como PMD, Checkstyle, Pylint e Flake8. Este trabalho também não mencionou treinamento prévio dos participantes. As métricas de avaliação incluíram a taxa de aprovação em testes (pass@1) e a análise estática para identificar problemas de qualidade, como bugs e questões de estilo. Os autores concluíram que o ChatGPT pode ser eficaz em tarefas mais simples de programação, mas que a qualidade do código gerado diminui à medida que a complexidade das tarefas aumenta. O estudo identificou quatro categorias principais de problemas de qualidade de código: problemas de estilo, uso incorreto de APIs, manipulação inadequada de exceções e lógica. Embora o ChatGPT tenha demonstrado alguma capacidade de autocorreção, o estudo conclui que ainda há necessidade de aprimoramentos significativos para garantir a confiabilidade em tarefas mais complexas.

O estudo de Nejjar et al. (2024) investigou a eficácia de modelos de linguagem de grande porte (LLMs) na geração de código e análise de dados em contextos científicos específicos, selecionando três cenários: cálculo da média de uma lista de valores, análise de dados em Python e visualização de dados em R. Para isso, foram utilizados diferentes LLMs — incluindo ChatGPT, Bard, Bing Chat, YouChat e Copilot — com prompts específicos para cada tarefa, avaliando a capacidade de cada modelo em produzir código que atendesse às exigências científicas. A análise das soluções geradas considerou a corretude (verificação do funcionamento adequado do código), eficiência, qualidade da análise ou visualização e compreensibilidade do código (abrangendo clareza nos comentários, documentação e concisão). Os resultados mostraram uma variação significativa entre os LLMs: alguns modelos produziram código funcional e eficiente, enquanto outros falharam ao gerar soluções compiláveis ou adequadas. O estudo concluiu que, embora os LLMs tenham potencial para apoiar tarefas científicas, são necessárias avaliações críticas e aprimoramentos constantes para garantir maior confiabilidade em aplicações científicas complexas.

No trabalho de Kazemitabaar et al. (2023), foi avaliado o uso de LLMs para a geração de código e análise de dados em cenários científicos. O estudo abrangeu três cenários principais: geração de código para calcular a média de uma lista, análise de dados em Python e visualização de dados em R, utilizando diferentes LLMs, como ChatGPT, Bard, Bing Chat, YouChat e Copilot. Foram criados prompts específicos para cada tarefa, e o desempenho dos LLMs foi avaliado em termos de corretude, eficiência, qualidade da análise ou visualização e compreensibilidade (incluindo clareza nos comentários e concisão do código). Os resultados mostraram que o desempenho dos LLMs variou consideravelmente entre ferramentas e cenários, destacando tanto o potencial quanto as limitações dessas ferramentas em aplicações científicas.

Dakhel et al. (2023) avaliaram a eficácia do GitHub Copilot como assistente de programação, investigando se a ferramenta é um recurso valioso ou uma potencial limitação para desenvolvedores. A avaliação foi baseada na similaridade de Árvores Sintáticas Abstratas (AST) e incluiu métricas como número de tentativas, corretude, similaridade, custos de reparo, diversidade, complexidade ciclomática e domínio sintático. O estudo, que não envolveu treinamento prévio dos participantes, concluiu que o Copilot pode ser uma ferramenta útil para os programadores, mas a qualidade do código gerado varia consideravelmente.

O trabalho de Peng et al. (2023) avaliou o impacto do GitHub Copilot na produtividade dos desenvolvedores. Para isso, desenvolvedores foram divididos em grupos de controle e tratamento, e cada grupo realizou tarefas padronizadas em um ambiente controlado. A produtividade foi medida pela porcentagem de sucesso na conclusão das tarefas e pelo tempo necessário para completá-las. Os resultados mostraram um aumento de 55% na produtividade dos desenvolvedores que utilizaram o Copilot, indicando que o uso da ferramenta pode acelerar significativamente o ritmo de desenvolvimento de software.

Em Prather et al. (2023), o foco foi a usabilidade e as interações de programadores novatos com o GitHub Copilot. A pesquisa explorou como jovens entre 10 e 17 anos interagiram com o Copilot em tarefas de programação em C++. Sem treinamento prévio, os participantes realizaram tarefas enquanto suas interações com a ferramenta foram observadas, incluindo o tempo dedicado a ler e depurar o código sugerido e feedback qualitativo sobre a experiência. Os resultados sugerem que o Copilot pode ser útil para novatos, mas que os estudantes enfrentam desafios para entender e depurar o código sugerido, indicando a necessidade de suporte adicional, os autores, ainda, concluem que a familiaridade com a ferramenta e o desenvolvimento de habilidades em programação são fatores essenciais para seu uso eficaz.

Yetiştiren et al. (2023) conduziram um estudo que compara a qualidade do código gerado por três assistentes de código — GitHub Copilot, Amazon CodeWhisperer e ChatGPT. Utilizando o conjunto de dados HumanEval, problemas de programação foram

extraídos e transformados em testes unitários e prompts em Python. As soluções foram geradas no Visual Studio Code (para Copilot e CodeWhisperer) e na interface web do ChatGPT. A qualidade das soluções foi avaliada com o uso do SonarQube, que verificou segurança, bugs, code smells, confiabilidade e manutenibilidade. Observou-se que o ChatGPT teve a maior taxa de sucesso (65,2%) na geração de código correto, enquanto a clareza e precisão dos enunciados dos problemas impactaram diretamente a qualidade do código gerado. Além disso, identificou-se que tanto o CodeWhisperer quanto o Copilot passam por aprimoramentos contínuos.

Wermelinger (2023) analisou a capacidade do GitHub Copilot para resolver problemas simples de programação em Python. A metodologia envolveu a geração de código pelo Copilot para problemas de níveis variados de dificuldade. Sem treinamento prévio, o estudo utilizou como métricas a corretude do código gerado e o tempo para concluir as tarefas. As conclusões sugerem que o Copilot pode ser uma ferramenta útil para iniciantes em programação, mas os autores ressaltaram a importância de avaliar o impacto da ferramenta na formação e nas habilidades de desenvolvimento dos estudantes, destacando um ponto de cautela em seu uso educativo.

Em outro estudo, Imai (2022) investou se o GitHub Copilot poderia substituir a programação em pares humana. Para isso, foi comparado o número de linhas de código adicionadas e removidas em três condições de desenvolvimento: programação individual, programação em pares humanos e programação com o Copilot. A produtividade foi medida pelo número de linhas de código adicionadas, enquanto a qualidade do código foi avaliada pelo número de linhas removidas nas tentativas subsequentes. Os resultados mostram que o Copilot pode aumentar a produtividade do desenvolvedor, embora a qualidade do código gerado ainda seja inferior em comparação com a colaboração humana em pares.

O trabalho de Nguyen e Nadi (2022) avaliaram empiricamente as sugestões de código do GitHub Copilot em contextos variados de consulta, utilizando 33 questões do LeetCode, com diferentes níveis de dificuldade. Foram extraídas manualmente informações como nome da função, parâmetros, entradas e saídas, além de comentários, para criar contextos ideais para consulta. A partir disso, um script personalizado gerou arquivos de código separados em quatro linguagens — Python, Java, JavaScript e C — resultando em 132 arquivos. A análise das soluções fornecidas pelo Copilot foi feita por meio da complexidade cognitiva e ciclomática, cujas medianas foram 6 e 5, respectivamente. Embora os resultados não indiquem diferenças significativas entre as linguagens, limitações do estudo incluem o número restrito de consultas e o sistema fechado do Copilot.

Tabela 1 – Comparação de Trabalhos Relacionados sobre Ferramentas de Geração de Código Assistido por IA.

Ordem	Referência	Ferramenta(s) Avaliada(s)	Linguagem	Contexto do Estudo	Nível dos Participantes	Principais Resultados
1	Ngwenyama, Kanita e Rowe (2025)	Copilot	Não especificada	Real (empresa)	Profissionais	Aumento da produtividade e do bem-estar; redução da frustração; satisfação no trabalho impactada positivamente.
2	Bakal et al. (2025)	Copilot	Não especificada	Real (empresa)	Profissionais	Taxa de aceitação de 33%; economia de tempo (20%); útil para tarefas repetitivas; limitações em lógica de domínio.
3	Stray et al. (2025)	Copilot, ChatGPT	Não especificada	Real (empresa)	Profissionais	Aumenta eficiência em programação solo; menos efetivo em par; economia média de 23 min/dia.

Continua na próxima página...

Ordem	Referência	Ferramenta(s) Avaliada(s)	Linguagem	Contexto do Estudo	Nível dos Participantes	Principais Resultados
4	Corso et al. (2024)	Copilot, ChatGPT, Bard, Tabnine	Java	Controlado (dataset)	N/A (avaliação de ferramentas)	Resultados similares ao anterior; Copilot com 32% de sucesso.
5	Davila et al. (2024)	Copilot, ChatGPT	Não especificada	Real (empresa)	Profissionais	Dificuldade de contexto em sugestões; descrições detalhadas ajudam.
6	Gu et al. (2024)	ChatGPT	Python, JavaScript	Controlado (dataset)	N/A (avaliação de ferramentas)	LLMs têm desempenho inferior em domínios específicos.
7	Koyanagi et al. (2024)	Copilot	Python	Controlado (dataset)	N/A (avaliação de ferramenta)	Copilot varia conforme idioma; melhor desempenho em inglês.
8	Liu et al. (2024)	ChatGPT	Java, Python	Controlado (dataset)	N/A (avaliação de ferramenta)	Efetivo em tarefas simples; problemas de qualidade em tarefas complexas.

Continua na próxima página...

Ordem	Referência	Ferramenta(s) Avaliada(s)	Linguagem	Contexto do Estudo	Nível dos Participantes	Principais Resultados
9	Nejjar et al. (2024)	Copilot, ChatGPT, Bard, Bing Chat, YouChat	Python, R	Controlado (cenários)	N/A (avaliação de ferramentas)	LLMs funcionam bem com prompts específicos.
10	Kazemitabaar et al. (2023)	Copilot, ChatGPT, Bard, Bing Chat, YouChat	Python, R	Controlado (cenários)	N/A (avaliação de ferramentas)	Desempenho variável entre ferramentas e cenários.
11	Dakhel et al. (2023)	Copilot	Não especificada	Controlado (métricas)	N/A (avaliação de ferramenta)	Ferramenta útil, mas a qualidade do código varia.
12	Peng et al. (2023)	Copilot	Não especificada	Controlado (experimento)	Desenvolvedores	Produtividade aumentou em 55% com o Copilot.
13	Prather et al. (2023)	Copilot	C++	Controlado (estudo c/ usuários)	Novatos / Estudantes	Novatos enfrentam desafios para entender e depurar sugestões.

Continua na próxima página...

Ordem	Referência	Ferramenta(s) Avaliada(s)	Linguagem	Contexto do Estudo	Nível dos Participantes	Principais Resultados
14	Yetiştiren et al. (2023)	Copilot, CodeWhisperer, ChatGPT	Python	Controlado (dataset)	N/A (avaliação de ferramentas)	ChatGPT teve maior sucesso (65,2%) em código correto.
1al (empresa)	Profissionais	Dificuldade de contexto em sugestões; descrições detalhadas ajudam.				
6	Gu et al. (2024)	ChatGPT	Python, JavaScript	Controlado (dataset)	N/A (avaliação de ferramentas)	LLMs têm desempenho inferior em domínios específicos.
7	Koyanagi et al. (2024)	Copilot	Python	Controlado (dataset)	N/A (avaliação de ferramenta)	Copilot varia conforme idioma; melhor desempenho em inglês.
8	Liu et al. (2024)	ChatGPT	Java, Python	Controlado (dataset)	N/A (avaliação de ferramenta)	Efetivo em tarefas simples; problemas de qualidade em tarefas complexas.

Continua na próxima página...

Ordem	Referência	Ferramenta(s) Avaliada(s)	Linguagem	Contexto do Estudo	Nível dos Participantes	Principais Resultados
9	Nejjar et al. (2024)	Copilot, ChatGPT, Bard, Bing Chat, YouChat	Python, R	Controlado (cenários)	N/A (avaliação de ferramentas)	LLMs funcionam bem com prompts específicos.
10	Kazemitabaar et al. (2023)	Copilot, ChatGPT, Bard, Bing Chat, YouChat	Python, R	Controlado (cenários)	N/A (avaliação de ferramentas)	Desempenho variável entre ferramentas e cenários.
11	Dakhel et al. (2023)	Copilot	Não especificada	Controlado (métricas)	N/A (avaliação de ferramenta)	Ferramenta útil, mas a qualidade do código varia.
12	Peng et al. (2023)	Copilot	Não especificada	Controlado (experimento)	Desenvolvedores	Produtividade aumentou em 55% com o Copilot.
13	Prather et al. (2023)	Copilot	C++	Controlado (estudo c/ usuários)	Novatos / Estudantes	Novatos enfrentam desafios para entender e depurar sugestões.

Continua na próxima página...

Ordem	Referência	Ferramenta(s) Avaliada(s)	Linguagem	Contexto do Estudo	Nível dos Participantes	Principais Resultados
14	Yetiştiren et al. (2023)	Copilot, CodeWhisperer, ChatGPT	Python	Controlado (dataset)	N/A (avaliação de ferramentas)	ChatGPT teve maior sucesso (65,2%) em código correto.
15	Wermelinger (2023)	Copilot	Python	Controlado (dataset)	N/A (avaliação de ferramenta)	Útil para iniciantes em tarefas simples.
16	Imai (2022)	Copilot	Não especificada	Controlado (experimento)	Não especificado	Aumenta produtividade, mas colaboração humana ainda é melhor.
17	Nguyen e Nadi (2022)	Copilot	Python, Java, JavaScript, C	Controlado (dataset)	N/A (avaliação de ferramenta)	Eficaz, mas precisa de melhorias em consultas complexas.

3.1 Considerações Finais

A análise dos trabalhos relacionados revela uma diversidade de abordagens na avaliação do impacto das ferramentas de geração de código assistido por IA, como o GitHub Copilot, ChatGPT e outros LLMs. Os estudos revisados abrangem desde investigações sobre a correção e qualidade do código gerado (NGUYEN; NADI, 2022; YETIŞTIREN et al., 2023) até análises mais amplas sobre produtividade, usabilidade e percepção dos desenvolvedores (PRATHER et al., 2023; DAVILA et al., 2024).

Uma característica comum entre muitos desses trabalhos é o foco na comparação entre soluções produzidas pelas ferramentas de IA e aquelas desenvolvidas por programadores humanos (DAKHEL et al., 2023; CORSO et al., 2024). Os estudos indicam que, embora as ferramentas sejam capazes de acelerar o desenvolvimento em tarefas rotineiras, ainda enfrentam dificuldades em garantir a qualidade do código em cenários mais complexos, além de demandarem revisão manual para ajustes finos.

Nos estudos mais recentes, de 2025, observa-se uma ampliação do escopo de análise, com ênfase em aspectos subjetivos como o bem-estar e a satisfação dos desenvolvedores (NGWENYAMA; KANITA; ROWE, 2025; BAKAL et al., 2025; STRAY et al., 2025). Esses trabalhos destacam que o uso do Copilot, além de aumentar a produtividade, pode reduzir a frustração e melhorar a percepção de bem-estar no trabalho, desde que a gestão das equipes evite uma ênfase excessiva em métricas puramente produtivas.

Outros estudos aprofundaram as análises sobre a eficácia dos assistentes de código na geração de métodos Java (CORSO et al., 2024) e no uso das ferramentas em contextos industriais brasileiros (DAVILA et al., 2024), reforçando que a qualidade das sugestões depende fortemente do contexto fornecido (como descrições e docstrings) e da familiaridade dos usuários com a ferramenta. Essas conclusões complementam trabalhos anteriores, que já apontavam a importância de prompts bem elaborados para melhorar a eficácia das sugestões (YETIŞTIREN et al., 2023).

Além disso, a questão da interação de programadores novatos com essas ferramentas continua sendo uma preocupação. Estudos como os de Prather et al. (2023) e Kazemitabaar et al. (2023) destacam que, embora a IA possa ser uma aliada no processo de aprendizagem, sua utilização sem acompanhamento pedagógico pode resultar em dependência e comprometer o desenvolvimento de habilidades fundamentais de programação.

No entanto, uma limitação comum entre muitos desses estudos, incluindo os mais recentes, é o foco em cenários controlados ou em tarefas pontuais, como a resolução de problemas do LeetCode ou a geração de métodos isolados, sem considerar o impacto das ferramentas em fluxos de trabalho mais complexos e realistas.

Diante disso, o presente trabalho propõe uma abordagem empírica estruturada e mais alinhada ao cotidiano profissional, simulando cenários reais de manutenção de

software, onde os participantes devem recriar métodos a partir de descrições funcionais, elevando o nível de complexidade da tarefa. Além disso, diferencia-se ao integrar um estudo de caso real em uma empresa de desenvolvimento de software, permitindo avaliar o impacto do uso do GitHub Copilot em fluxos de trabalho reais, considerando fatores como adaptação da equipe, percepção de utilidade e mudanças nos processos de codificação.

Outro diferencial importante desta pesquisa é a inclusão de uma etapa formal de treinamento dos participantes, tanto no uso da ferramenta quanto na compreensão da estrutura dos sistemas envolvidos. Essa preparação visa reduzir vieses relacionados à falta de familiaridade, garantindo que os resultados reflitam de maneira mais precisa o impacto efetivo da ferramenta sobre o desempenho dos desenvolvedores.

Em suma, enquanto os estudos revisados oferecem contribuições para a compreensão do potencial e das limitações das ferramentas de geração de código, permanece evidente a necessidade de pesquisas que explorem contextos mais complexos e realistas. Este trabalho busca preencher essa lacuna, oferecendo uma análise empírica prática e aplicada, capaz de subsidiar decisões mais informadas sobre a adoção dessas tecnologias em ambientes produtivos e desafiadores do mercado de trabalho.

4 Metodologia

Este capítulo descreve os métodos empregados para conduzir o presente estudo, cujo objetivo principal é avaliar o impacto das ferramentas de geração de código assistido por IA no desenvolvimento de software. A metodologia de pesquisa pode ser definida como o conjunto de métodos, técnicas e procedimentos organizados e sistemáticos que orientam a condução de um estudo científico, permitindo alcançar os objetivos propostos e responder às questões de pesquisa (seção 1.3). Neste trabalho, a metodologia consiste em quatro fases principais: *Investigação*, *Coleta de Dados*, *Análise* e *Resultados*. Essas fases foram definidas com base no método de *Technical Action Research* (TAR).

O TAR combina aspectos de *Action Research* e *Design Science*, conforme descrito por (WIERINGA; MORALI, 2012), e é amplamente utilizado em estudos que envolvem a avaliação de tecnologias inovadoras. Sua principal característica é permitir a solução de problemas práticos enquanto gera conhecimento científico. A escolha pelo TAR se justifica pela necessidade de investigar o impacto de ferramentas de geração de código assistido por IA, permitindo integrar a avaliação experimental – necessária para validar os indicadores selecionados – com intervenções controladas em contextos reais de desenvolvimento de software. Dessa forma, o TAR facilita a conexão entre os resultados empíricos e suas implicações práticas e teóricas.

A Figura 1 ilustra o fluxo das atividades realizadas em cada fase. As fases da metodologia foram concebidas de forma integrada, com cada etapa desempenhando um papel específico para atingir os objetivos da pesquisa. A fase de *Investigação* estabelece as bases teóricas, identificando indicadores amplamente utilizados e categorizando os principais desafios e benefícios associados ao uso de ferramentas de geração de código assistido por IA. Essas descobertas fundamentam a fase de *Coleta de Dados*, na qual são realizados estudos empíricos para avaliar o impacto dessas ferramentas no desenvolvimento de software. Em seguida, a fase de *Análise* utiliza os dados coletados para validar os indicadores selecionados e gerar insights quantitativos e qualitativos. Por fim, os *Resultados* sintetizam essas descobertas, permitindo responder às questões de pesquisa e atingir os objetivos propostos.

4.1 Investigação

A fase de investigação teve como objetivo central realizar um levantamento bibliográfico para embasar as questões de pesquisa **QP1** e **QP2**. Esse levantamento da literatura focou em identificar as oportunidades, desafios e principais indicadores usados para avaliar ferramentas de geração de código. Este processo inicial é fundamental para estabelecer

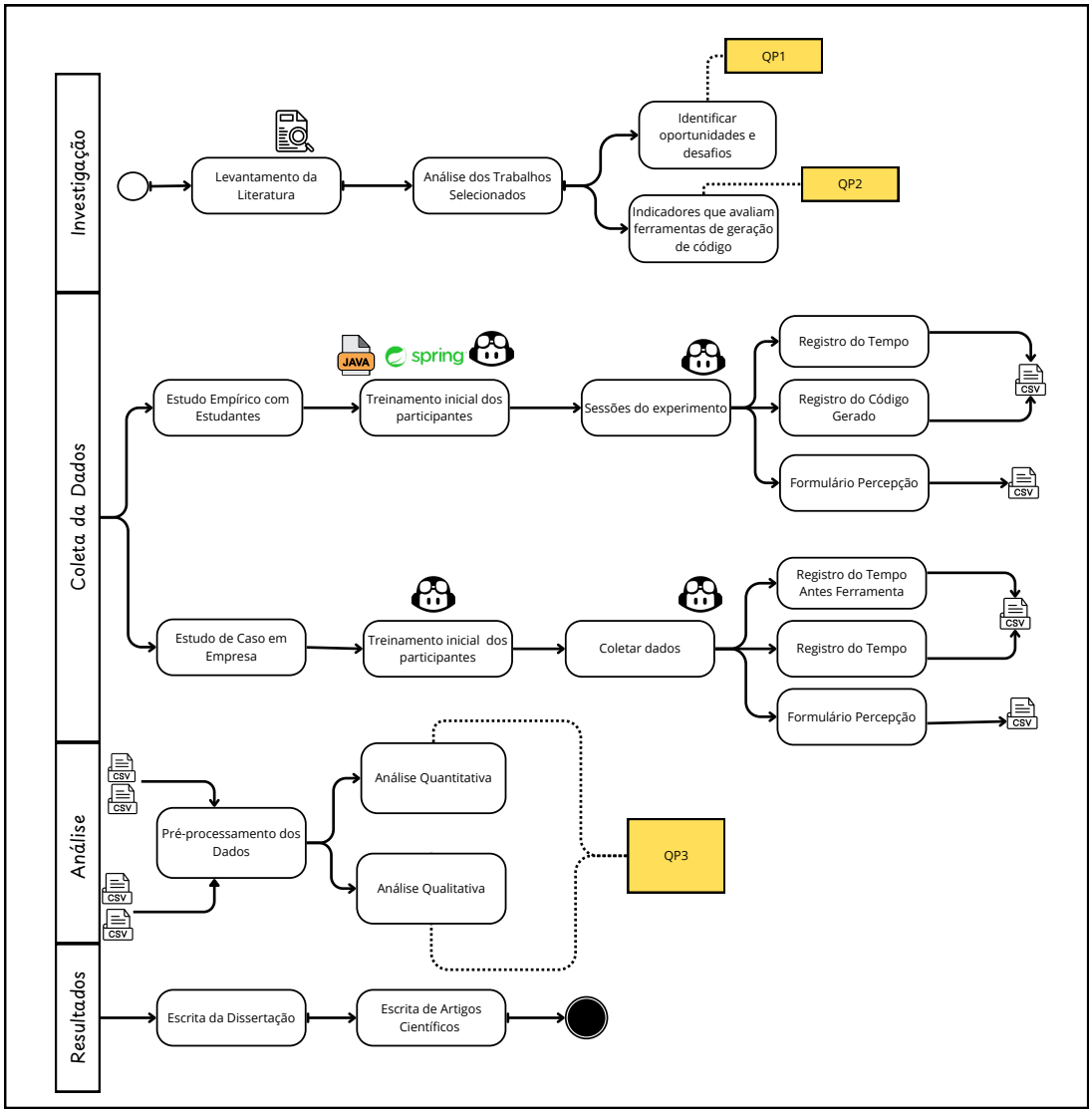


Figura 1 – Fluxo de atividades na metodologia do estudo.
Fonte: autor.

as bases teóricas e metodológicas do estudo, permitindo a seleção de indicadores que seriam aplicados nas fases subsequentes para avaliar o desempenho e a experiência dos desenvolvedores ao utilizar essas ferramentas.

Inicialmente, foram selecionados artigos relevantes que abordam o uso de ferramentas de geração de código assistido por IA, utilizando o *Google Scholar* como principal plataforma de busca, devido à sua ampla cobertura de bases de dados acadêmicas e facilidade de acesso. Como critério adicional, o autor optou por incluir apenas trabalhos que possuíam, no momento da pesquisa, pelo menos três citações, considerando que esse número mínimo poderia indicar um nível inicial de reconhecimento ou relevância no meio acadêmico. Essa escolha foi uma decisão livre do autor, sem pretensão de exaurir a literatura, mas com o objetivo de compor um conjunto de estudos que já tivessem algum impacto mensurável na comunidade científica.

Esses estudos foram analisados com o objetivo de identificar indicadores e métricas amplamente utilizados para avaliar a eficiência, usabilidade e eficácia dessas ferramentas. Essa análise orientou a definição dos parâmetros de avaliação aplicados na fase de coleta de dados. Os indicadores identificados foram, posteriormente, categorizados conforme descrito na seção 2.2.

4.2 Coleta de Dados

A fase de coleta de dados foi dividida em duas partes: um **Estudo Empírico com Estudantes** e um **Estudo de Caso em Empresa**. Essas abordagens complementares permitiram avaliar o uso da ferramenta de geração de código em diferentes contextos e perfis de desenvolvedores, proporcionando uma visão ampla sobre o impacto da ferramenta no desenvolvimento de software. A coleta de dados integrada dessas duas abordagens permitirá avaliar a ferramenta em cenários diversos, fornecendo evidências empíricas para responder às questões de pesquisa e alcançar os objetivos deste trabalho.

4.2.1 Estudo Empírico com Estudantes

O estudo empírico com estudantes teve como objetivo avaliar o desempenho de participantes novatos em programação ao utilizar o Copilot para implementar métodos que foram apagados de um sistema. Essa abordagem permitiu investigar como a ferramenta pode impactar a eficiência e a qualidade do código gerado por desenvolvedores com menor experiência.

O Copilot foi escolhido por ser uma ferramenta amplamente investigada por outros autores, além de estar disponível gratuitamente para estudantes por meio do programa *GitHub Education* (GITHUB, 2024). Essa escolha também se justifica pela facilidade de integração do Copilot ao ambiente de desenvolvimento.

Os 49 participantes do estudo empírico foram estudantes de Computação, nos níveis de graduação e pós-graduação, com diferentes níveis de conhecimento. Todos os alunos das disciplinas Engenharia de Software (21 participantes) e Programação Orientada a Objetos (17 participantes)(Graduação) e Engenharia de Software (Pós-Graduação) (11 participantes) foram convidados a participar, sem realização de sorteio.

Para garantir um ambiente padronizado para todos os participantes, foi utilizado o Visual Studio Code (VSCode) na versão 1.90.1, juntamente com a extensão oficial do GitHub Copilot[1], versão 0.12. Diferentemente da configuração padrão, a funcionalidade de *auto complete (Enable Auto Completions)* foi desativada, garantindo que os participantes utilizassem exclusivamente a funcionalidade de chat do Copilot. Essa restrição foi imposta

[1] https://marketplace.visualstudio.com/items?itemName=GitHub.copilot

para isolar o impacto da assistência via chat no processo de desenvolvimento, evitando interferências de sugestões automáticas durante a implementação das funcionalidades ausentes.

A etapa de Estudo Empírico com Estudantes foi dividida nas seguintes subetapas: Treinamento inicial dos participantes, sessões de experimento e registros (do tempo, código gerado e formulário de percepção). Esse conjunto de passos produzirá a base de dados de análise da ferramenta de geração de código. A seguir, cada subetapa é descrita.

Considerações Éticas: Todos os participantes foram informados sobre os objetivos da pesquisa e participaram de forma voluntária, após consentimento informado. Os dados coletados foram tratados de maneira confidencial e anonimizada, assegurando a privacidade dos envolvidos.

4.2.1.1 Treinamento Inicial dos Participantes

A fim de garantir que todos os participantes estivessem no mesmo nível de conhecimento, foram ministradas oficinas, como forma de treinamento inicial.

Foram realizados 04 encontros de forma remota, por meio da plataforma Google Meet[2] e 02 encontros presenciais. Em cada encontro, foram trabalhados os seguintes conteúdos:

- **Encontro 01**: Apresentação do framework Spring boot, Configuração do ambiente de desenvolvimento (Vscode, Maven, Java, ...), padrão arquitetural Controller-Service-Repository[3], apresentação da ferramenta Copilot e processo para solicitar a licença estudantil por meio do programa *GitHub Education* (GITHUB, 2024).
- **Encontro 02**: Validações, exceções e JPAQuery.
- **Encontro 03**: Geração de código por meio do Copilot e estrutura da sessão experimental. Neste encontro os participantes tiveram que recriar 1 método apagado do sistema 1 usando a ferramenta.
- **Encontro 04**: Encontro presencial para tirar dúvidas sobre os conteúdos trabalhados de forma remota.
- **Encontro 05**: Encontro presencial para tirar dúvidas sobre os conteúdos trabalhados de forma remota.
- **Encontro 06**: Piloto do experimento, neste encontro os participantes foram direcionados a recriar 2 métodos que foram apagados de dois sistema, um método deveria ser recriado usando o Copilot e outro sem o uso da ferramenta.

[2] <https://meet.google.com/>
[3] <https://en.wikipedia.org/wiki/Software_design_pattern>

4.2.1.2 Sessões do Experimento

O estudo empírico consistiu em uma sessão experimental, na qual os participantes utilizaram ou não a ferramenta de geração de código para recriar métodos que foram excluídos dos sistemas fornecidos. Durante o experimento, os participantes realizaram as atividades sob supervisão, sendo coletados dados sobre o tempo de execução das tarefas, o código gerado, a interação com a ferramenta Copilot – incluindo os *prompts* utilizados – e as percepções sobre o uso da ferramenta, registradas em um formulário de feedback ao final. Os participantes também foram orientados a gravar a tela durante a resolução dos problemas, permitindo uma análise detalhada de suas interações e estratégias adotadas ao longo das tarefas.

Durante a sessão experimental, foram disponibilizados aos estudantes dois sistemas desenvolvidos em Java/Spring Boot (nomeados como locadora-api, disponível em Osorio (2024b) e associacao-api, disponível em Osorio (2024a)). Em cada sistema, foram suprimidos dois métodos, gerando os problemas identificados como #1, #2, #3 e #4, conforme Tabela 2. O Apêndice C apresenta os códigos originais que foram apagados e os quais os participantes do experimento precisariam reescrever.

Tabela 2 – Associação entre o ID do problema e o método removido nos sistemas.

ID	**Método Removido**	**Sistema**
#1	VeiculoService.registrarAluguel(placa, data, dias, cpf)	locadora-api
#2	VeiculoService.aumentarDiaria(int tipo, double taxaAumento)	
#3	TaxaService.calcularTotalDeTaxas(numAssociacao, vigencia)	associacao-api
#4	PagamentoService.registrarPagamento(numAssociacao, taxa, vigencia, numAssociado, data, valor)	

Com o objetivo de manter a proporcionalidade entre as abordagens com e sem o uso da ferramenta, os problemas foram distribuídos aleatoriamente entre os estudantes e eles deveriam recriar o método de acordo com essa distribuição (exemplo de distribuição dos problemas por participantes, vide Tabela 3), de modo que cada problema tivesse uma quantidade equilibrada de resoluções com e sem o Copilot, conforme os dados apresentados na Tabela 4.

Os participantes foram orientados a recriar apenas os métodos suprimidos, sem adicionar novos métodos, pois apenas esses métodos haviam sido excluídos, mantendo-se os métodos auxiliares já presentes para garantir que o comportamento fosse conforme o esperado. Embora os testes unitários não estivessem disponíveis, os participantes puderam verificar o resultado das suas implementações utilizando dados presentes na base de dados gerada pela própria API ao iniciar. Essa verificação foi feita por meio de requisições à API

Tabela 3 – Exemplo de distribuição dos problemas por participantes.

ID Participante	ID Problema	Usar Copilot?
1	#1	Sim
	#2	Não
	#3	Não
	#4	Sim
2	#1	Não
	#2	Sim
	#3	Sim
	#4	Não
3	#1	Sim
	#2	Não
	#3	Não
	#4	Sim

usando o Swagger[4].

Para a coleta dos resultados, os estudantes foram orientados a submeter o Questionário de Solução Desenvolvida ao final de cada problema resolvido (Apêndice A). Após a conclusão dos quatro problemas, cada participante foi instruído a responder ao Questionário Final, que inclui questões demográficas e de percepção sobre o uso da ferramenta (Apêndice B).

4.2.1.3 Dados Coletados

Os dados coletados durante o estudo empírico incluíram:

- **Registro do Tempo de Execução**: Os participantes foram instruídos a registrar manualmente o horário de início e término de cada tarefa, tanto nas condições com o uso da ferramenta de geração de código quanto sem seu auxílio. Esse registro permitiu uma comparação direta do tempo necessário para completar as atividades em ambos os cenários, fornecendo dados para análise do impacto da ferramenta no tempo de execução.

- **Registro do Código Gerado**: O código produzido por cada participante foi coletado e submetido a testes unitários para avaliar sua correção. Esse processo foi utilizado exclusivamente para verificar se o código gerado ou criado pelo participante atendia aos critérios de funcionalidade esperados.

- **Feedback Qualitativo**: Após cada sessão, os participantes responderam a um formulário de feedback, fornecendo informações sobre a experiência de uso, desafios enfrentados e percepção sobre a utilidade da ferramenta.

[4] <https://swagger.io/>

Esses dados serviram de base para a análise quantitativa e qualitativa, com o objetivo de avaliar o impacto da ferramenta em termos de eficiência e experiência dos participantes.

4.2.2 Estudo de Caso

Com o objetivo de ampliar a análise do impacto do GitHub Copilot em ambientes reais de desenvolvimento de software, foi conduzido um estudo de caso na Superintendência de Tecnologia da Informação (STI) da Universidade Federal do Piauí (UFPI). A STI é responsável por diversos sistemas corporativos da instituição, incluindo o conjunto de sistemas integrados de gestão (SIG), utilizados por setores acadêmicos e administrativos da universidade.

Nesta etapa do estudo, participaram quatro desenvolvedores que atuam diretamente na manutenção e evolução dos sistemas SIG. O quinto integrante do time foi excluído da amostra por ser recém-integrado à equipe, não possuindo histórico de atividades no período anterior à introdução do Copilot, necessário para fins comparativos.

Diferentemente do estudo empírico com estudantes, que contou com tarefas específicas e controladas, os participantes deste estudo de caso seguiram com suas atividades rotineiras, sem interferência externa na definição ou execução das tarefas. O objetivo foi avaliar o impacto da ferramenta em um cenário realista, minimizando alterações no fluxo de trabalho natural da equipe.

Para permitir o uso do GitHub Copilot no ambiente de desenvolvimento utilizado pela equipe, foi instalada a extensão Copilot4Eclipse (versão 1.3.0)[5], compatível com a versão 2023-06 (4.28) do Eclipse[6]. Este procedimento exigiu que os desenvolvedores atualizassem suas versões do Eclipse, uma vez que todos utilizavam versões anteriores não compatíveis com a extensão. Após a atualização, o uso do Copilot foi iniciado no mês de março de 2025 e se estendeu até o final de abril do mesmo ano.

Este estudo de caso se diferencia por proporcionar uma visão mais aplicada do uso do GitHub Copilot, considerando o contexto organizacional, o fluxo contínuo de trabalho e o impacto da ferramenta no desempenho de desenvolvedores em um ambiente real. A análise dos dados coletados contribuirá para verificar se os padrões observados entre estudantes se repetem em um cenário profissional, e para identificar possíveis diferenças entre os dois contextos de uso.

Considerações Éticas: Todos os participantes foram informados sobre os objetivos da pesquisa e participaram de forma voluntária, após consentimento informado. Os dados coletados foram tratados de maneira confidencial e anonimizados, assegurando a privacidade

[5] https://marketplace.eclipse.org/content/copilot4eclipse
[6] https://www.eclipse.org/

dos envolvidos. Além disso, os participantes assinaram o Termo de Consentimento Livre e Esclarecido (TCLE). O modelo do termo encontra-se disponível no Apêndice E.

4.2.2.1 Dados Coletados

Os dados coletados nesta etapa do estudo concentram-se em dois principais aspectos: o **tempo de execução das tarefas** e o **volume de código alterado** durante sua realização.

O tempo de execução foi calculado com base no fluxo operacional das tarefas, que segue um processo estruturado na ferramenta de gerenciamento adotada pela equipe. Nesse ambiente, uma tarefa é criada normalmente a partir de uma solicitação de usuários ou da própria demanda interna, podendo estar associada tanto à correção de erros quanto ao desenvolvimento de novas funcionalidades ou melhorias. Após sua criação e análise inicial, a tarefa é atribuída a um dos desenvolvedores responsáveis pela manutenção dos sistemas.

Quando o desenvolvedor inicia efetivamente o trabalho na tarefa, ela é movida para o status *"Em Progresso"*. A partir desse ponto, o tempo de execução é monitorado até sua transição para um dos status que indicam a finalização técnica do desenvolvimento, como *"Para Homologação"*, *"Em Homologação"*, *"Para Deploy"*, *"Resolvido"* ou *"Finalizado"*. Esse intervalo de tempo reflete, portanto, o período em que o desenvolvedor esteve ativamente engajado na resolução da demanda.

Adicionalmente, os sistemas desenvolvidos pela STI são versionados por meio do sistema de controle de versão Git. Cada tarefa possui uma associação explícita com os commits realizados durante sua execução, permitindo relacionar diretamente o volume de alterações no código-fonte (linhas adicionadas, modificadas ou removidas) com a respectiva atividade registrada no sistema de tarefas. A Figura 2 ilustra essa associação, mostrando um exemplo da interface da ferramenta de gerenciamento de tarefas com os registros vinculados ao repositório Git.

Essas duas fontes — a ferramenta de gerenciamento de tarefas e o repositório Git — fornecem os dados necessários para análise quantitativa do impacto do GitHub Copilot, permitindo avaliar indicadores relacionados à produtividade e ao esforço dos desenvolvedores no contexto real de trabalho.

4.3 Análise dos Dados

A fase de análise dos dados tem como objetivo responder à **QP3**, investigando como o uso de ferramentas de geração de código impacta o desempenho dos desenvolvedores em diferentes contextos. Nesta etapa, os dados coletados passarão por um *pré-processamento*, seguido de uma análise sob duas abordagens complementares: *quantitativa* e *qualitativa*.

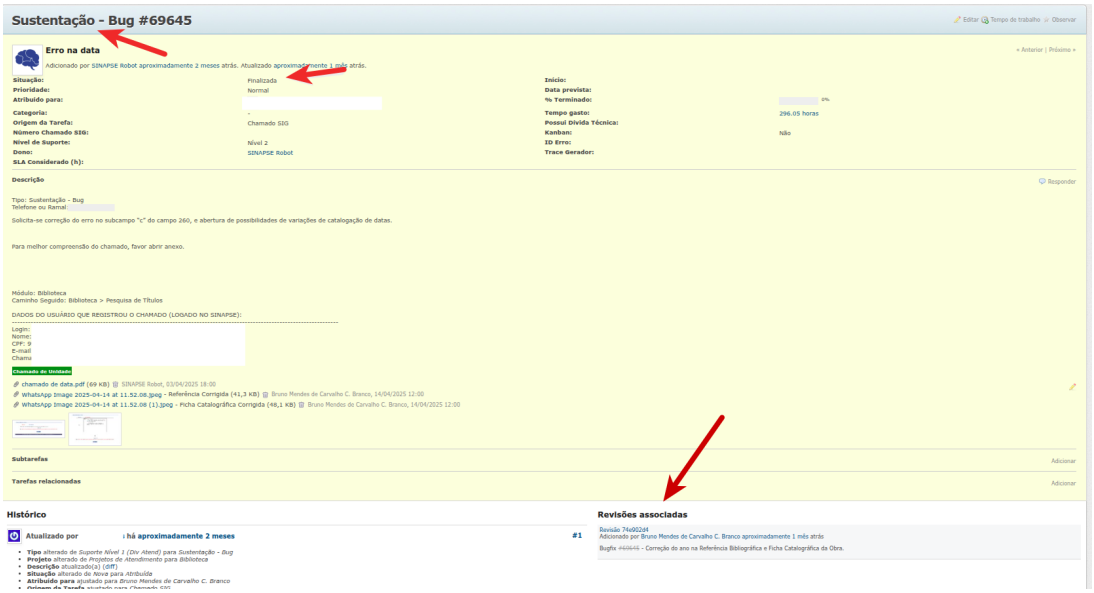


Figura 2 – Exemplo de uma tarefa no sistema de gerenciamento da STI, evidenciando a associação com commits no repositório Git.

Pré-processamento dos Dados

O pré-processamento, ou limpeza dos dados, representa o primeiro passo desta etapa, no qual as informações fornecidas pelos participantes nos formulários são verificadas e comparadas com outras fontes de dados, visando assegurar a fidedignidade e consistência das respostas. Esse procedimento é essencial para reduzir possíveis vieses e garantir a confiabilidade das informações antes da aplicação das abordagens analíticas.

Análise Quantitativa

A análise quantitativa com foco em dados objetivos, como o tempo de execução das tarefas e os indicadores de correção do código gerado. Esses dados foram tratados estatisticamente para identificar possíveis correlações entre o uso da ferramenta e a eficiência dos participantes.

Análise Qualitativa

A análise qualitativa foi conduzida com base nas respostas dos formulários de feedback preenchidos pelos participantes. As respostas foram analisadas tematicamente, buscando padrões nas percepções dos participantes sobre o uso da ferramenta, desafios enfrentados e sugestões de melhorias. Essa análise permite uma compreensão dos fatores que influenciam a aceitação e o uso eficaz da ferramenta em diferentes contextos.

4.4 Resultados Alcançados

O último passo desta abordagem é a divulgação dos resultados deste estudo, organizados em duas principais vertentes: a **dissertação** e a **produção de artigos científicos**.

Esses documentos incluem as análises detalhadas dos dados coletados, discussões sobre as limitações encontradas e recomendações práticas para o uso de ferramentas de geração de código em ambientes de desenvolvimento de software.

4.5 Considerações Finais

Neste capítulo, foram detalhadas as etapas metodológicas que orientam a realização deste estudo, desde a investigação preliminar até a coleta e análise de dados no contexto acadêmico e corporativo. A metodologia proposta busca responder de maneira estruturada às questões de pesquisa, utilizando indicadores de desempenho para avaliar o impacto das ferramentas de geração de código assistido por IA. A divisão da pesquisa em um estudo empírico com estudantes e um estudo de caso em ambiente empresarial permite obter uma visão abrangente dos efeitos do uso dessas ferramentas em diferentes contextos de experiência e aplicação.

Além disso, o compromisso com considerações éticas e o fornecimento de treinamentos iniciais destacam o rigor e a responsabilidade na condução do estudo, minimizando possíveis influências de variabilidade no domínio das tarefas e no uso da ferramenta. Com isso, espera-se que a análise dos resultados forneça dados para entender as potencialidades e limitações dessas ferramentas, contribuindo tanto para a academia quanto para a prática de desenvolvimento de software em diferentes contextos profissionais.

Nos capítulos seguintes, serão apresentados e discutidos os dados coletados, proporcionando uma visão sobre o desempenho dos participantes e as implicações práticas do uso de ferramentas de geração de código no desenvolvimento.

5 Resultados

5.1 Introdução

Este capítulo apresenta os resultados obtidos a partir da investigação empírica sobre o impacto do GitHub Copilot em atividades de desenvolvimento de software. A análise contempla dois contextos distintos: um experimento com estudantes e um estudo de caso com desenvolvedores profissionais. Em ambos os cenários, buscou-se comparar o desempenho das tarefas realizadas com e sem o uso da ferramenta, considerando indicadores como tempo de execução, falhas nos testes e ritmo de alterações no código. Os achados contribuem para responder à questão de pesquisa sobre como o uso de ferramentas de geração de código influencia o desempenho de desenvolvedores com diferentes níveis de experiência.

5.2 Estudo Empírico com Estudantes

5.2.1 Análise dos Resultados

A coleta de dados resultou em um conjunto de informações sobre o desempenho e o perfil dos participantes nas tarefas realizadas com e sem o uso do Copilot. Esses dados foram organizados e analisados com o intuito de extrair informações relevantes para responder às questões de pesquisa, especificamente no que se refere à execução das tarefas, ao Tempo Gasto e à quantidade de Testes Falhos obtida nas atividades propostas. Os dados completos estão disponíveis em Osorio (2025), onde podem ser consultados nos arquivos nomeados `experimento-dados-qualitativo.csv` e `experimento-dados-quantitativo.csv`, que compõem o repositório público do projeto de pesquisa.

Participaram do estudo um total de **49** estudantes, que realizaram atividades de recriar os método apagados, numerados de #1 a #4, conforme identificado na Tabela 2. No total, foram coletadas **184** respostas, distribuídas conforme a Tabela 4, vale destacar que eram esperadas 196 (4 x 49) respostas, porém nem todos os participantes conseguiram enviar a solução para os quatro problemas.

Conforme apresentado na Tabela 4, cada questão contou com um número específico de respostas, tanto com o uso do Copilot quanto sem. Observa-se que o total de respostas com o Copilot foi de 96, enquanto sem o Copilot foi de 88.

A análise desses dados fornece ideias sobre como o Copilot influenciou o tempo de resolução, a taxa de sucesso na execução das tarefas e a quantidade de Testes Falhos.

Tabela 4 – Distribuição das respostas por questão e uso do Copilot

Questão	Total de Respostas	Com Copilot	Sem Copilot
#1	46	22	24
#2	47	26	21
#3	46	22	24
#4	45	26	19
Total	184	96	88

Além disso, a variação no número de respostas por questão pode indicar diferenças na complexidade das tarefas ou na familiaridade dos participantes com os conceitos envolvidos.

5.2.2 Análise Exploratória dos Dados

Nesta subseção, é realizada uma análise exploratória dos dados coletados com o objetivo de observar características gerais das variáveis e identificar padrões e tendências. Esse exame inicial é essencial para contextualizar as condições experimentais com e sem o uso do GitHub Copilot e avaliar o impacto preliminar dessa ferramenta no desempenho dos estudantes nas tarefas de programação.

5.2.2.1 Estatísticas Descritivas

Para iniciar a análise exploratória, foram calculadas estatísticas descritivas das variáveis principais, incluindo *tempo gasto* e *número de testes falhos*. As medidas de tendência central (média e mediana) e de dispersão (desvio padrão) foram utilizadas para caracterizar a distribuição dos dados.

Essas estatísticas fornecem uma visão inicial sobre o desempenho médio dos participantes, permitindo observar, por exemplo, se o tempo médio gasto na realização das tarefas é reduzido com o uso do Copilot e se há uma diferença significativa no número de Testes Falhos nas duas condições experimentais (com e sem Copilot). A Tabela 5 apresenta as estatísticas descritivas para o *tempo gasto*, enquanto a Tabela 6 resume as estatísticas para o *número de testes falhos*.

As Tabelas 5 e 6 apresentam as estatísticas descritivas das variáveis *tempo gasto* e *número de testes falhos*, respectivamente, separadas entre as condições de uso e não uso do Copilot pelos participantes. A análise comparativa entre esses dois grupos possibilita uma avaliação preliminar do impacto da ferramenta nos desempenhos observados.

Na análise do **tempo gasto** (Tabela 5), a *média* para os participantes que utilizaram o Copilot foi de 24,73 minutos, enquanto para aqueles que não utilizaram a ferramenta, a média foi de 30,45 minutos. Esse dado sugere uma redução média no tempo de execução das tarefas para os participantes com suporte da ferramenta. Observa-se, ainda, que o *desvio padrão* é próximo entre os dois grupos (25,03 para o grupo com Copilot e 23,05 para

Tabela 5 – Estatísticas Descritivas do Tempo Gasto (em minutos), Separadas pelo Uso do Copilot

Indicador: Tempo(min)	**Não Usou Copilot**	**Usou Copilot**
$\text{Média}_{\text{Tempo Gasto}}$	30.45	**24.73**
$\text{Desvio Padrão}_{\text{Tempo Gasto}}$	**23.05**	25.03
$\text{Mediana}_{\text{Tempo Gasto}}$	23.5	**16.0**
$\text{Mínimo}_{\text{Tempo Gasto}}$	**4.0**	5.0
$\text{Máximo}_{\text{Tempo Gasto}}$	119.0	**125.0**

o grupo sem Copilot), indicando variações similares no tempo de execução em ambos os casos. A *mediana* também reflete essa tendência, sendo menor para o grupo que utilizou o Copilot (16 minutos) em comparação com o grupo que não utilizou (23,5 minutos), o que reforça a hipótese de que o uso da ferramenta pode ter contribuído para uma execução mais rápida das tarefas.

Tabela 6 – Estatísticas Descritivas do Número de Testes Falhos, Separadas pelo Uso do Copilot

Indicador: Testes Falhos	**Não Usou Copilot**	**Usou Copilot**
$\text{Média}_{\text{Testes Falhou}}$	**1.48**	1.65
$\text{Desvio Padrão}_{\text{Testes Falhou}}$	**1.84**	2.44
$\text{Mediana}_{\text{Testes Falhou}}$	1.0	1.0
$\text{Mínimo}_{\text{Testes Falhou}}$	0	0
$\text{Máximo}_{\text{Testes Falhou}}$	**10**	17

Na análise do **número de testes que falharam** (Tabela 6), a *média* para os participantes que utilizaram o Copilot foi de 1,65, enquanto para aqueles que não utilizaram a ferramenta, a média foi de 1,48. Esse dado sugere um leve aumento no número de falhas para os participantes com suporte da ferramenta. Observa-se, ainda, que o *desvio padrão* é maior no grupo com Copilot (2,44 em comparação com 1,84 no grupo sem Copilot), indicando uma maior variação no número de falhas entre os participantes que utilizaram a ferramenta. A *mediana* também reflete essa observação, sendo similar entre os dois grupos (1,0 em ambos), sugerindo que, apesar do aumento na média e na variabilidade de falhas para os usuários do Copilot, a maioria dos participantes teve um número semelhante de falhas. Esse achado pode ser explicado pela introdução de métodos auxiliares, que geraram inconsistências, pois os participantes foram orientados a recriar unicamente o método excluído e a utilizar os métodos auxiliares disponíveis em outros pontos dos sistemas.

5.2.3 Análise Visual dos Indicadores Normalizados

A Figura 3 apresenta a distribuição normalizada dos dois indicadores avaliados neste estudo: *Tempo Gasto* e *Número de Testes Falhos*, divididos entre as condições "Com

Copilot"e "Sem Copilot". A normalização dos dados foi realizada utilizando a técnica de Min-Max Scaling, que transforma os valores para um intervalo entre 0 e 1, permitindo comparações diretas entre indicadores de escalas diferentes. Esta abordagem foi necessária porque os dois indicadores possuem escalas distintas, sendo o *Tempo Gasto* medido em minutos e o *Número de Testes Falhos* representado por valores inteiros discretos, o que poderia comprometer a análise visual caso os dados não fossem ajustados.

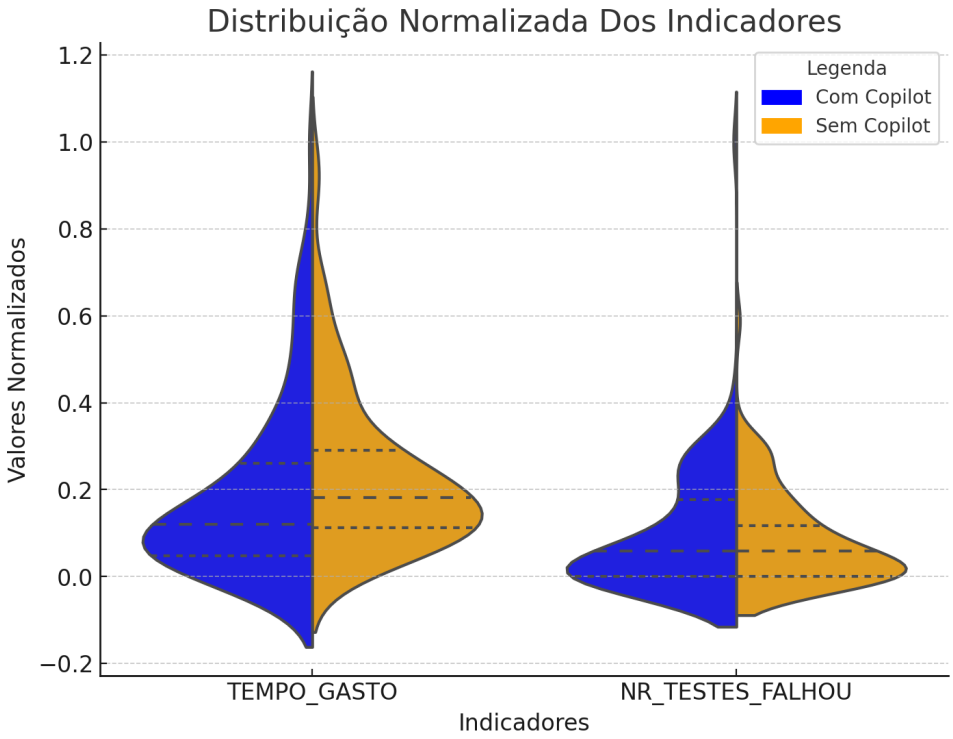


Figura 3 – Distribuição Normalizada Dos Indicadores por Uso do Copilot

Ao observar o **Tempo Gasto**, nota-se que a distribuição para "Com Copilot"apresenta uma concentração maior de valores em intervalos inferiores, indicando uma redução no tempo necessário para completar as tarefas em comparação à condição "Sem Copilot". O perfil da distribuição sugere que o uso do Copilot pode melhorar a eficiência dos desenvolvedores, uma tendência já apontada pelas análises estatísticas apresentadas anteriormente.

Já no **Número de Testes Falhos**, as distribuições normalizadas para ambas as condições mostram uma maior uniformidade, sem diferenças visíveis significativas entre as duas categorias. Isso reforça a conclusão das análises anteriores, indicando que o uso do Copilot não impactou de forma relevante a correção funcional do código produzido.

Por fim, a análise visual dos gráficos normalizados corrobora as conclusões quantitativas: enquanto o Copilot demonstra um impacto positivo na eficiência (*Tempo Gasto*), sua influência na qualidade do código (*Número de Testes Falhos*) é limitada. A normalização permitiu uma comparação mais justa entre os indicadores, evidenciando claramente os efeitos observados.

5.2.3.1 Análise de Normalidade e Teste de Hipótese

Para determinar a normalidade dos dados, foi aplicado o teste de Shapiro-Wilk nos tempos gastos e no número de Testes Falhos dos grupos com e sem o uso do Copilot. Abaixo estão os resultados obtidos:

- **Tempo Gasto (Usou Copilot)**: $p = 4.98 \times 10^{-10}$
- **Tempo Gasto (Não Usou Copilot)**: $p = 1.52 \times 10^{-8}$
- **Número de Testes Falhos (Usou Copilot)**: $p = 2.76 \times 10^{-13}$
- **Número de Testes Falhos (Não Usou Copilot)**: $p = 4.53 \times 10^{-10}$

Os valores de p inferiores ao nível de significância convencional ($\alpha = 0.05$) (CHARNESS; GNEEZY; KUHN, 2012) indicam que a hipótese nula de normalidade dos dados pode ser rejeitada. Assim, os dados não apresentam distribuição normal.

Diante disso, optou-se pelo **teste de Mann-Whitney U** (ou *Wilcoxon rank-sum*), que é adequado para comparar grupos não paramétricos (KITCHENHAM et al., 2017; CHARNESS; GNEEZY; KUHN, 2012) e permite avaliar a diferença nas medianas entre os grupos com e sem o uso do Copilot.

Resultados do Teste de Mann-Whitney U:

- **Tempo Gasto**: $U = 3148.0$, $p = 0.0029$ — indica uma diferença estatisticamente significativa no tempo gasto entre os grupos.
- **Número de Testes Falhos**: $U = 4075.0$, $p = 0.866$ — não indica diferença significativa entre os grupos para o número de Testes Falhos.

Interpretação: Este teste avalia se uma variável tende a ter valores maiores ou menores em um grupo em comparação a outro, sem a necessidade de pressupor distribuição normal. A análise revelou que o uso do Copilot impacta significativamente o tempo de execução das tarefas, mas não apresenta diferença significativa no número de testes que falharam em relação ao grupo que não utilizou a ferramenta.

5.2.3.2 Cliff's Delta

Para complementar a análise estatística realizada com o teste de Mann-Whitney U, foi calculado o *Cliff's delta* para as variáveis *Tempo Gasto* e *Número de Testes Falhos*, a fim de avaliar a magnitude do efeito. O *Cliff's delta* é uma medida não paramétrica que indica a proporção de pares ordenados entre dois grupos, sendo útil para comparar distribuições de maneira independente de sua escala.

Os resultados são apresentados a seguir:

- **Tempo Gasto:** O valor de $\delta = -0.254$ indica um efeito *pequeno* (*small*). Isso sugere que há uma tendência de os tempos de execução sem o uso do Copilot serem maiores do que aqueles com o uso da ferramenta, corroborando os resultados do teste de Mann-Whitney U, que indicaram uma diferença estatisticamente significativa.

- **Número de Testes Falhos:** O valor de $\delta = -0.014$ indica um efeito *negligenciável* (*negligible*). Este resultado reforça que não há diferenças relevantes entre os grupos com e sem Copilot em relação ao número de testes falhos, conforme também observado no teste de Mann-Whitney U.

Esses resultados complementam a análise anterior, evidenciando que, enquanto o Copilot contribui para uma maior eficiência em termos de tempo, seu impacto na qualidade do código gerado, medido pelo número de testes falhos, é insignificante. O uso do *Cliff's delta* permite, portanto, uma interpretação mais detalhada da magnitude das diferenças entre os grupos.

5.2.4 Análise Comparativa entre Turmas

Considerando que os participantes deste estudo pertencem a turmas com diferentes níveis de formação acadêmica[1]: EngSW (Graduação - 5º período, 21 participantes), POO (Graduação - 2º período, 17 participantes) e Pós-graduação (11 participantes), foi realizada uma análise comparativa adicional com o objetivo de investigar se o uso do GitHub Copilot teve efeitos distintos conforme o nível dos estudantes.

A análise foi estratificada por turma e por uso do Copilot, permitindo observar se estudantes mais experientes se beneficiaram mais (ou menos) da ferramenta em termos de tempo e qualidade das tarefas. As Tabelas 7 e 8 apresentam as estatísticas descritivas das variáveis *Tempo Gasto* e *Número de Testes Falhos*, respectivamente.

Observa-se que, nas turmas de graduação (EngSW e POO), a média e a mediana do tempo gasto foram consistentemente menores para os participantes que utilizaram o Copilot. Por exemplo, na turma POO, a média caiu de 34,06 minutos (sem Copilot) para 23,86 minutos (com Copilot), enquanto na EngSW a redução foi de 25,89 para 19,11 minutos. Esse padrão sugere que o uso da ferramenta pode ter contribuído para uma maior eficiência na realização das tarefas.

Por outro lado, na turma da pós-graduação, os valores médios e medianos de tempo gasto foram similares com e sem o uso da ferramenta, o que indica que o Copilot não teve impacto expressivo na performance temporal desses participantes mais experientes.

No que se refere ao número de testes falhos, os resultados foram mais homogêneos entre os grupos. As médias variaram pouco, e em todas as turmas a mediana foi igual

[1] <https://sigaa.ufpi.br/sigaa/public/curso/curriculo.jsf?lc=pt_BR&id=74268>

ou próxima, geralmente em torno de 1 falha. Na POO, por exemplo, a média de falhas foi praticamente idêntica (1,46 com Copilot e 1,38 sem). Isso sugere que a qualidade funcional do código produzido — medida indiretamente pela quantidade de erros — não foi significativamente afetada pelo uso do Copilot em nenhum dos níveis analisados.

Tabela 7 – Estatísticas Descritivas do Tempo Gasto (min), por Turma e Uso do Copilot

Turma	**Uso**	**Média**	**Mediana**	**Desvio Padrão**
EngSW	Com Copilot	19,11	15,0	22,36
EngSW	Sem Copilot	25,89	19,0	19,19
POO	Com Copilot	23,86	13,0	24,71
POO	Sem Copilot	34,06	25,0	28,40
Pós-graduação	Com Copilot	33,46	18,0	27,31
Pós-graduação	Sem Copilot	34,02	20,0	26,15

Tabela 8 – Estatísticas Descritivas do Número de Testes Falhos, por Turma e Uso do Copilot

Turma	**Uso**	**Média**	**Mediana**	**Desvio Padrão**
EngSW	Com Copilot	1,91	1,0	3,10
EngSW	Sem Copilot	1,29	1,0	1,60
POO	Com Copilot	1,46	0,0	2,02
POO	Sem Copilot	1,38	1,0	1,68
Pós-graduação	Com Copilot	1,56	1,0	1,94
Pós-graduação	Sem Copilot	1,43	1,0	1,96

A seguir, as Figuras 4 e 5 apresentam a distribuição dos dados por meio de gráficos de violino, permitindo uma análise visual mais clara das diferenças entre os grupos.

A Figura 4 mostra que, nas turmas de graduação (EngSW e POO), a distribuição do tempo com o uso do Copilot tende a se concentrar em valores mais baixos, indicando uma redução no tempo necessário para completar as tarefas. Em contraste, na turma da pós-graduação, as distribuições com e sem o Copilot são semelhantes, sugerindo um impacto reduzido da ferramenta para esse grupo mais experiente.

A Figura 5 revela que o número de testes falhos apresenta padrões visuais semelhantes entre os grupos com e sem Copilot, com distribuições sobrepostas e sem grandes variações. Isso sugere que a qualidade funcional das soluções não foi substancialmente afetada pelo uso da ferramenta, independentemente da turma.

Para verificar estatisticamente essas observações, foram aplicados o teste de Mann-Whitney U e a medida de efeito Cliff's Delta para comparar, dentro de cada turma, os grupos com e sem Copilot.

- **EngSW:** o uso do Copilot reduziu significativamente o tempo gasto ($p = 0{,}0297$),

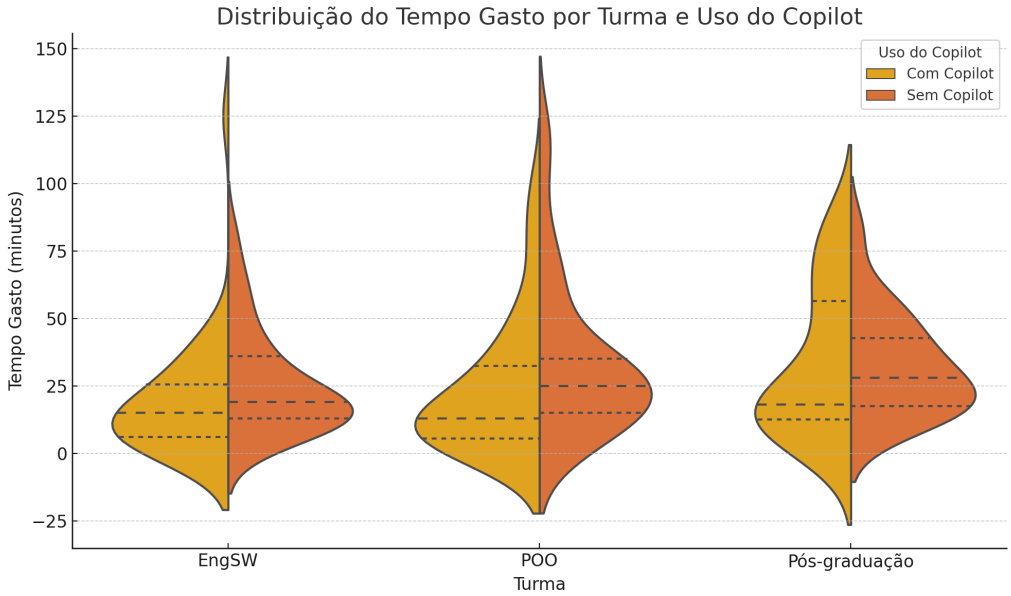


Figura 4 – Distribuição do Tempo Gasto por Turma e Uso do Copilot

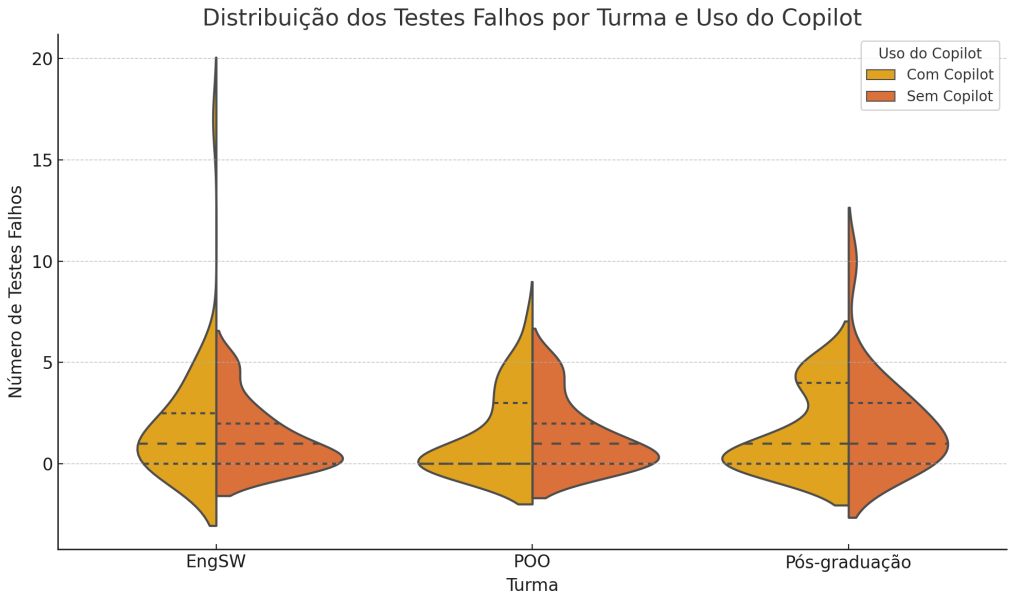


Figura 5 – Distribuição do Número de Testes Falhos por Turma e Uso do Copilot

com um efeito de magnitude *pequena* ($\delta = -0{,}303$). Para o número de testes falhos, não houve diferença significativa ($p = 0{,}4799$), com efeito *negligenciável*.

- **POO:** também houve redução significativa no tempo com o uso do Copilot ($p = 0{,}0223$), com *efeito pequeno* ($\delta = -0{,}323$). A variável *número de testes falhos* apresentou novamente efeito *negligenciável*.

- **Pós-graduação:** diferentemente das turmas de graduação, não foi observada diferença estatisticamente significativa no tempo ($p = 0{,}5347$), com *efeito negligenciável* ($\delta = -0{,}110$). Isso sugere que os participantes mais experientes não se beneficiaram

em termos de tempo com o uso da ferramenta. Da mesma forma, não houve diferença relevante no número de erros.

Em síntese, os resultados indicam que o GitHub Copilot trouxe ganhos de eficiência para estudantes de graduação, especialmente aqueles em níveis mais iniciais, ao passo que seu impacto na pós-graduação foi nulo ou marginal. Por outro lado, em nenhuma das turmas o uso da ferramenta comprometeu a qualidade das soluções geradas.

5.2.5 Análise Qualitativa

A análise qualitativa buscou compreender a percepção dos participantes sobre o uso do GitHub Copilot, a efetividade das oficinas oferecidas e os desafios enfrentados no processo de aprendizado das tecnologias envolvidas.

5.2.5.1 Perfil dos Participantes

O perfil dos participantes do estudo foi analisado com base em três aspectos principais: turma de origem, tempo de experiência em programação e autoavaliação das habilidades técnicas.

A Tabela 9 apresenta a distribuição dos participantes por turma. A turma EngSW corresponde a estudantes do 5º período da graduação, POO ao 2º período da graduação, e a terceira turma é composta por alunos da pós-graduação. A presença de estudantes com diferentes níveis de formação possibilita uma análise comparativa mais diversificada quanto à familiaridade com ferramentas e tecnologias.

Tabela 9 – Distribuição dos Participantes por Turma

Turma	**Quantidade**
EngSW (Graduação – 5º período)	21
POO (Graduação – 2º período)	17
Pós-graduação	11

Quanto ao tempo de experiência com programação (Tabela 10), observa-se que a maioria dos participantes (aproximadamente 68%) declarou ter até dois anos de prática, o que é coerente com os níveis iniciais da formação universitária. Apenas uma minoria relatou mais de cinco anos de envolvimento com desenvolvimento de software.

Por fim, foi aplicada uma **Escala de Likert de cinco pontos** para que os participantes classificassem suas habilidades em programação, sendo 1 equivalente a "Iniciante" e 5 a "Muito Avançado". Os resultados são apresentados na Tabela 11. A maior concentração ocorreu no ponto 3 da escala (intermediário), escolhido por 19 participantes, seguido do ponto 2 (básico) por 14 participantes. Apenas dois participantes se consideraram

Tabela 10 – Tempo de Experiência com Programação

Tempo de Experiência	Quantidade
1 ano	14
2 anos	12
Menos de 1 ano	8
De 6 a 10 anos	8
3 anos	4
De 4 a 5 anos	4

muito avançados. De acordo com a interpretação clássica da escala de Likert (LIKERT, 1932), esses dados sugerem uma predominância de estudantes que ainda estão em processo de consolidação de suas habilidades técnicas, o que é coerente com o tempo de experiência previamente analisado.

Tabela 11 – Autoavaliação das Habilidades de Programação (Escala de Likert)

Autoavaliação (1–5)	Quantidade
1 (Iniciante)	7
2 (Básico)	14
3 (Intermediário)	19
4 (Avançado)	8
5 (Muito Avançado)	2

5.2.5.2 Esforço no Aprendizado das Tecnologias

Foi solicitada aos participantes uma autoavaliação do esforço necessário para aprender diferentes aspectos abordados no estudo, utilizando uma **Escala de Likert de 0 a 5 pontos**, em que 0 representa nenhum esforço e 5 representa esforço máximo. As tecnologias avaliadas foram: (i) compreensão da arquitetura das APIs RESTful, (ii) uso do framework Spring Boot, e (iii) uso da ferramenta GitHub Copilot.

A Tabela 12 apresenta a distribuição das respostas entre os diferentes níveis da escala.

Tabela 12 – Distribuição do Esforço no Aprendizado das Tecnologias (Escala de Likert)

Esforço (1–5)	APIs e Arquitetura	Spring Boot	Copilot
1 (Muito Baixo)	2	1	17
2 (Baixo)	4	3	8
3 (Moderado)	16	15	12
4 (Alto)	19	21	2
5 (Muito Alto)	8	9	11

Ao analisar os dados, observa-se que o **uso do Copilot foi percebido como menos exigente** em termos de esforço: 25 participantes (mais da metade) o classificaram entre 1 (Muito Baixo) e 2 (Baixo), enquanto apenas 2 o avaliaram com esforço alto (nível 4). Apesar disso, 11 participantes atribuíram nota máxima (5), o que sugere que a experiência com a ferramenta pode ter variado de acordo com o grau de familiaridade prévia com assistentes de código.

Por outro lado, o **Spring Boot** foi percebido como mais exigente: 30 dos participantes o classificaram com esforço alto (níveis 4 ou 5), e apenas 4 o classificaram entre 1 e 2. A arquitetura das APIs REST também foi percebida como desafiadora, com 27 participantes classificando o esforço entre 4 e 5.

Esses dados reforçam a ideia de que tecnologias tradicionais de *backend* como Spring Boot e arquitetura RESTful exigem maior dedicação e preparação prévia, enquanto ferramentas de apoio baseadas em IA, como o Copilot, foram mais acessíveis à maioria dos participantes — ainda que com alguma variação individual. A interpretação segue os princípios da escala de Likert, segundo os quais valores mais altos indicam maior esforço percebido (LIKERT, 1932).

5.2.5.3 Percepção das Oficinas

A análise da percepção dos participantes em relação às oficinas ministradas previamente ao experimento revelou uma avaliação amplamente positiva, tanto em termos de utilidade quanto em relação à sua organização e formato.

Conforme apresentado na Tabela 13, 23 participantes classificaram as oficinas como *muito úteis* e 21 como *úteis*, totalizando 44 respostas favoráveis (aproximadamente 88% da amostra). Apenas dois participantes avaliaram negativamente as oficinas.

Tabela 13 – Avaliação Geral das Oficinas

Classificação	**Quantidade**
Muito úteis	23
Úteis	21
Neutro	4
Pouco úteis	1
Não foram úteis	1

A Tabela 14 mostra a avaliação quanto à *quantidade* de oficinas. Assim como na avaliação geral, a maioria dos participantes demonstrou satisfação: 20 avaliaram como *muito úteis* e outros 20 como *úteis*, enquanto apenas três participantes apresentaram avaliações negativas.

Com relação ao formato on-line das oficinas, a maioria dos participantes (72%) discordou que essa característica tenha impactado negativamente o desempenho, conforme

Tabela 14 – Avaliação da Quantidade de Oficinas

Classificação	**Quantidade**
Muito úteis	20
Úteis	20
Neutro	7
Pouco úteis	2
Não foram úteis	1

a Tabela 15. Essa percepção demonstra que o modelo remoto foi bem aceito.

Tabela 15 – Impacto Percebido do Formato On-line

Classificação	**Quantidade**
Discordo totalmente	21
Discordo mais ou menos	12
Concordo mais ou menos	11
Concordo totalmente	2

Adicionalmente, a Tabela 16 evidencia a aceitação do recurso de gravações. Um total de 44 participantes concordaram que as gravações foram úteis para entender melhor as ferramentas utilizadas — sendo 38 deles de forma totalmente favorável.

Tabela 16 – Avaliação das Gravações das Oficinas

Classificação	**Quantidade**
Concordo totalmente	38
Concordo mais ou menos	6
Discordo totalmente	2

Esses dados indicam que a estratégia de formação adotada — baseada em encontros remotos e com suporte por gravações — foi bem-sucedida em apoiar o aprendizado dos participantes e em prepará-los adequadamente para a realização das tarefas experimentais.

5.2.5.4 Desafios ao Usar o Copilot

Entre os participantes que utilizaram o GitHub Copilot, alguns relataram dificuldades ou limitações durante o uso da ferramenta. A análise qualitativa dessas respostas foi conduzida com base na técnica de *Análise de Conteúdo* proposta por Bardin (1977), que permite a codificação e categorização sistemática das manifestações textuais em unidades de significado. A metodologia envolveu leitura, identificação de temas recorrentes e contagem de frequência.

A Tabela 17 resume os principais desafios relatados, organizados por tema, acompanhados de uma breve descrição e exemplos representativos das respostas.

Tabela 17 – Desafios Relatados no Uso do GitHub Copilot

Tema	Descrição	Exemplo	Total
Problemas de conexão	Dificuldade de envio ou recebimento de contexto pelo Copilot	O arquivo de contexto enviado ao copilot às vezes não é compreendido...	4
Instabilidade do chat	Mau funcionamento do chat embutido no VSCode	Por vezes, o chat embutido no vscode apresentava lentidão ou não respondia...	4
Sugestões incoerentes	Respostas imprecisas, genéricas ou fora do escopo da tarefa	Muitas vezes ele não entende exatamente a pergunta e retorna algo que não ajuda...	6
Reescrita excessiva de código	Substituição indevida de blocos já existentes	Ele escrevia o código inteiro de uma sessão e apagava os métodos que eu já havia feito...	1
Necessidade de ajustes manuais	Demandava intervenções humanas mesmo com sugestões geradas	Encontrei algumas dificuldades ao utilizar essa ferramenta e realizei alguns ajustes...	2
Inexperiência técnica	Dificuldade em compreender ou aproveitar o Copilot por limitação de conhecimento prévio	Ainda estou aprendendo a usar essa IA e às vezes não sabia o que pedir...	3
Limitações por ser pago	Barreiras associadas ao plano gratuito ou limitações da versão usada	Acredito que por não ser pago, o Copilot tem algumas limitações no que entrega...	1

Os dados mostram que a principal crítica está relacionada à qualidade e pertinência das sugestões geradas pelo Copilot (tema *Sugestões incoerentes*), seguido por problemas técnicos como falhas de conexão e instabilidade do chat integrado. Participantes com menor familiaridade com a ferramenta ou com o ambiente de desenvolvimento também relataram dificuldades decorrentes da inexperiência técnica.

Esses achados evidenciam que, embora o Copilot possa oferecer suporte à codificação, seu uso ainda apresenta limitações técnicas e pedagógicas que afetam a experiência de parte dos usuários.

5.2.5.5 Confiança na Manutenção com Copilot

Com o objetivo de compreender a percepção dos participantes quanto à utilidade prática do GitHub Copilot, foi incluída no questionário a seguinte pergunta: *"Você sentiria mais confiante de fazer manutenção de um sistema, se tivesse disponível a ferramenta Copilot?"*

A maioria dos participantes (88%) respondeu afirmativamente, conforme apre-

sentado na Tabela 18. Esse resultado sugere que a ferramenta tem potencial para ser percebida como um apoio confiável no contexto de manutenção de sistemas, aumentando a autoconfiança dos desenvolvedores durante o processo de codificação.

Tabela 18 – Confiança para Manutenção com o uso do Copilot

Resposta	**Quantidade**
Sim	44
Não	6

Embora uma minoria tenha expressado não se sentir mais confiante com a presença do Copilot, não foram fornecidas justificativas abertas que permitissem identificar os motivos dessa percepção. Esse achado pode estar relacionado a experiências individuais de uso ou limitações percebidas durante as oficinas.

5.2.6 Discussão dos Resultados do Estudo Empírico

A análise dos resultados obtidos permite algumas considerações relevantes sobre o impacto do uso do Copilot no contexto do desenvolvimento de software por estudantes. Os dados descritivos e os testes de hipótese indicaram uma redução significativa no tempo necessário para completar as tarefas quando os participantes utilizaram o Copilot. Esse achado está alinhado com a literatura, que sugere que ferramentas de geração de código assistido por IA podem atuar como facilitadores no desenvolvimento, permitindo que os desenvolvedores concluam tarefas mais rapidamente ao delegarem partes rotineiras ou repetitivas do código à ferramenta, como apresentado em Peng et al. (2023).

Por outro lado, o número de Testes Falhos, avaliado por meio de testes unitários, não apresentou diferença estatisticamente significativa entre os grupos que usaram e não usaram o Copilot. Esse resultado sugere que, apesar da eficiência temporal proporcionada pela ferramenta, sua utilização não necessariamente melhora a qualidade do código em termos de correção funcional. De fato, essa conclusão vai ao encontro de estudos que indicam que a assistência ao código pode oferecer soluções rápidas, mas requer a análise crítica do desenvolvedor para manter a qualidade desejada Yetiştiren et al. (2023). Essa observação reforça a importância de orientar os desenvolvedores quanto ao uso consciente e criterioso das sugestões de código oferecidas pelo Copilot.

A análise qualitativa reforça e complementa esses achados. A maioria dos participantes relatou se sentir mais confiante na manutenção de sistemas com o suporte do Copilot, evidenciando o impacto positivo da ferramenta na autoconfiança dos desenvolvedores. Além disso, o esforço percebido para aprender a utilizar o Copilot foi inferior ao esforço necessário para dominar tecnologias como Spring Boot e arquitetura RESTful, o que indica uma curva de aprendizado mais acessível para ferramentas baseadas em IA.

No entanto, desafios técnicos também foram relatados, como instabilidade no chat, sugestões incoerentes e problemas de integração no ambiente de desenvolvimento. Essas limitações reforçam a necessidade de um acompanhamento pedagógico adequado ao introduzir tais ferramentas em contextos educacionais. Ainda assim, a percepção geral sobre as oficinas (de apresentação das tecnologias) foi amplamente positiva, com destaque para o papel das gravações e da metodologia adotada.

Portanto, a discussão dos dados quantitativos e qualitativos sugere que o Copilot pode ser um aliado relevante no processo de aprendizagem e desenvolvimento, sobretudo quando seu uso é mediado por estratégias didáticas que promovam senso crítico e responsabilidade na aceitação das sugestões geradas automaticamente.

5.3 Estudo de Caso em Ambiente Profissional

Com o objetivo de aprofundar a análise do impacto do GitHub Copilot em ambientes reais de desenvolvimento de software, foi conduzido um estudo de caso na Superintendência de Tecnologia da Informação (STI) da Universidade Federal do Piauí (UFPI). A STI é responsável pela gestão e manutenção dos Sistemas Integrados de Gestão (SIG), um conjunto de sistemas corporativos amplamente utilizados pelos setores acadêmicos e administrativos da universidade.

Os sistemas SIG são compostos pelos seguintes módulos principais: SIGAA (Sistema Integrado de Gestão de Atividades Acadêmicas), voltado para a gestão acadêmica da instituição, incluindo atividades de ensino, pesquisa, extensão e processos relacionados à vida acadêmica; SIPAC (Sistema Integrado de Gestão de Patrimônio, Administração e Contratos), que gerencia os processos administrativos, financeiros, de compras, contratos e patrimônio; SIGRH (Sistema Integrado de Gestão de Recursos Humanos), responsável pela administração dos processos de gestão de pessoal, folha de pagamento, férias, ponto eletrônico, entre outros; SIGEleição (Sistema de Controle de Processos Eleitorais), utilizado na condução de eleições institucionais internas; e SIGAdmin (Sistema de Administração e Comunicação), que oferece suporte a processos de comunicação institucional e administração geral.

O desenvolvimento e manutenção desses sistemas são realizados utilizando uma arquitetura tecnológica consolidada, embora atualmente considerada legada. As tecnologias adotadas incluem: Java 6, Hibernate 3.2.5, JavaServer Faces (JSF) 1.2, RichFaces 3.3.3, Struts 1.2, EJB 2.1, Spring 3.2.1 e JBoss 5.1.0. Essa arquitetura, baseada em tecnologias tradicionais da plataforma Java EE[2], apresenta desafios relevantes tanto no aspecto de manutenção e evolução dos sistemas quanto na adoção de ferramentas modernas que possam apoiar e otimizar o esforço de desenvolvimento, como é o caso do GitHub Copilot.

[2] https://www.oracle.com/br/java/technologies/java-ee-glance.html

Participaram deste estudo quatro desenvolvedores que atuam diretamente na manutenção e evolução dos sistemas SIG. Um quinto integrante da equipe foi excluído da amostra por ser um colaborador recém-integrado, sem histórico de atividades no período anterior à introdução do Copilot — requisito necessário para que fosse possível realizar comparações consistentes entre os períodos com e sem uso da ferramenta.

Diferentemente do estudo experimental conduzido com estudantes, que foi baseado em tarefas específicas e controladas, os participantes deste estudo de caso realizaram suas atividades rotineiras normalmente, sem qualquer intervenção externa na definição, atribuição ou execução das tarefas. Este delineamento buscou avaliar o impacto do Copilot em um cenário realista, mantendo as características operacionais do ambiente de desenvolvimento e minimizando interferências no fluxo de trabalho.

Para viabilizar o uso do GitHub Copilot no ambiente de desenvolvimento adotado pela equipe, foi necessária a instalação da extensão Copilot4Eclipse (versão 1.3.0), compatível com a versão 2023-06 (4.28) do Eclipse. Este processo exigiu que todos os desenvolvedores atualizassem suas versões do Eclipse, uma vez que utilizavam versões antigas e incompatíveis com a extensão. Concluído esse ajuste, o uso do Copilot foi oficialmente iniciado em março de 2025, estendendo-se até o final de maio do mesmo ano.

Os dados coletados nesta etapa da pesquisa concentram-se, principalmente, no tempo de execução das tarefas, mensurado com base no intervalo entre o momento em que uma tarefa é movida para o status “Em Progresso” e sua conclusão em um dos seguintes status: “Para Homologação”, “Em Homologação”, “Para Deploy”, “Resolvido” ou “Finalizado”. Essas informações foram extraídas diretamente da ferramenta de gerenciamento de tarefas utilizada pela equipe. Além disso, foram obtidos dados do repositório Git da instituição, que permitiram calcular o volume de código alterado (linhas adicionadas, modificadas e removidas), complementando a análise quantitativa com métricas relacionadas ao esforço de desenvolvimento e ao ritmo de alterações no código.

Este estudo de caso se destaca por oferecer uma visão aplicada e concreta sobre o uso do GitHub Copilot em um ambiente institucional público, considerando as particularidades do contexto organizacional, o fluxo contínuo de trabalho e as demandas reais do desenvolvimento de software. A análise dos dados permitirá verificar se os padrões observados na fase experimental com estudantes se mantêm em um contexto profissional e, além disso, contribuirá para identificar semelhanças e diferenças entre os dois cenários, possibilitando uma compreensão mais abrangente dos efeitos do Copilot sobre o esforço de desenvolvimento e a dinâmica das atividades de codificação.

5.3.1 Caracterização dos Desenvolvedores Participantes

Esta subseção apresenta informações sobre os quatro desenvolvedores que participaram do estudo de caso, todos atuando diretamente na manutenção e evolução dos Sistemas Integrados de Gestão (SIG) da Universidade Federal do Piauí (UFPI). A caracterização inclui aspectos relacionados ao tempo de experiência em programação e ao período de atuação nos projetos dos sistemas SIG, com base nas respostas fornecidas por meio de um questionário, permitindo contextualizar o **esforço de desenvolvimento** analisado posteriormente.

Os desenvolvedores possuem **mais de cinco anos de experiência em programação**, o que indica um nível elevado de maturidade técnica e conhecimento acumulado na área de desenvolvimento de software. Em relação ao tempo de atuação nos sistemas SIG, os mesmos estão envolvidos há **mais de três anos** com os projetos, o que evidencia familiaridade com o contexto institucional, os padrões de desenvolvimento adotados e os fluxos de manutenção e evolução dos sistemas.

Cabe destacar que, além das tarefas analisadas nesta pesquisa, os quatro desenvolvedores também atuam em outras atividades relacionadas aos sistemas, como suporte técnico, levantamento de requisitos, testes, reuniões com usuários e tarefas administrativas. Assim, as tarefas sumarizadas neste estudo não representam a totalidade do trabalho desempenhado por esses profissionais, mas sim um recorte específico voltado à análise do impacto do uso do GitHub Copilot **sobre o esforço de desenvolvimento aplicado a tarefas rotineiras**.

Durante o período com uso do Copilot, um dos desenvolvedores precisou se afastar temporariamente por motivos de saúde e, por isso, não participou ativamente dessa etapa da coleta. Sendo assim, os resultados quantitativos e qualitativos apresentados nas seções seguintes concentram-se nos três desenvolvedores que permaneceram ativos ao longo de todo o período de observação com a ferramenta.

Essas informações contribuem para uma compreensão mais contextualizada dos resultados obtidos, considerando que os participantes possuíam experiência tanto em desenvolvimento quanto nos sistemas analisados, o que pode influenciar diretamente sua adaptação ao uso de ferramentas baseadas em inteligência artificial, como o GitHub Copilot.

5.3.2 Descrição das Tarefas Analisadas

O conjunto de dados analisado neste estudo é composto por tarefas realizadas por quatro desenvolvedores que atuam na manutenção e evolução dos Sistemas Integrados de Gestão (SIG) da Universidade Federal do Piauí (UFPI). A análise contempla tarefas realizadas tanto no período anterior quanto no período posterior à adoção do GitHub

Copilot, permitindo uma avaliação comparativa do **esforço de desenvolvimento** nos dois contextos. Os dados utilizados encontram-se disponíveis no repositório público do projeto Osorio (2025), especificamente nos arquivos `estudo-de-caso-qualitativo.csv` e `estudo-de-caso-quantitativo.csv`.

As tarefas foram classificadas em duas categorias principais: **Customização** e **Bugfix**. A categoria **Customização** abrange atividades destinadas a adaptar funcionalidades já existentes para atender a demandas específicas de usuários ou setores. Já a categoria **Bugfix** consolida atividades de manutenção corretiva e evolutiva, reunindo tanto a resolução de erros quanto melhorias incrementais e ajustes, agrupando as tarefas originalmente classificadas como *Sustentação - Aprimoramento* e *Sustentação - Bug*.

Tarefas categorizadas como orientações, reuniões, verificações administrativas ou outras que não envolviam alterações no código-fonte foram agrupadas sob o rótulo **Suporte/Documentação** e, por não refletirem esforço direto de desenvolvimento, foram excluídas desta análise.

Para fins de segmentação temporal, considerou-se como período **sem Copilot** as tarefas iniciadas nos meses de março, abril e maio de 2024. Já o período **com Copilot** abrange exclusivamente os meses de março, abril e maio de 2025, intervalo no qual a ferramenta foi oficialmente adotada pela equipe.

A Tabela 19 apresenta a distribuição das tarefas por desenvolvedor, agrupadas por tipo e período (com e sem uso do GitHub Copilot), já considerando o corte trimestral definido para a análise.

Observa-se que o Desenvolvedor 72 concentrou o maior volume de atividades, totalizando 40 tarefas, sendo 31 de *BugFix* e 9 de *Customização*. Desse total, 10 tarefas de *BugFix* e 7 de *Customização* foram realizadas no período com uso do Copilot, enquanto 21 tarefas de *BugFix* e 2 de *Customização* foram executadas no período sem a ferramenta.

O Desenvolvedor 48 aparece com apenas 4 tarefas no recorte, sendo 3 de *BugFix* e 1 de *Customização*. Nesse caso, a maioria das atividades foi realizada no período sem Copilot (2 *BugFix*), enquanto as demais (1 *BugFix* e 1 *Customização*) ocorreram no período com a ferramenta.

Por fim, o Desenvolvedor 82 contabilizou 4 tarefas, sendo 1 de *BugFix* e 3 de *Customização*. No período com Copilot foram observadas 2 tarefas de *Customização*, enquanto no período sem Copilot registraram-se 1 tarefa de *BugFix* e 1 de *Customização*.

De modo geral, nota-se que o maior volume de tarefas, tanto com quanto sem Copilot, foi realizado pelo Desenvolvedor 72, o que reflete sua predominância na execução das atividades dentro do período analisado.

Esses dados refletem não apenas a natureza das demandas enfrentadas pela equipe ao

Tabela 19 – Quantidade de tarefas por desenvolvedor, tipo e período (com e sem Copilot, corte trimestral)

Desenvolvedor (ID)	Tipo	Com Copilot	Sem Copilot	Total
48	BugFix	1	2	3
48	Customização	1	0	1
72	BugFix	10	21	31
72	Customização	7	2	9
82	BugFix	0	1	1
82	Customização	2	1	3

longo do período analisado, mas também possíveis especializações entre os desenvolvedores, bem como a dinâmica das atividades de manutenção e evolução dos sistemas institucionais.

5.3.3 Preparação dos Dados para Análise do Esforço de Desenvolvimento

Para avaliar o esforço de desenvolvimento dos participantes no contexto do uso do GitHub Copilot, foi necessário realizar um processo de preparação e consolidação dos dados coletados. Inicialmente, identificou-se que uma mesma tarefa (representada por um único identificador *id*) poderia possuir múltiplos registros na base de dados. Essa duplicidade ocorre devido à presença da coluna *revision*, que indica versões distintas de commits relacionados à mesma tarefa. Cada revisão inclui informações sobre o volume de código alterado (linhas adicionadas e removidas), o que pode influenciar diretamente a métrica de análise.

Visando obter uma representação mais precisa do esforço envolvido em cada tarefa, os dados foram consolidados por *id*, somando-se o total de linhas de código alteradas (*loc_total*) ao longo das diferentes revisões. Os campos de data de início (*EM_PROGRESSO*), data de finalização (*data_fim*) e tipo de tarefa (*tipo_issue*) foram preservados com base na primeira ocorrência registrada para cada tarefa.

Durante esse processo, duas tarefas foram identificadas com status de bloqueio ou incompletas, por não apresentarem data de finalização. Isso inviabilizava o cálculo da duração da tarefa em horas e, por consequência, do ritmo de alterações no código. Esses registros foram excluídos da análise final.

Com os dados devidamente consolidados e filtrados, foi calculada a métrica de **ritmo de alterações no código** (OLIVEIRA et al., 2017), definida como a razão entre o número total de linhas de código alteradas e o tempo decorrido entre o início e a finalização da tarefa, conforme expresso na Equação 5.1:

$$\text{Ritmo de Alterações (LOC/h)} = \frac{\text{Linhas de Código Alteradas}}{\text{Tempo Gasto (horas)}} \tag{5.1}$$

Essa métrica foi adotada por fornecer uma visão proporcional do volume de trabalho realizado em relação ao tempo despendido, permitindo comparações entre tarefas com diferentes durações e tamanhos de alteração no código-fonte, e servindo como um indicador objetivo do esforço de desenvolvimento.

5.3.4 Ritmo de Alterações por Desenvolvedor

Com base no recorte trimestral definido, a Tabela 20 apresenta o ritmo médio e mediano de alterações no código (LOC/h) por desenvolvedor nos períodos com e sem o uso do GitHub Copilot. Para os desenvolvedores 48 e 72, o período sem Copilot correspondeu aos meses de março, abril e maio de 2024, enquanto para o desenvolvedor 82 foi considerado o trimestre especial de outubro a dezembro de 2024. O período com Copilot correspondeu a março, abril e maio de 2025 para todos.

Tabela 20 – Ritmo de alterações no código (LOC/h) por desenvolvedor e período

ID	Período	Média	Mediana	n
48	Sem Copilot	0,01	0,01	2
48	Com Copilot	4,46	4,46	2
72	Sem Copilot	3779,36	15,27	23
72	Com Copilot	86420,43	36,33	17
82	Sem Copilot	1910,11	1910,11	2
82	Com Copilot	122,21	122,21	2

Para avaliar a significância estatística das diferenças observadas, foram aplicados testes de Mann–Whitney U e a magnitude do efeito foi calculada por meio do *Cliff's Delta*. Os resultados indicaram:

- **Desenvolvedor 48:** U = 0,0; valor-p = 0,667; $\delta = -1{,}00$ (efeito extremo, mas n muito pequeno)
- **Desenvolvedor 72:** U = 178,0; valor-p = 0,642; $\delta = -0{,}09$ (efeito desprezível)
- **Desenvolvedor 82:** U = 3,0; valor-p = 0,667; $\delta = 0{,}50$ (efeito moderado, mas n muito pequeno)

Os resultados sugerem que, embora haja variações expressivas nas médias entre os períodos, o tamanho reduzido das amostras para os desenvolvedores 48 e 82 limita conclusões mais robustas. Para o desenvolvedor 72, que possui maior número de tarefas, não foi identificada diferença estatisticamente significativa entre os períodos.

5.3.5 Ritmo de Alterações Agregado da Equipe

Para avaliar o impacto do GitHub Copilot em uma perspectiva coletiva, foi realizada uma análise agregada do ritmo de alterações no código da equipe, considerando todas as tarefas válidas dos desenvolvedores incluídos no estudo. Para os desenvolvedores 48 e 72, o recorte *sem Copilot* corresponde ao trimestre de março, abril e maio de 2024; já para o desenvolvedor 82, foi considerado o trimestre especial de outubro, novembro e dezembro de 2024. O período *com Copilot* abrange os meses de março, abril e maio de 2025.

A Tabela 21 apresenta os valores médios, medianos do ritmo de alterações (em LOC/h) e o número de tarefas (n) nos dois períodos analisados.

Tabela 21 – Ritmo de alterações no código (LOC/h) da equipe por período (recorte trimestral)

Período	**Média**	**Mediana**	**n**
Sem Copilot	3.360,95	13,93	27
Com Copilot	73.469,81	28,94	20

Observa-se que a média do ritmo de alterações apresentou um crescimento expressivo no período com Copilot. Contudo, nota-se a presença de valores extremos que inflacionam a média, já que a mediana (menos sensível a outliers) mostra uma evolução mais estável, de aproximadamente 13,93 para 28,94 LOC/h.

Para verificar a significância estatística dessa diferença, foi realizado o teste de Mann–Whitney U, cujos resultados foram:

- **U = 246,0**; valor-p = 0,61

Esse resultado indica que não há evidência estatística suficiente para rejeitar a hipótese nula de igualdade entre os dois períodos. A magnitude do efeito, calculada pelo *Cliff's Delta*, foi:

- $\delta = 0,09$ (efeito desprezível)

Assim, embora os valores descritivos sugiram aumento no ritmo de alterações, essa diferença não se mostrou estatisticamente significativa no recorte trimestral agregado da equipe. Isso reforça a importância de análises adicionais por desenvolvedor e de considerar distribuições assimétricas e outliers na interpretação dos resultados.

5.3.6 Análise do Ritmo de Alterações por Tipo de Tarefa

Além das análises realizadas por desenvolvedor e pela equipe como um todo, foi conduzida uma avaliação comparativa do ritmo de alterações no código segundo o tipo

de tarefa executada. As tarefas foram agrupadas em duas categorias principais: **Bugfix**, que reúne atividades originalmente classificadas como *Sustentação - Bug* e *Sustentação - Aprimoramento*, e **Customização**, que corresponde a adaptações de funcionalidades conforme demandas específicas.

A Tabela 22 apresenta os valores médios, medianos e o número de tarefas em cada categoria nos períodos sem e com o uso do Copilot.

Tabela 22 – Ritmo de alterações (LOC/h) por tipo de tarefa

Categoria	**Período**	**Média**	**Mediana**	**n**
Bugfix	Sem Copilot	10,47	5,73	24
Bugfix	Com Copilot	10,84	7,82	10
Customização	Sem Copilot	5,46	4,23	3
Customização	Com Copilot	10,71	6,39	10

No caso das tarefas de **Bugfix**, observa-se que tanto a média quanto a mediana do ritmo de alterações se mantiveram próximas nos dois períodos, sugerindo que o uso do Copilot não trouxe impacto expressivo nesse tipo de atividade.

Para as tarefas de **Customização**, houve aumento da média de 5,46 para 10,71 LOC/h e da mediana de 4,23 para 6,39 LOC/h. No entanto, o número de tarefas sem Copilot nessa categoria foi bastante reduzido (n = 3), o que limita a confiabilidade dessa comparação.

Para avaliar a significância estatística das diferenças observadas, aplicou-se o teste de Mann–Whitney U e calculou-se o *Cliff's Delta*. Os resultados estão descritos a seguir:

- **Bugfix:** U = 117, valor-p = 0,925; $\delta = -0,025$ (efeito desprezível).
- **Customização:** U = 17, valor-p = 0,811; $\delta = 0,13$ (efeito desprezível).

Assim, os resultados indicam que, embora se observe um aumento médio no ritmo de alterações em tarefas de customização, as diferenças entre os períodos sem e com Copilot não foram estatisticamente significativas. Além disso, a magnitude do efeito em ambos os casos foi classificada como desprezível, reforçando a necessidade de cautela na interpretação.

5.3.7 Análise Qualitativa das Percepções sobre o Uso do GitHub Copilot

Com o objetivo de complementar a análise quantitativa do estudo de caso e capturar percepções subjetivas dos desenvolvedores, foi aplicado um questionário (Apêndice D) aos profissionais que participaram da fase com uso do GitHub Copilot. A coleta teve caráter exploratório e visou compreender como os participantes avaliam o uso da ferramenta no

contexto real de trabalho, bem como identificar possíveis impactos cognitivos, operacionais e comportamentais associados à sua adoção.

Os dois primeiros respondentes possuem experiência consolidada tanto em programação quanto na manutenção dos Sistemas Integrados de Gestão (SIG) da UFPI. Um dos desenvolvedores afirmou ter mais de 10 anos de experiência com programação e com os sistemas SIG; o outro, entre 6 e 10 anos de experiência geral em desenvolvimento e entre 3 e 5 anos com os SIG. Ambos declararam já conhecer o GitHub Copilot antes do experimento, embora ainda não tivessem utilizado a ferramenta em atividades práticas.

Do ponto de vista técnico, os participantes relataram que conseguiram instalar a extensão Copilot4Eclipse sem maiores dificuldades. Foi necessário, entretanto, atualizar o Eclipse para a versão 2023-06 (4.28), compatível com a extensão, visto que as versões anteriormente utilizadas não suportavam o plugin. Essa necessidade já havia sido mapeada na fase de preparação e foi gerenciada com apoio da equipe técnica.

Quanto à experiência de uso, os participantes relataram que o Copilot foi de fácil utilização e que contribuiu positivamente para a execução das tarefas. Um dos desenvolvedores classificou o uso da ferramenta como “bastante útil”, enquanto o outro apontou benefícios principalmente em tarefas de customização. Ambos reconheceram que a ferramenta ofereceu sugestões relevantes e que, em diversos momentos, reduziu a necessidade de escrita manual de código.

Sobre os tipos de tarefa mais beneficiados, ambos identificaram impacto mais perceptível nas atividades de *customização*, o que contrasta parcialmente com as análises quantitativas, que apontaram uma redução no ritmo de alterações nesse tipo de tarefa com o uso do Copilot. Tarefas classificadas como *bugfix* pareceram ter menor aproveitamento da ferramenta, segundo os relatos, principalmente em situações que exigiam maior compreensão do sistema legado.

No aspecto comportamental, os participantes indicaram que o Copilot influenciou positivamente sua forma de pensar e resolver problemas. A ferramenta foi descrita como uma espécie de "auxiliar de raciocínio", que estimulava abordagens diferentes ou acelerava a implementação de soluções já concebidas. Apesar disso, um dos desenvolvedores destacou que, mesmo com o Copilot, ainda foi necessário recorrer frequentemente a fóruns como Stack Overflow, especialmente para esclarecer dúvidas sobre bibliotecas antigas ou trechos específicos do sistema.

Em relação à continuidade do uso da ferramenta, ambos demonstraram interesse em utilizá-la no futuro. Entretanto, apenas um deles afirmou que pagaria pela licença individual do Copilot (avaliada em 10 dólares por mês), enquanto o outro respondeu que somente utilizaria a ferramenta caso a instituição arcasse com os custos.

As respostas abertas do questionário forneceram reflexões adicionais. Um dos

participantes comentou que o Copilot “ajuda principalmente na parte chata de codificação”, poupando tempo com estruturas repetitivas ou sintaxe detalhada. O outro destacou que o Copilot "ainda não substitui o conhecimento prévio sobre o sistema", mas pode ser um “bom suporte em tarefas menores”.

Essas percepções qualitativas reforçam que, embora o GitHub Copilot não elimine a necessidade de conhecimento técnico e domínio do sistema, ele é percebido como uma ferramenta útil no cotidiano do desenvolvimento, com maior impacto em tarefas de menor complexidade e maior repetitividade. Tais percepções convergem com os achados quantitativos deste estudo, sobretudo em relação à redução do esforço de desenvolvimento em tarefas específicas.

5.3.8 Discussão dos Resultados do Estudo de Caso

Para complementar a análise quantitativa e compreender os aspectos subjetivos relacionados ao uso do GitHub Copilot, foi aplicado um questionário aos desenvolvedores participantes da etapa com a ferramenta. O objetivo foi capturar percepções sobre utilidade, usabilidade e impactos da ferramenta no cotidiano do trabalho, além de identificar efeitos comportamentais e cognitivos associados à sua adoção.

Os três desenvolvedores que responderam ao questionário são profissionais experientes, com mais de cinco anos de atuação em programação e envolvimento direto com os Sistemas Integrados de Gestão (SIG) da UFPI. Nenhum dos participantes havia utilizado o GitHub Copilot anteriormente em seu ambiente de trabalho, embora já conhecessem a ferramenta. Isso reforça a relevância do estudo como uma primeira experiência formal de adoção em contexto real.

A Tabela 23 apresenta um resumo das respostas às questões fechadas aplicadas no questionário.

Tabela 23 – Resumo das respostas fechadas dos desenvolvedores participantes

Pergunta	**Sim**	**Não**	**Outro**
Já havia utilizado o Copilot antes?	0	3	0
Dificuldade técnica na instalação?	1	2	0
Copilot influenciou sua forma de pensar?	3	0	0
Gostaria de continuar usando?	3	0	0
Pagaria pela licença?	1	2	0

A análise das respostas evidencia uma percepção positiva em relação ao uso da ferramenta. Todos os participantes afirmaram que o Copilot influenciou sua forma de pensar ou resolver problemas, o que sugere que o assistente atua não apenas como um gerador de código, mas como um suporte cognitivo para a organização e estruturação

de ideias durante a programação. Essa percepção é coerente com os relatos abertos, que descrevem a ferramenta como um "auxiliar de raciocínio".

Todos os desenvolvedores também manifestaram interesse em continuar utilizando o Copilot, o que indica uma boa aceitação da ferramenta mesmo diante das limitações técnicas do ambiente utilizado (Eclipse). No entanto, apenas um dos três afirmou que estaria disposto a pagar pela licença mensal, enquanto os outros dois condicionaram seu uso à disponibilização institucional. Isso sugere que, embora a ferramenta seja bem recebida, o custo pode representar uma barreira para adoção contínua em contextos públicos ou com orçamento limitado.

No aspecto técnico, dois participantes relataram dificuldades na instalação e configuração da extensão Copilot4Eclipse, exigindo atualizações de ambiente e suporte adicional. Essa barreira inicial destaca a importância de soluções mais integradas e acessíveis para promover a adoção de ferramentas baseadas em IA no desenvolvimento tradicional em plataformas legadas.

Quanto às respostas abertas, os desenvolvedores relataram que o Copilot ofereceu sugestões úteis, principalmente em tarefas de *customização*, onde estruturas padrão e repetições sintáticas são comuns. Em contrapartida, nas atividades de *bugfix*, a ferramenta mostrou menor desempenho, oferecendo sugestões menos relevantes e exigindo maior esforço manual para adequação ao contexto do sistema.

Em síntese, a análise qualitativa revela que, embora o Copilot não substitua a expertise do desenvolvedor, ele é percebido como um recurso útil e complementar, com maior impacto em tarefas previsíveis e de menor complexidade. A aceitação geral é positiva, mas condicionada a fatores como custo, facilidade de integração e contexto tecnológico. Esses achados reforçam a ideia de que o sucesso da adoção de ferramentas de geração de código assistido por IA depende não apenas de seu desempenho técnico, mas também da percepção de valor por parte dos usuários e da adequação ao ambiente de trabalho.

5.4 Desafios na Condução do Estudo de Caso

A realização do estudo de caso em um ambiente real de desenvolvimento de software implicou em diversos desafios de natureza logística, técnica e institucional, os quais merecem ser destacados para uma compreensão mais ampla da complexidade envolvida nesse tipo de pesquisa aplicada.

O primeiro desafio enfrentado foi a identificação de uma organização disposta a participar do estudo e que apresentasse viabilidade técnica para utilizar a ferramenta GitHub Copilot. Trata-se de uma solução comercial, cuja adoção está condicionada à aquisição de licenças, o que pode representar um obstáculo, sobretudo em instituições

públicas com orçamento restrito. A alternativa encontrada consistiu em realizar o estudo em uma instituição pública de ensino superior, aproveitando o programa educacional do GitHub, que concede acesso gratuito à ferramenta para membros de instituições acadêmicas devidamente credenciadas (GITHUB, 2024).

Mesmo após a escolha da instituição, surgiram questões relacionadas à disponibilidade da equipe técnica. Os desenvolvedores envolvidos acumulavam funções administrativas e técnicas, o que naturalmente impunha limitações de tempo e foco. O pesquisador precisou respeitar o ritmo e as prioridades do setor, ajustando o cronograma de coleta de dados conforme a disponibilidade dos participantes, sem comprometer as atividades rotineiras da equipe.

Do ponto de vista técnico, um obstáculo adicional foi a necessidade de atualização do ambiente de desenvolvimento utilizado pela equipe. A versão do Eclipse em uso não era compatível com a extensão Copilot4Eclipse, exigindo que os participantes realizassem atualizações em suas instalações e reconfigurassem o ambiente de trabalho. Esse processo demandou suporte técnico adicional e acompanhamento contínuo por parte do pesquisador.

Além disso, por se tratar de um estudo observacional, inserido no fluxo de trabalho real da equipe, não foi possível controlar variáveis como a complexidade das tarefas atribuídas ou o nível de motivação individual de cada desenvolvedor. O sucesso da pesquisa dependia, em grande medida, da colaboração espontânea da equipe, tanto na adoção da ferramenta quanto na comunicação de dificuldades e no fornecimento dos dados necessários para análise.

Outro desafio relevante foi a integração dos dados oriundos do sistema de gerenciamento de tarefas com os registros dos repositórios Git. Essa etapa exigiu esforços adicionais de mapeamento e verificação, tendo em vista que nem todas as tarefas apresentavam históricos completos ou padronizados, o que poderia comprometer a consistência dos dados analisados.

Esses desafios evidenciam a complexidade inerente à condução de estudos de caso envolvendo ferramentas tecnológicas emergentes em contextos profissionais. Também reforçam a importância de adotar estratégias adaptativas e manter um diálogo constante com os participantes ao longo de todo o processo de pesquisa, garantindo a viabilidade e a qualidade dos resultados obtidos.

5.5 Lições Aprendidas

A experiência prática de introduzir o GitHub Copilot em um ambiente profissional voltado à manutenção de sistemas legados resultou na identificação de diversas lições que podem orientar iniciativas semelhantes em outras organizações. Essas lições aprendidas

representam um conjunto de recomendações e observações valiosas para pesquisadores, gestores e equipes técnicas interessados em explorar o uso de ferramentas baseadas em inteligência artificial no desenvolvimento de software.

- **A integração técnica é desafiadora:** A adoção de ferramentas baseadas em IA demanda uma infraestrutura compatível. No caso estudado, foi necessário atualizar o ambiente de desenvolvimento (Eclipse) e configurar extensões específicas, o que representou uma barreira inicial significativa e exigiu apoio técnico especializado.

- **A curva de aprendizagem é baixa, mas o julgamento técnico é essencial:** Embora os desenvolvedores tenham se adaptado rapidamente ao uso da ferramenta, sua utilização efetiva exigiu uma avaliação crítica das sugestões geradas e conhecimento prévio sobre a arquitetura dos sistemas envolvidos. A aceitação cega das sugestões pode levar a erros sutis e problemas de manutenção.

- **O Copilot atua como suporte cognitivo:** A ferramenta foi percebida como útil para organizar ideias e iniciar trechos de código, funcionando como um "assistente de raciocínio" nas fases iniciais da implementação. Essa função foi particularmente valorizada em tarefas de escrita de código repetitivo ou em estruturas padrão.

- **A aceitação da ferramenta foi positiva, mas depende de fatores institucionais:** Todos os participantes manifestaram interesse em continuar utilizando o Copilot, entretanto, a manutenção do uso está condicionada à viabilidade financeira e às decisões institucionais relacionadas à aquisição de licenças permanentes.

- **O impacto da ferramenta varia conforme o tipo de tarefa:** A análise dos dados revelou maior eficácia em tarefas repetitivas ou padronizadas, como correções simples de bugs. Em contrapartida, o desempenho da ferramenta foi mais limitado em tarefas complexas de customização, que exigem maior capacidade de abstração e entendimento do domínio.

- **Estudos de caso exigem flexibilidade metodológica:** A realização de pesquisas em ambientes reais demanda adaptações contínuas no cronograma, na coleta e na análise dos dados. A capacidade de ajustar os procedimentos metodológicos às rotinas e às demandas dos participantes foi essencial para a viabilidade do estudo.

Essas lições apontam para o potencial do GitHub Copilot como uma ferramenta de apoio ao desenvolvimento de software, especialmente em contextos que envolvem manutenção de sistemas e tarefas repetitivas. No entanto, ressaltam também a necessidade de contextualização cuidadosa e de preparo técnico das equipes para garantir uma adoção eficaz e responsável da tecnologia.

5.6 Ameaças à Validade

A validade deste estudo pode ser impactada por diversos fatores que devem ser considerados na interpretação dos resultados. As ameaças foram categorizadas em validade interna, externa, de construção e de conclusão.

Validade Interna: A validade interna pode ter sido comprometida pela variabilidade no nível de experiência dos participantes. Na primeira fase do estudo, os participantes eram estudantes com conhecimentos diversos em programação. Embora todos tenham recebido o mesmo treinamento sobre o uso do GitHub Copilot, a familiaridade com a ferramenta e as habilidades prévias em programação podem ter influenciado os resultados. Já na segunda fase, realizada em ambiente profissional, o estudo foi observacional, e o uso do Copilot foi inserido no contexto natural de trabalho. Ainda assim, fatores como a autonomia para decidir quando utilizar a ferramenta e a especialização individual dos desenvolvedores podem ter influenciado os resultados. Além disso, tarefas bloqueadas ou com múltiplas revisões também exigiram cuidados adicionais na consolidação dos dados.

Validade Externa: A generalização dos resultados requer cautela. A primeira fase do estudo, conduzida com estudantes, limita a extrapolação para contextos industriais. Já a segunda fase envolveu desenvolvedores profissionais atuando na manutenção de sistemas legados em uma instituição pública, o que contribui para uma análise mais aplicada. No entanto, por se tratar de uma equipe específica, com tecnologias e fluxos próprios, os resultados podem não se aplicar diretamente a outros ambientes de desenvolvimento, especialmente aqueles com arquiteturas mais modernas ou metodologias diferentes.

Validade de Construção: O estudo utilizou indicadores como tempo de execução das tarefas, linhas de código alteradas e, posteriormente, a razão entre linhas modificadas e tempo (LOC/h). Esses indicadores foram selecionados com base na literatura e na viabilidade de extração automatizada. Entretanto, a ausência de métricas diretas de qualidade de código ou satisfação do usuário limita a abrangência da análise. Na segunda fase, a consolidação de tarefas com múltiplas revisões exigiu decisões metodológicas, como o somatório das alterações por tarefa, o que pode introduzir variações na comparação entre os grupos.

Validade de Conclusão: As análises estatísticas foram realizadas com amostras relativamente pequenas, tanto na fase com estudantes quanto na etapa com desenvolvedores profissionais. Embora tenham sido aplicados testes não paramétricos e medidas de efeito apropriadas, a variabilidade natural das tarefas e a distribuição desigual entre os grupos (com e sem Copilot) podem influenciar nas conclusões. Recomenda-se, portanto, interpretar os resultados com cautela, reconhecendo a natureza exploratória do estudo.

Essas ameaças à validade reforçam a importância de estudos complementares em diferentes contextos organizacionais, tecnológicos e com variados perfis de desenvolvedores, a

fim de ampliar a compreensão sobre o impacto de ferramentas de assistência à programação como o GitHub Copilot.

5.7 Considerações Finais

Este capítulo apresentou os resultados empíricos obtidos a partir de dois contextos distintos: um estudo com estudantes e um estudo de caso com desenvolvedores profissionais. Os achados quantitativos e qualitativos fornecem evidências sobre o impacto do GitHub Copilot no processo de desenvolvimento de software.

No estudo com estudantes, os dados indicaram que o uso do Copilot proporcionou uma redução no tempo necessário para completar tarefas, especialmente entre os participantes de menor experiência (graduação). No entanto, não foram observadas melhorias na qualidade funcional do código, medida pelo número de testes falhos. Isso reforça a ideia de que, embora o Copilot aumente a eficiência temporal, a análise crítica e o conhecimento técnico continuam sendo indispensáveis para assegurar bons resultados.

A análise qualitativa complementou esses achados ao revelar uma percepção positiva dos participantes quanto à facilidade de uso do Copilot e à sua utilidade como apoio ao raciocínio e à codificação. Também foram identificados desafios técnicos e limitações operacionais, como instabilidade e sugestões incoerentes, que reforçam a importância de um uso orientado pedagogicamente. As oficinas ministradas foram bem avaliadas, tanto em formato quanto em conteúdo, contribuindo para a familiarização com as tecnologias envolvidas.

No estudo de caso com desenvolvedores da UFPI, observou-se um aumento expressivo no ritmo de alterações no código com o uso do Copilot, tanto em nível individual quanto coletivo. Os resultados estatísticos indicaram efeitos significativos, especialmente nas tarefas realizadas pelos dois desenvolvedores mais ativos. As percepções dos profissionais confirmaram o potencial da ferramenta em acelerar tarefas rotineiras e sugeriram que o Copilot pode atuar como um apoio complementar no cotidiano do desenvolvimento, contribuindo para a redução do esforço necessário para concluir tarefas.

Esses achados contribuem diretamente para a resposta à terceira questão de pesquisa desta dissertação:

QP3: *Como o uso de ferramentas de geração de código impacta o desempenho de desenvolvedores com diferentes níveis de experiência?*
Resposta: O GitHub Copilot impacta positivamente o desempenho de desenvolvedores em diferentes níveis de experiência. No contexto acadêmico, estudantes menos experientes demonstraram maior eficiência no tempo de execução das tarefas. Já em ambiente profissional, observou-se um aumento no ritmo de alterações no código da equipe. Em ambos os casos, a qualidade funcional do código permaneceu estável, evidenciando que o Copilot pode ser uma ferramenta de apoio eficaz, desde que seu uso seja orientado e acompanhado por supervisão crítica.

Em conjunto, os resultados sugerem que o GitHub Copilot pode ser uma ferramenta promissora no apoio ao desenvolvimento de software, tanto em ambientes educacionais quanto profissionais. No entanto, sua adoção deve vir acompanhada de estratégias didáticas e de capacitação que estimulem o uso consciente e crítico da ferramenta, especialmente para públicos em processo de formação.

6 Conclusões e Trabalhos Futuros

Este capítulo apresenta as principais conclusões da pesquisa realizada, com base nas análises quantitativas e qualitativas conduzidas nas diferentes etapas do estudo. Também são discutidas limitações observadas e propostas direções para trabalhos futuros, visando aprofundar o conhecimento sobre o uso de ferramentas de geração de código assistido por inteligência artificial no desenvolvimento de software.

6.1 Conclusões

Este trabalho investigou o impacto do uso de ferramentas de geração de código assistido por *Inteligência Artificial* (*IA*), com foco na avaliação do *GitHub Copilot*, a partir de uma abordagem metodológica estruturada em quatro etapas: (i) uma fase de investigação, com levantamento da literatura e identificação dos principais desafios e indicadores utilizados na avaliação de ferramentas de geração de código; (ii) uma etapa de coleta de dados, composta por um estudo empírico com estudantes e um estudo de caso com desenvolvedores profissionais; (iii) uma fase de análise, com aplicação de procedimentos quantitativos e qualitativos que permitiram responder às questões de pesquisa e interpretar os efeitos do uso da ferramenta em diferentes contextos de desenvolvimento de software; e (iv) uma fase de sistematização e divulgação dos resultados, incluindo a escrita da dissertação, elaboração de artigos científicos e compartilhamento das evidências geradas com a comunidade acadêmica e profissional.

Com base no levantamento da literatura, foram selecionados dois indicadores principais para análise: o *tempo de execução das tarefas* e o *número de testes falhos*. Esses indicadores se mostraram adequados para avaliar, respectivamente, a eficiência temporal e a correção do código gerado com e sem suporte da ferramenta.

No estudo empírico com estudantes, os dados indicaram que o uso do Copilot resultou em uma redução estatisticamente significativa no tempo necessário para concluir as tarefas de programação. No entanto, não foram observadas diferenças relevantes no número de testes falhos, sugerindo que a qualidade do código, em termos de correção funcional, não foi impactada de forma significativa pela ferramenta. A análise estratificada por turma também mostrou que os estudantes de graduação (especialmente os de níveis mais iniciais) foram os mais beneficiados em termos de tempo, enquanto os estudantes de pós-graduação não apresentaram ganho relevante.

A análise qualitativa complementou esses achados, revelando que o Copilot foi percebido como uma ferramenta de fácil adoção, com baixa curva de aprendizado e impacto

positivo na autoconfiança dos participantes. Ainda assim, foram relatados desafios, como sugestões incoerentes, instabilidade técnica e dificuldades de integração com o ambiente de desenvolvimento.

No estudo de caso com desenvolvedores da Superintendência de Tecnologia da Informação da UFPI, foi observado um aumento estatisticamente significativo no **ritmo de alterações no código (LOC/h)** após a adoção do Copilot, especialmente para os dois principais desenvolvedores da equipe. No entanto, a análise por tipo de tarefa revelou que os ganhos foram mais expressivos em atividades do tipo *bugfix*, enquanto tarefas de *customização* apresentaram redução no ritmo de alterações, possivelmente devido à necessidade de maior adaptação do código sugerido às demandas específicas.

De modo geral, os achados deste trabalho apontam que o Copilot pode ser um aliado na **redução do esforço de desenvolvimento** em determinadas tarefas, desde que seu uso seja acompanhado por senso crítico e domínio técnico por parte do desenvolvedor. A ferramenta não elimina a necessidade de supervisão humana, mas pode contribuir positivamente, especialmente em contextos educacionais ou em tarefas de menor complexidade.

As três questões de pesquisa propostas ao longo da metodologia foram respondidas ao longo deste trabalho. A **QP1**, relacionada às oportunidades e desafios do uso de ferramentas de geração de código assistido por IA, foi respondida com base na revisão da literatura, que identificou benefícios como aumento de eficiência e acessibilidade, bem como limitações relacionadas à qualidade das sugestões e à necessidade de supervisão humana. A **QP2**, que buscava identificar os principais indicadores para avaliação dessas ferramentas, resultou na seleção de dois indicadores centrais: *tempo de execução das tarefas* e *número de testes falhos*, além da análise de percepções qualitativas. Por fim, a **QP3**, que investigou o impacto do GitHub Copilot em contextos acadêmico e profissional, foi respondida por meio da análise dos estudos realizados, que demonstraram ganhos de eficiência (redução de tempo e aumento no ritmo de alterações no código) e percepções majoritariamente positivas, ainda que sem impacto significativo na qualidade funcional do código.

6.2 Trabalhos Futuros

A continuidade desta pesquisa prevê o aprofundamento da análise qualitativa com base em dados já coletados, como notas obtidas nas disciplinas, histórico de rendimento acadêmico e respostas abertas dos estudantes sobre o uso do Copilot. Essas informações permitirão investigar, por exemplo, se o nível de desempenho acadêmico influenciou o aproveitamento da ferramenta ou a percepção de seus benefícios.

Além disso, será possível ampliar o estudo de caso com o envolvimento de mais desenvolvedores e tarefas em contextos variados, o que pode trazer mais robustez estatística à análise e permitir a identificação de padrões mais generalizáveis sobre o uso de ferramentas

baseadas em IA no desenvolvimento de software.

Por fim, estudos futuros podem ampliar a investigação com novos perfis de usuários, tarefas mais variadas e contextos tecnológicos distintos (como desenvolvimento front-end, testes automatizados ou engenharia de dados), contribuindo para uma compreensão mais abrangente sobre o papel da IA generativa no processo de codificação.

Referências

BAKAL, G. et al. *Experience with GitHub Copilot for Developer Productivity at Zoominfo*. 2025. Disponível em: <https://arxiv.org/abs/2501.13282>. Citado 3 vezes nas páginas 15, 20 e 26.

BARDIN, L. *Análise de Conteúdo*. Lisboa: Edições 70, 1977. Citado na página 50.

BATISTA, S. et al. Code on demand: A comparative analysis of the efficiency, understandability, and self-correction capability of copilot, chatgpt, and gemini. In: *Anais do Simpósio Brasileiro de Qualidade de Software (SBQS)*. Brasil: SBC, 2024. Citado 2 vezes nas páginas 9 e 12.

BROWN, T. B. et al. *Language Models are Few-Shot Learners*. 2020. Disponível em: <https://arxiv.org/abs/2005.14165>. Citado na página 8.

CHARNESS, G.; GNEEZY, U.; KUHN, M. A. Experimental methods: Between-subject and within-subject design. *Journal of Economic Behavior & Organization*, v. 81, n. 1, p. 1–8, 2012. ISSN 0167-2681. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0167268111002289>. Citado na página 43.

CHEN, M. et al. Evaluating large language models trained on code. Disponível em <https://arxiv.org/abs/2107.03374>. 2021. Disponível em: <https://arxiv.org/abs/2107.03374>. Citado 3 vezes nas páginas 2, 8 e 9.

CORSO, V. et al. Assessing ai-based code assistants in method generation tasks. In: *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*. New York, NY, USA: Association for Computing Machinery, 2024. (ICSE-Companion '24), p. 380–381. ISBN 9798400705021. Disponível em: <https://doi.org/10.1145/3639478.3643122>. Citado 6 vezes nas páginas 9, 10, 11, 16, 21 e 26.

DAKHEL, A. M. et al. Github copilot ai pair programmer: Asset or liability? *Journal of Systems and Software*, v. 203, p. 111734, 2023. ISSN 0164-1212. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0164121223001292>. Citado 8 vezes nas páginas 2, 8, 10, 11, 18, 22, 24 e 26.

DAVILA, N. et al. An industry case study on adoption of ai-based programming assistants. In: *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice*. New York, NY, USA: Association for Computing Machinery, 2024. (ICSE-SEIP '24), p. 92–102. ISBN 9798400705014. Disponível em: <https://doi.org/10.1145/3639477.3643648>. Citado 4 vezes nas páginas 11, 16, 21 e 26.

GITHUB. Introducing github copilot: your ai pair programmer. *GitHub Blog*, 2021. Available at: <https://github.blog/2021-06-29-introducing-github-copilot-ai-pair-programmer/>. Citado 2 vezes nas páginas 2 e 8.

GitHub. *GitHub Copilot – November 30th Update*. 2023. <https://github.blog/changelog/2023-11-30-github-copilot-november-30th-update/>. Accessed: December 4, 2024. Citado na página 8.

GITHUB. *GitHub Education*. 2024. <https://github.com/education>. Acesso em: 9 nov. 2024. Citado 3 vezes nas páginas 31, 32 e 64.

GU, X. et al. On the effectiveness of large language models in domain-specific code generation. *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, out. 2024. ISSN 1049-331X. Just Accepted. Disponível em: <https://doi.org/10.1145/3697012>. Citado 3 vezes nas páginas 16, 21 e 23.

HERBSLEB, J. D.; MOITRA, D. Global software development. *IEEE software*, IEEE, v. 18, n. 2, p. 16–20, 2001. Citado na página 1.

HUSEIN, R. A.; ABURAJOUH, H.; CATAL, C. Large language models for code completion: A systematic literature review. *Computer Standards & Interfaces*, v. 92, p. 103917, 2025. ISSN 0920-5489. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0920548924000862>. Citado 2 vezes nas páginas 7 e 8.

IMAI, S. Is github copilot a substitute for human pair-programming? an empirical study. In: . New York, NY, USA: Association for Computing Machinery, 2022. (ICSE '22), p. 319–321. ISBN 9781450392235. Disponível em: <https://doi.org/10.1145/3510454.3522684>. Citado 2 vezes nas páginas 19 e 25.

KAZEMITABAAR, M. et al. Studying the effect of ai code generators on supporting novice learners in introductory programming. In: *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: Association for Computing Machinery, 2023. (CHI '23). ISBN 9781450394215. Disponível em: <https://doi.org/10.1145/3544548.3580919>. Citado 6 vezes nas páginas 10, 11, 18, 22, 24 e 26.

KITCHENHAM, B. et al. Robust statistical methods for empirical software engineering. *Empirical Software Engineering*, v. 22, n. 2, p. 579–630, 2017. ISSN 1573-7616. Disponível em: <https://doi.org/10.1007/s10664-016-9437-5>. Citado na página 43.

KITCHIN, R.; DODGE, M. *Code/space: Software and everyday life*. [S.l.]: Mit Press, 2014. Citado na página 1.

KOYANAGI, K. et al. Exploring the effect of multiple natural languages on code suggestion using github copilot. In: *2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR)*. [S.l.: s.n.], 2024. p. 481–486. Citado 3 vezes nas páginas 16, 21 e 23.

LEE, Y. Y. et al. Temporal code completion and navigation. In: *2013 35th International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2013. p. 1181–1184. Citado na página 7.

LEVINE, J. R.; MASON, T.; BROWN, D. *Lex & yacc*. [S.l.]: "O'Reilly Media, Inc.", 1992. Citado na página 7.

LIKERT, R. A technique for the measurement of attitudes. *Archives of Psychology*, v. 22, n. 140, p. 5–55, 1932. Citado 2 vezes nas páginas 48 e 49.

LIRA, W.; NETO, P. S.; OSORIO, L. Uma análise do uso de ferramentas de geração de código por alunos de computação. In: *Anais do IV Simpósio Brasileiro de Educação em Computação*. Porto Alegre, RS, Brasil: SBC, 2024. p. 63–71. ISSN 0000-0000. Disponível

em: <https://sol.sbc.org.br/index.php/educomp/article/view/28174>. Citado 2 vezes nas páginas 2 e 8.

LIU, Y. et al. Refining chatgpt-generated code: Characterizing and mitigating code quality issues. *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, v. 33, n. 5, jun. 2024. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/3643674>. Citado 3 vezes nas páginas 17, 21 e 23.

MICROSOFT. *IntelliCode*. 2024. <https://learn.microsoft.com/en-us/visualstudio/ide/using-intellisense>. Acessado: 19 de out. de 2024. Citado na página 7.

MICROSOFT. *Visual Studio Code - Code Editing. Redefined*. 2024. <https://code.visualstudio.com/>. Acessado: 19 de out. de 2024. Citado na página 7.

MONTEIRO, M. et al. Nocodegpt: A no-code interface for building web apps with language models. *Software: Practice and Experience*, v. 55, n. 8, p. 1408–1424, 2025. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3432>. Citado na página 2.

MOON, J. et al. chatgpt vs mentor : Programming language learning assistance system for beginners. In: *2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS)*. Penang, Malaysia: 2023 IEEE 8th International Conference On Software Engineering and Computer Systems (ICSECS), 2023. p. 106–110. Citado na página 2.

NEJJAR, M. et al. Llms for science: Usage for code generation and data analysis. *Journal of Software: Evolution and Process*, n/a, n. n/a, p. e2723, 2024. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2723>. Citado 5 vezes nas páginas 10, 11, 17, 22 e 24.

NGUYEN, N.; NADI, S. An empirical evaluation of github copilot's code suggestions. In: *Proceedings of the 19th International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2022. (MSR '22), p. 1–5. ISBN 9781450393034. Disponível em: <https://doi.org/10.1145/3524842.3528470>. Citado 6 vezes nas páginas 3, 9, 10, 19, 25 e 26.

NGWENYAMA, O.; KANITA, N.; ROWE, F. Can generative ai contribute to both productivity gains and human flourishing, and in fine satisfaction at work? research on github copilot use in software development. 2025. Citado 3 vezes nas páginas 15, 20 e 26.

OLIVEIRA, E. et al. How have software engineering researchers been measuring software productivity? - a systematic mapping study. In: INSTICC. *Proceedings of the 19th International Conference on Enterprise Information Systems - Volume 2: ICEIS,*. [S.l.]: SciTePress, 2017. p. 76–87. ISBN 978-989-758-248-6. Citado na página 57.

ORTIN, F. et al. An empirical evaluation of lex/yacc and antlr parser generation tools. *PLOS ONE*, Public Library of Science, v. 17, n. 3, p. 1–16, 03 2022. Disponível em: <https://doi.org/10.1371/journal.pone.0264326>. Citado na página 7.

OSORIO, F. *Associacao API*. 2024. <https://github.com/fernandoosorio/associacao-api-artigo>. Citado na página 33.

OSORIO, F. *Locadora API*. 2024. <https://github.com/fernandoosorio/locadora-api-artigo>. Citado na página 33.

OSORIO, F. *Dataset do Projeto de Mestrado: Impacto de Ferramentas de Geração de Código Assistido por IA*. 2025. <https://github.com/fernandoosorio/dataset-mestrado>. Disponível em repositório GitHub. Citado 2 vezes nas páginas 39 e 56.

PENG, S. et al. *The Impact of AI on Developer Productivity: Evidence from GitHub Copilot*. 2023. Disponível em: <https://arxiv.org/abs/2302.06590>. Citado 9 vezes nas páginas 1, 2, 3, 8, 10, 18, 22, 24 e 52.

PRATHER, J. et al. "it's weird that it knows what i want": Usability and interactions with copilot for novice programmers. *ACM Trans. Comput.-Hum. Interact.*, Association for Computing Machinery, New York, NY, USA, v. 31, n. 1, nov. 2023. ISSN 1073-0516. Disponível em: <https://doi.org/10.1145/3617367>. Citado 6 vezes nas páginas 8, 11, 18, 22, 24 e 26.

PRESSMAN, R. *Software Engineering: A Practitioner's Approach*. 7. ed. USA: McGraw-Hill, Inc., 2009. ISBN 0073375977. Citado na página 1.

PURYEAR, B.; SPRINT, G. Github copilot in the classroom: Learning to code with ai assistance. *J. Comput. Sci. Coll.*, Consortium for Computing Sciences in Colleges, Evansville, IN, USA, v. 38, n. 1, p. 37–47, nov 2022. ISSN 1937-4771. Citado 2 vezes nas páginas 2 e 11.

SHAH, V. Towards efficient software engineering in the era of ai and ml: Best practices and challenges. *International Journal of Computer Science and Technology*, v. 3, n. 3, p. 63–78, 2019. Citado na página 1.

SHULL, F.; SINGER, J.; SJØBERG, D. I. *Guide to advanced empirical software engineering*. [S.l.]: Springer, 2008. v. 93. Citado na página 12.

SOMMERVILLE, I. *Software Engineering*. Pearson, 2016. (Always learning). ISBN 9780133943030. Disponível em: <https://books.google.com.br/books?id=tW4VngEACAAJ>. Citado na página 1.

STRAY, V. et al. Generative ai and developer workflows: How github copilot and chatgpt influence solo and pair programming. 2025. Citado 3 vezes nas páginas 15, 20 e 26.

VASWANI, A. et al. *Attention Is All You Need*. 2023. Citado 2 vezes nas páginas 2 e 8.

WERMELINGER, M. Using github copilot to solve simple programming problems. In: *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. New York, NY, USA: Association for Computing Machinery, 2023. (SIGCSE 2023), p. 172–178. ISBN 9781450394314. Disponível em: <https://doi.org/10.1145/3545945.3569830>. Citado 2 vezes nas páginas 19 e 25.

WIERINGA, R.; MORALI, A. Technical action research as a validation method in information systems design science. In: *International Conference on Design Science Research in Information Systems*. Berlin, Heidelberg: Springer, 2012. p. 220–238. Citado na página 29.

WOHLIN, C.; AURUM, A. Towards a decision-making structure for selecting a research design in empirical software engineering. *Empirical Softw. Engg.*, Kluwer Academic Publishers, USA, v. 20, n. 6, p. 1427–1455, dez. 2015. ISSN 1382-3256. Disponível em: <https://doi.org/10.1007/s10664-014-9319-7>. Citado 2 vezes nas páginas 4 e 12.

XUE, H.; NIU, Y. Exercise generation and student cognitive ability research based on chatgpt and rasch model. *IEEE Access*, v. 11, p. 1–1, 2023. Citado na página 2.

YETIŞTIREN, B. et al. *Evaluating the Code Quality of AI-Assisted Code Generation Tools: An Empirical Study on GitHub Copilot, Amazon CodeWhisperer, and ChatGPT*. 2023. Disponível em: <https://arxiv.org/abs/2304.10778>. Citado 7 vezes nas páginas 10, 11, 18, 23, 25, 26 e 52.

ZHANG, L. et al. Empirical research in software engineering — a literature survey. *Journal of Computer Science and Technology*, v. 33, n. 5, p. 876–899, September 2018. ISSN 1860-4749. Disponível em: <https://doi.org/10.1007/s11390-018-1864-x>. Citado na página 12.

Apêndices

APÊNDICE A – Questionário Solução Desenvolvida

Q1. Identificador:

Q2. Problema resolvido:

- ☐ #1
- ☐ #2
- ☐ #3
- ☐ #4

Q3. Horário de início:

Q4. Horário de Conclusão:

Q5. Você utilizou Copilot para gerar a solução para essa questão?

- ☐ Sim
- ☐ Não

Q6. Em caso de uso do Copilot, anexar o chat.

Q7. Enviar apenas o arquivo modificado.

Q8. Enviar vídeo da solucionando o problema.

APÊNDICE B – Questionário Final

Q1. Identificador:

Q2. Turma:

- ☐ Pós-graduação
- ☐ Engenharia de Software
- ☐ Programação Orientada a Objetos

Q3. IRA - Índice de Rendimento Acadêmico:

Q4. Idade:

Q5. Gênero?

- ☐ Masculino
- ☐ Feminino
- ☐ Outro

Q6. A quanto tempo está envolvido com programação:

- ☐ menos de 1 ano (ou não tem experiência com programação)
- ☐ 1 ano
- ☐ 2 anos
- ☐ 3 anos
- ☐ de 4 a 5 anos
- ☐ mais de 5 anos

Q7. Quais linguagens de programação você utiliza com mais frequência?

- ☐ Python
- ☐ Java
- ☐ JavaScript
- ☐ C/C++
- ☐ PHP
- ☐ Ruby
- ☐ Go

☐ Swift

☐ Dart

☐ Outras

Q8. Em uma escala de 1 a 5, onde 1 é "Iniciante"e 5 é "Avançado", como você classificaria suas habilidades de programação?

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

Q9. Você já passou por treinamentos formais ou possui certificações relacionadas à programação? Se sim, descreva brevemente.

Q10. Em uma escala de 1 a 5, onde 1 é "Iniciante"e 5 é "Avançado", como você descreveria seu nível de conhecimento relacionado ao uso de ferramentas de IA (Chat-GPT, GitHub Copilot) na programação?

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

Q11. Você já utilizou o GitHub Copilot anteriormente? (antes do início das oficinas de capacitação)

☐ Sim

☐ Não

Q12. Caso tenha respondido sim para a pergunta anterior, descreva sua experiência.

Q13. Por que você decidiu participar deste estudo?

Q14. Quais são suas expectativas em relação ao uso das ferramentas de IA citadas neste experimento?

Q15. Há alguma informação adicional que você gostaria de compartilhar conosco em relação à sua experiência, habilidades ou motivação?

Q16. Você utilizava Visual Studio Code antes do inicio das oficinas?

- ☐ Sim
- ☐ Não

Q17. Você já utilizava o framework Spring ou Spring Boot antes do inicio das oficinas?

- ☐ Sim
- ☐ Não

Q18. Classifique sua satisfação geral com o uso do GitHub Copilot durante o experimento:

- ☐ Muito Insatisfeito
- ☐ Insatisfeito
- ☐ Neutro
- ☐ Satisfeito
- ☐ Muito Satisfeito

Q19. Como você avalia a utilidade das sugestões de código fornecidas pelo GitHub Copilot?

- ☐ Não foram úteis
- ☐ Pouco úteis
- ☐ Neutro
- ☐ Úteis
- ☐ Muito úteis

Q20. Em termos de velocidade de desenvolvimento, você sentiu que o GitHub Copilot agilizou seu trabalho?

- ☐ Não agilizou
- ☐ Agilizou um pouco
- ☐ Neutro
- ☐ Agilizou
- ☐ Agilizou muito

Q21. Na sua opinião, o uso do GitHub Copilot afetou a qualidade do código produzido?

- ☐ A qualidade piorou
- ☐ A qualidade não foi afetada

☐ A qualidade melhorou

☐ A qualidade melhorou muito

Q22. Você precisou realizar algum ajuste no código gerado pelo GitHub Copilot antes de enviar sua resposta?

☐ Sim

☐ Não

Q23. Você encontrou alguma dificuldade ou limitação ao usar o GitHub Copilot?

☐ Sim

☐ Não

Q24. Caso tenha respondido sim para a pergunta anterior, descreva sua dificuldade.

Q25. Em relação ao contexto específico do domínio de desenvolvimento de software estudado, você sentiu que o GitHub Copilot ofereceu sugestões de código relevantes e apropriadas?

☐ Sim

☐ Não

Q26. Com base na sua experiência durante o experimento, você acredita que o GitHub Copilot pode ser uma ferramenta útil no seu trabalho diário como desenvolvedor(a) de software?

☐ Sim

☐ Não

Q27. Comente sobre quaisquer outros aspectos relevantes sobre o uso do GitHub Copilot que você gostaria de compartilhar.

Q28. Você sentiria mais confiante de fazer manutenção de um sistema, se tivesse disponível a ferramenta Copilot?

☐ Sim

☐ Não

Q29. Se você respondeu Não à pergunta anterior, justifique.

APÊNDICE C – Trechos do Código a Serem Recriados

```
//Este é o problema 1
public Aluguel registrarAluguel(String placa, Date data, int dias, Long cpf) throws
VeiculoNaoCadastrado, VeiculoAlugado, ClienteNaoCadastrado {
        // Verificar se o veículo está cadastrado
        Veiculo veiculo = this.getVeiculoByPlaca(placa);
        // Verificar se o veículo está alugado
        if( aluguelRepository.findFirstByVeiculoPlacaAndBaixoFalse(placa) != null)
        throw new VeiculoAlugado();

        // Verificar se o cliente está cadastrado
        Cliente cliente = Optional.ofNullable( this.clienteRepository.findByCpf(cpf) )
        .orElseThrow( () -> new ClienteNaoCadastrado() );

        // Calcular a data de término do aluguel
        Date dataFim = new Date(data.getTime() + (dias * 24 * 60 * 60 * 1000L));
        //converter a data e dataFim para LocalDateTime
        LocalDateTime dataInicioLocalDateTime =
        data.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime();
        LocalDateTime dataFimLocalDateTime =
        dataFim.toInstant().atZone(ZoneId.systemDefault()).toLocalDateTime();
        // Calcular o valor total do aluguel
        double totalAluguel = veiculo.aluguel(dias);

        // Criar uma nova instância de Aluguel
        Aluguel aluguel = new Aluguel(cliente, veiculo, dataInicioLocalDateTime,
        dataFimLocalDateTime, totalAluguel);

        aluguelRepository.save(aluguel);

        // Retornar a instância de Aluguel
        return aluguel;


}
```

Figura 6 – Problema #1.
Fonte: autor.

```
//Este é o problema 002
public List<Veiculo> aumentarDiaria(int tipo, double taxaAumento) throws
TaxaAumentoForaIntervalo, TipoVeiculoDesconhecido{
        // Verificar se a taxaAumento está dentro do intervalo permitido
        if (taxaAumento <= 0 || taxaAumento > 100) {
        throw new TaxaAumentoForaIntervalo();
        }
        // Obter a classe do tipo de veículo
        Class<?> classe = TipoVeiculoClassHelper.getTipoVeiculoClass(tipo);
        if (classe == null) {
        throw new TipoVeiculoDesconhecido();
        }
        // Buscar todos os veículos do tipo especificado
        List<Veiculo> veiculos = veiculoRepository.findAllVeiculosByType(classe);
        // Atualizar a diária de cada veículo
        for (Veiculo veiculo : veiculos) {
        veiculo.setValorDiariaAntigo(veiculo.getValorDiaria());
        veiculo.aumentarDiaria(taxaAumento);
        }
        // Retornar a lista de veículos atualizados
        return veiculos;

}
```

Figura 7 – Problema #2.
Fonte: autor.

```
//Este é o problema: **ID Questão = 003**
public Double calcularTotalDeTaxas(int numAssociacao, int vigencia) throws
AssociacaoNaoExistente, TaxaNaoExistente {

    if (!associacaoRepository.existsByNumero(numAssociacao)) {
    throw new AssociacaoNaoExistente();
    }
    Associacao associacao = associacaoRepository.findByNumero(numAssociacao);
    if (!taxaRepository.existsByAssociacaoAndVigencia(associacao, vigencia)) {
    throw new TaxaNaoExistente();
    }
    List<Taxa> taxas = taxaRepository.findByAssociacaoAndVigencia(associacao,
    vigencia);
    return taxas.stream().mapToDouble(Taxa::getValorAno).sum();


}
```

Figura 8 – Problema #3.
Fonte: autor.

```java
//Este é o problema: **ID Questão = 004**
public Pagamento registrarPagamento(int numAssociacao, String nomeTaxa, int vigencia,
int numAssociado, LocalDate data,double valor) throws AssociacaoNaoExistente,
AssociadoNaoExistente, AssociadoJaRemido, TaxaNaoExistente, ValorInvalido {
    // Verificar se a associação existe
    if (!associacaoRepository.existsByNumero(numAssociacao)) {
    throw new AssociacaoNaoExistente();
    }
    // Verificar se o associado existe
    if (!associadoRepository.existsByNumero(numAssociado)) {
    throw new AssociadoNaoExistente();
    }
    // Verificar se o associado é remido
    Associado associado = associadoRepository.findByNumero(numAssociado);
    if (associadoRepository.existsByNumeroAndRemido(numAssociado, true)) {
    throw new AssociadoJaRemido();
    }

    // Verificar se a taxa existe para a associação e vigência
    Associacao associacao = associacaoRepository.findByNumero(numAssociacao);
    Taxa taxa = taxaRepository.findByAssociacaoAndNomeAndVigencia(associacao,
    nomeTaxa, vigencia);
    if (taxa == null) {
    throw new TaxaNaoExistente();
    }

    // Verificar se o valor é válido
    double valorMinimo = taxa.getValorAno() / taxa.getParcelas();
    double valorMaximo = taxa.getValorAno();
    if (valor < valorMinimo || valor > valorMaximo || valor < 0) {
    throw new ValorInvalido();
    }
    // Registrar o pagamento
    Pagamento pagamento = new Pagamento();
    pagamento.setAssociacao(associacao);
    pagamento.setAssociado(associado);
    pagamento.setTaxa(taxa);
    pagamento.setData(data);
    pagamento.setValor(valor);
    pagamentoRepository.save(pagamento);
    return pagamento;

}
```

Figura 9 – Problema #4.
Fonte: autor.

APÊNDICE D – Questionário aplicado no Estudo de Caso com Desenvolvedores Profissionais

Questionário de Impacto sobre o uso do GitHub Copilot

Você está sendo convidado(a) a participar de uma pesquisa que busca analisar a experiência prática de desenvolvedores de software com o uso da ferramenta GitHub Copilot, com ênfase em aspectos como produtividade, usabilidade e percepção geral.

Sua participação consiste em responder a um breve questionário com perguntas objetivas e/ou abertas. A participação é voluntária e você pode optar por não responder a alguma pergunta ou interromper sua colaboração a qualquer momento, sem prejuízo algum.

As informações coletadas serão utilizadas exclusivamente para fins acadêmicos, com garantia de sigilo e anonimato. Nenhuma informação que permita sua identificação será divulgada nos resultados da pesquisa.

Em caso de dúvidas, você poderá entrar em contato com o pesquisador responsável pelo e-mail: `fernando.mendes@ufpi.edu.br`.

Caracterização do Participante

1. Qual seu tempo total de experiência com programação?
   ( ) Menos de 1 ano ( ) De 1 a 3 anos ( ) De 3 a 5 anos ( ) Mais de 5 anos

2. Há quanto tempo você atua no desenvolvimento dos sistemas SIG da instituição?
   ( ) Menos de 6 meses ( ) De 6 meses a 1 ano ( ) De 1 a 3 anos ( ) Mais de 3 anos

3. Você já havia utilizado o GitHub Copilot ou ferramentas similares antes desta pesquisa?
   ( ) Sim ( ) Não

4. Se respondeu “Sim” à questão anterior, informe quais ferramentas já utilizou:
   _____

Instalação e Configuração do Ambiente

5. Você enfrentou dificuldades técnicas na instalação ou uso do Copilot no Eclipse?
   ( ) Não tive dificuldades ( ) Sim, precisei atualizar o Eclipse ( ) Sim, outros problemas técnicos

6. Caso tenha enfrentado problemas, descreva-os e explique como solucionou:

   _____

7. Como foi sua experiência inicial com o GitHub Copilot?
   ( ) Muito fácil ( ) Relativamente fácil ( ) Tive dificuldades ( ) Não consegui usar efetivamente

8. O Copilot contribuiu para sua produtividade nas tarefas?
   ( ) Aumentou muito ( ) Aumentou um pouco ( ) Não fez diferença ( ) Reduziu a produtividade

9. Em quais tipos de tarefa o Copilot foi mais útil para você? (marcar mais de uma)
   ☐ Customização ☐ Correção de bugs ☐ Aprimoramentos ☐ Escrita de testes ☐ Nenhum

10. Você notou melhor desempenho do Copilot em tarefas de Customização do que em Bugfix?
    ( ) Sim ( ) Não

11. Comente a resposta anterior:

    _____

Uso e Revisão das Sugestões

12. Com que frequência você revisava ou corrigia sugestões do Copilot?
    ( ) Nunca ( ) Quase nunca ( ) Às vezes ( ) Frequentemente ( ) Sempre

13. Durante o uso do Copilot, você recorreu a fóruns ou documentação online?
    ( ) Menos do que o habitual ( ) Mesma frequência ( ) Mais do que o habitual ( ) Não costumo usar

Impacto Cognitivo e Percepção Técnica

14. O uso do Copilot influenciou sua forma de pensar ou resolver problemas?
    ( ) Sim, ajudou a pensar de forma mais estruturada ( ) Sim, mas gerou dependência ( ) Não influenciou

15. Você aprendeu algo novo com o Copilot durante o período de uso?
    ( ) Sim, frequentemente ( ) Sim, ocasionalmente ( ) Pouco ou nada

16. Você gostaria de continuar usando o Copilot em sua rotina?
    ( ) Sim, com certeza ( ) Talvez, dependendo da tarefa ( ) Não vejo necessidade

17. Você pagaria pela licença do GitHub Copilot (aproximadamente US$10/mês)?
    ( ) Sim ( ) Talvez, dependendo do uso ( ) Não

18. Deseja deixar alguma observação adicional sobre o uso do Copilot?

    _____

APÊNDICE E – TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO.

Prezado(a) Senhor (a),

Você está sendo convidado(a) a participar como voluntário(a) de uma pesquisa denominada **IMPACTO DE FERRAMENTAS DE GERAÇÃO DE CÓDIGO ASSISTIDO POR IA NO DESENVOLVIMENTO DE SOFTWARE: UMA ABORDAGEM EMPÍRICA**, Esta pesquisa está sob a responsabilidade do pesquisador **LUIZ FERNANDO MENDES OSORIO, discente do PPGCC/UFPI,** e tem como objetivo: **avaliar se as ferramentas de geração de código efetivamente auxiliam os desenvolvedores a programar de forma mais eficaz.**

Neste sentido, solicitamos sua colaboração mediante a assinatura deste termo. Este documento, chamado Termo de Consentimento Livre e Esclarecido (TCLE), visa assegurar seus direitos como participante. Após seu consentimento, assine todas as páginas e ao final desse documento que está em duas vias. O mesmo também será assinado pelo pesquisador em todas as páginas, ficando uma via com você participante da pesquisa e outra com o pesquisador. Por favor, leia com atenção e calma, aproveite para esclarecer todas as suas dúvidas. Se houver perguntas antes ou mesmo depois de indicar sua concordância, você poderá esclarecê-las com o pesquisador responsável pela pesquisa através dos seguintes telefones: 86-9 9402-8055 / 86 9 99512317.

Se mesmo assim, as dúvidas ainda persistirem você pode entrar em contato com o Comitê de Ética em Pesquisa da UFPI, que acompanha e analisa as pesquisas científicas que envolvem seres humanos, no Campus Universitário Ministro Petrônio Portella, Bairro Ininga, Teresina - PI, telefone (86) 2222-4824, e-mail: cep.ufpi@ufpi.edu.br; no horário de atendimento ao público, segunda a sexta, manhã: 08h00 às 12h00 e a tarde: 14h00 às 18h00. Se preferir, pode levar este Termo para casa e consultar seus familiares ou outras pessoas antes de decidir participar. Esclarecemos mais uma vez que sua participação é voluntaria, caso decida não participar ou retirar seu consentimento a qualquer momento da

pesquisa, não haverá nenhum tipo de penalização ou prejuízo e o pesquisador estará a sua disposição para qualquer esclarecimento.

A pesquisa tem como justificativa: **avaliar empiricamente a eficácia das ferramentas de geração de código. Através de estudos empíricos, será possível identificar de que maneira essas ferramentas influenciam o desempenho de desenvolvedores com diferentes níveis de experiência** e para a realização desta pesquisa, será adotado o seguinte procedimento de coleta de dados: a instalação do plugin Copilot no ambiente de desenvolvimento Eclipse, permitindo a observação de seu impacto no fluxo de trabalho dos participantes.

Esclareço que esta pesquisa acarreta os seguintes riscos:

- A instalação e uso do plugin Copilot no Eclipse podem alterar a dinâmica usual de desenvolvimento do participante, o que pode gerar uma leve curva de aprendizado e possíveis distrações no início da experiência. Para minimizar esse impacto, o participante poderá interromper sua participação a qualquer momento caso sinta que sua produtividade está sendo comprometida.
- Como o Copilot é uma ferramenta baseada em inteligência artificial, ele pode gerar sugestões de código que não estejam em conformidade com as melhores práticas de programação ou que contenham erros lógicos e de segurança. Para contornar esse risco, recomenda-se que os participantes avaliem criticamente as sugestões geradas e utilizem seu julgamento técnico antes de implementá-las.
- O Copilot pode utilizar fragmentos de código de fontes públicas para sugerir implementações, o que pode levar à exposição acidental de trechos de código sensíveis caso o participante esteja lidando com informações restritas. Para mitigar esse risco, orientamos que os participantes evitem inserir dados sensíveis ou confidenciais durante o uso da ferramenta. Além disso, todas as informações coletadas na pesquisa serão tratadas de forma anônima, garantindo o sigilo dos participantes.

- Há um risco de que, ao longo do uso contínuo do Copilot, alguns participantes se tornem excessivamente dependentes da ferramenta para a escrita de código, o que pode afetar seu desenvolvimento autônomo de habilidades de programação. Esse risco será minimizado pelo fato de que a pesquisa não interfere diretamente nas decisões dos participantes e que eles podem continuar utilizando seu próprio julgamento crítico ao programar.

Os resultados obtidos nesta pesquisa serão utilizados para fins acadêmico-científicos (divulgação em revistas e em eventos científicos) e os pesquisadores se comprometem a manter o sigilo e identidade anônima, como estabelecem as Resoluções do Conselho Nacional de Saúde nº. 466/2012 e 510/2016 e a Norma Operacional 01 de 2013 do Conselho Nacional de Saúde, que tratam de normas regulamentadoras de pesquisas que envolvem seres humanos. E você terá livre acesso a todas as informações e esclarecimentos adicionais sobre o estudo, bem como lhe é garantido acesso a seus resultados.

Esclareço ainda que você não terá nenhum custo com a pesquisa, e caso haja por qualquer motivo, asseguramos que você será devidamente ressarcido. Não haverá nenhum tipo de pagamento por sua participação, ela é voluntária. Caso ocorra algum dano comprovadamente decorrente de sua participação neste estudo você poderá ser indenizado conforme determina a Resolução 466/12 do Conselho Nacional de Saúde, bem como lhe será garantido a assistência integral.

Após os devidos esclarecimentos e estando ciente de acordo com os que me foi exposto, Eu _____ declaro que aceito participar desta pesquisa, dando pleno consentimento para uso das informações por mim prestadas. Para tanto, assino este consentimento em duas vias, rubrico todas as páginas e fico com a posse de uma delas.

Preencher quando necessário

( ) Autorizo o uso de dados sobre tarefas que realizarei durante o experimento.;

( ) Não autorizo.

Teresina, 06 de março de 2025

_____

Assinatura do Participante

_____

Assinatura do Pesquisador Responsável