Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologia em Engenharia – FCTE
Engenharia de Software

Os impactos do uso de Inteligência Artificial no processo de produção de código

Autor: Lude Yuri de Castro Ribeiro e Eric Chagas de Oliveira
Orientador: Prof. Dr. Ricardo Matos Chaim

Brasília, DF
2025

Lude Yuri de Castro Ribeiro e Eric Chagas de Oliveira

Os impactos do uso de Inteligência Artificial no processo de produção de código

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília – UnB

Faculdade de Ciências e Tecnologia em Engenharia – FCTE

Orientador: Prof. Dr. Ricardo Matos Chaim

Brasília, DF

2025

Lude Yuri de Castro Ribeiro e Eric Chagas de Oliveira

Os impactos do uso de Inteligência Artificial no processo de produção de código/ Lude Yuri de Castro Ribeiro e Eric Chagas de Oliveira. – Brasília, DF, 2025-

45 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Dr. Ricardo Matos Chaim

Trabalho de Conclusão de Curso – Universidade de Brasília – UnB
Faculdade de Ciências e Tecnologia em Engenharia – FCTE , 2025.

1. IA, código fonte. 2. Engenharia de Software. I. Prof. Dr. Ricardo Matos Chaim. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Os impactos do uso de Inteligência Artificial no processo de produção de código

CDU 02:141:005.6

Lude Yuri de Castro Ribeiro e Eric Chagas de Oliveira

Os impactos do uso de Inteligência Artificial no processo de produção de código

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 14 de Fevereiro de 2025:

Prof. Dr. Ricardo Matos Chaim
Orientador

Prof. Dr. John Lenon Cardoso Gardenghi
Convidado 1

Prof. Dr. Giovanni Almeida Santos
Convidado 2

Brasília, DF
2025

Lista de abreviaturas e siglas

ABNT	Associação Brasileira de Normas Técnicas
API	*Application Programming Interface* (Interface de Programação de Aplicações)
DRY	*Don't Repeat Yourself* (Não se Repita)
ES	Engenharia de Software
FCTE	Faculdade de Ciências e Tecnologia em Engenharia
FP	Pontos de Função (*Function Points*)
GPT	*Generative Pre-trained Transformer*
IA	Inteligência Artificial
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
ISO/IEC	*International Organization for Standardization / International Electrotechnical Commission*
KLoC	*Kilo Lines of Code* (Mil Linhas de Código)
LLM	*Large Language Model* (Grande Modelo de Linguagem)
LOC	*Lines of Code* (Linhas de Código)
OWASP	*Open Web Application Security Project*
SQuaRE	*Systems and software Quality Requirements and Evaluation*
TCC	Trabalho de Conclusão de Curso
TIC	Tecnologia da Informação e Comunicação
UnB	Universidade de Brasília

Sumário

1 Introdução

1.1 Contexto

1.1.1 A nova realidade

> "A IA generativa é a tecnologia mais transformadora do nosso tempo, possibilitando avanços em criatividade, produtividade e descoberta científica" Huang (2023).

Desde novembro de 2022, o campo da tecnologia testemunha uma transformação significativa com a ascensão da Inteligência Artificial (IA) generativa, impulsionada pelo lançamento em larga escala do ChatGPT pela OpenAI. Essa ferramenta marcou o início de uma nova era no desenvolvimento de software, redefinindo processos e práticas tradicionais.

Nesse contexto de "*nova realidade*", emergem desafios associados a cenários antes considerados distópicos ou restritos à ficção científica. A IA tem transformado profundamente o desenvolvimento de software, introduzindo ferramentas que automatizam e aprimoram processos outrora exclusivos de desenvolvedores humanos. Ferramentas como GitHub Copilot, ChatGPT e Claude, baseadas em IA generativa, permitem a geração de código, sugestões em tempo real e execução de tarefas complexas a partir de comandos em linguagem natural, os chamados *prompts*. Esse cenário redefine a interação entre desenvolvedores e sistemas computacionais, inaugurando um ambiente colaborativo entre humanos e máquinas.

> "Eu diria que talvez 20%, 30% do código que está em nossos repositórios hoje e em alguns de nossos projetos provavelmente foi todo escrito por software" Nadella (2025)[1].

A adoção cada vez mais acelerada dessas ferramentas por empresas e comunidades de desenvolvedores levanta questões sobre produtividade, qualidade, segurança e o papel do desenvolvedor no processo de produção de código. Relatórios corporativos, como os da Microsoft e do GitHub, indicam que uma proporção significativa de código já é gerada por IA, evidenciando a escala dessa transformação. Contudo, os impactos reais dessas tecnologias — incluindo benefícios, desafios e riscos — ainda carecem de análises sistemáticas baseadas em dados públicos.

[1] As datas futuras em referências de notícias e artigos de opinião refletem a natureza prospectiva e de rápida evolução do tema, posicionando a discussão no contexto do ano de defesa desta monografia.

> "Como em todas as revoluções tecnológicas, espero que haja um impacto significativo nos empregos, mas é muito difícil prever exatamente como será esse impacto" Altman (2023).

Esta pesquisa é motivada pela necessidade de compreender como a IA está moldando o desenvolvimento de software, tanto em termos técnicos quanto nas dinâmicas de trabalho. A disponibilidade crescente, embora limitada, de dados públicos — como relatórios de empresas e repositórios — oferece uma oportunidade de investigar essas mudanças e contribuir para o debate acadêmico e profissional sobre o futuro da programação.

1.1.2 Motivação dos autores

Como estudantes de Engenharia de Software, vivenciamos um momento de ruptura tecnológica que redefine nossa profissão e o ecossistema ao seu redor. A integração acelerada da IA no desenvolvimento de software suscita dúvidas, expectativas e receios que nos motivam a investigar como tais ferramentas estão transformando os processos de produção de código e o papel dos desenvolvedores.

> "Nos próximos 18 meses, a maior parte da codificação será feita por IA, e ela será melhor que o trabalho da maioria dos engenheiros plenos" Zuckerberg (2025).

Declarações como essa, feitas por figuras públicas com grande influência no setor tecnológico, intensificam questionamentos sobre o futuro do trabalho em programação. De modo semelhante, Elon Musk apresenta uma visão ainda mais radical:

> "Provavelmente, nenhum de nós terá um emprego [...] Haverá um ponto em que nenhum trabalho será necessário. Você poderá ter um trabalho se quiser ter um trabalho por satisfação pessoal, mas a IA e os robôs fornecerão todos os bens e serviços que você desejar" Moneycontrol News (2024, tradução nossa).

Embora inseridas em um discurso de "abundância universal", tais afirmações funcionam como catalisadores de incertezas, especialmente para estudantes e profissionais em formação.

Além dessas projeções, notícias recentes abordam as dificuldades enfrentadas por profissionais de software no uso de ferramentas de IA, desde iniciantes até desenvolvedores experientes G1 (2024). As percepções sobre o futuro variam entre visões pessimistas e otimistas. Entre os pontos positivistas está a possível diminuição da necessidade do trabalho sistemático de escrita de código.

Segundo Bianchin (2025), Jensen Huang argumenta que não é mais necessário aprender a programar, pois a IA já pode assumir essa função. Nessa perspectiva, enge-

nheiros de software poderiam dedicar mais tempo a atividades que exigem criatividade e inovação Bianchin (2025).

Contribuindo com essa visão, McFarland discute o conceito de *Vibe Coding*, no qual modelos de linguagem permitem que qualquer pessoa crie aplicações ao simplesmente descrever ideias em linguagem natural. Ferramentas como Cursor e Lovable integram agentes inteligentes que ampliam o público capaz de desenvolver software McFarland (2025).

Entretanto, há também visões críticas. Bianchin evidencia preocupações de programadores experientes com a crescente dependência das ferramentas de IA. Para Namanyay Goel, desenvolvedores iniciantes produzem mais código, porém compreendem menos os fundamentos de suas soluções — o que pode gerar consequências futuras Bianchin (2025).

Esse cenário demanda mais do que adoção tecnológica: exige transformação estratégica e cultural. Como ressalta Salvador (2024), o uso da IA deve integrar-se cedo ao fluxo de trabalho, sob metas claras, governança robusta de dados, capacitação contínua e estratégias iterativas de implementação.

Questões jurídicas também emergem. Conforme Basilio (BASILIO, 2025), especialistas divergem sobre a responsabilidade por danos causados por agentes de IA. Empresas podem ser responsabilizadas mesmo quando o erro não se origina nelas, enquanto representantes das desenvolvedoras argumentam que usuários devem assumir parte da responsabilidade ao serem informados sobre limitações dos sistemas. Problemas como "superengenharia" — a criação de arquiteturas excessivamente complexas com múltiplos agentes — também são apontados como riscos (BASILIO, 2025).

Esse conjunto de incertezas motiva a pergunta central deste trabalho:

> *Quais são os reais impactos da Inteligência Artificial na produção de código e como eles afetam o futuro do desenvolvimento de software?*

Com isso, propomos uma análise crítica dos efeitos da IA, investigando suas aplicações e implicações para produtividade, qualidade e relações de trabalho.

1.2 Problema

A rápida adoção de ferramentas de IA no desenvolvimento de software não tem sido acompanhada, de maneira suficiente, por estudos sistemáticos que avaliem seus impactos reais no processo de produção de código. Embora prometam maior produtividade e automação, persistem incertezas quanto à qualidade, segurança, manutenibilidade e im-

plicações éticas e profissionais. Assim, torna-se necessária uma investigação baseada em métricas objetivas e dados públicos (ou privados, nos casos de estudos controlados).

1.3 Objetivos

O objetivo geral deste trabalho é avaliar o impacto prático do uso de ferramentas de IA generativa — especificamente o GitHub Copilot e o ChatGPT — na manutenção e evolução de software. Para isso, combina-se uma revisão teórica focada com um estudo de caso empírico realizado no projeto de código aberto *Zulip*, no qual atividades de refatoração e desenvolvimento incremental são analisadas quantitativamente a partir da ferramenta de análise estática SonarQube.

Para atingir esse objetivo, foram definidos os seguintes objetivos específicos:

- **Sintetizar**, por meio de revisão da literatura, as principais discussões, métricas e desafios sobre IA, qualidade e produtividade no desenvolvimento de software.
- **Estabelecer uma linha de base** de qualidade para um componente selecionado do projeto *Zulip*, por meio de análise inicial com o SonarQube.
- **Executar** um ciclo de manutenção e desenvolvimento empregando o GitHub Copilot como assistente de codificação e o ChatGPT como ferramenta de geração de blocos de código e estratégias de refatoração.
- **Avaliar comparativamente** os artefatos de código antes e após a intervenção assistida por IA, quantificando variações em métricas de qualidade, segurança e dívida técnica.
- **Discutir** as implicações dos resultados à luz da literatura, analisando a eficácia, os benefícios e os desafios das ferramentas avaliadas.

1.4 Contribuições Esperadas

Espera-se que este trabalho contribua para a comunidade acadêmica e profissional em três dimensões principais. Primeiro, fornecendo uma **análise quantitativa** do impacto de ferramentas de IA na qualidade de um software complexo e ativo. Segundo, apresentando um **protocolo de medição replicável** para futuros estudos. Por fim, oferecendo **insights qualitativos** sobre a colaboração humano–IA durante tarefas de refatoração e desenvolvimento, a partir de registros sistemáticos do processo.

1.5 Organização do Trabalho

Este trabalho está estruturado em cinco capítulos. A Introdução apresenta o contexto e a motivação. O Referencial Teórico discute conceitos, ferramentas e estudos relevantes. A Metodologia descreve os métodos empregados. Os Resultados trazem análises e dados coletados. Por fim, as **??**Considerações Finais sintetizam as conclusões, limitações e direções futuras.

2 Referencial Teórico

Este capítulo apresenta os fundamentos teóricos necessários para a compreensão deste trabalho. A discussão inicia-se com uma definição de Inteligência Artificial Generativa e dos Grandes Modelos de Linguagem (LLMs), situando o leitor no atual ecossistema tecnológico, que inclui concorrentes de peso como Claude, Gemini e GPT. Em seguida, justifica-se a seleção das ferramentas específicas utilizadas no experimento: GitHub Copilot e ChatGPT. Posteriormente, são detalhados os conceitos de qualidade de software e as métricas estáticas utilizadas na análise prática. Por fim, o capítulo discute o estado da arte sobre os impactos da IA no ciclo de vida do software, estabelecendo um diálogo entre a visão da indústria e os resultados de estudos acadêmicos recentes.

2.1 Inteligência Artificial Generativa e Modelos de Linguagem

A ascensão da Inteligência Artificial (IA) Generativa representa uma das transformações tecnológicas mais disruptivas da década. Em uma célebre declaração, Huang (2023) a define como "a tecnologia mais transformadora do nosso tempo, possibilitando avanços em criatividade, produtividade e descoberta científica". Diferente de sistemas de IA tradicionais, focados em classificação e previsão, a IA Generativa possui a capacidade de sintetizar conteúdo novo e original, incluindo texto, imagens e, crucialmente para este trabalho, código-fonte.

A base dessa revolução são os *Large Language Models* (LLMs). Estes são modelos de aprendizado profundo construídos sobre a arquitetura *Transformer*, treinados com volumes massivos de dados textuais e repositórios de código. Esse treinamento permite que os modelos internalizem não apenas a sintaxe das linguagens de programação, mas também padrões lógicos, semântica e estruturas de resolução de problemas complexos.

2.2 O Ecossistema Atual de IAs para Desenvolvimento

O cenário de ferramentas de IA para engenharia de software evoluiu rapidamente para um ecossistema competitivo e diversificado. Embora a OpenAI tenha sido pioneira com a série GPT, o mercado atual oferece alternativas robustas que disputam a liderança em *benchmarks* de codificação:

- **Claude 3.5 Sonnet (Anthropic) (Anthropic, 2024):** Frequentemente citado na literatura recente e em *benchmarks* como o SWE-bench por sua alta capacidade

de raciocínio lógico e menor taxa de alucinação em tarefas complexas de codificação. Diferencia-se por um comportamento mais cauteloso e seguro em relação à geração de código sensível.

- **Gemini 1.5 Pro (Google) (Google, 2024):** Destaca-se por sua janela de contexto massiva (até 2 milhões de tokens), permitindo a análise de repositórios inteiros de código em uma única interação. Sua integração nativa com o ecossistema Google facilita a recuperação de informações atualizadas da web.

- **Llama 3 (Meta) (Meta AI, 2024):** Representa o avanço significativo dos modelos de código aberto (*open-source*). Sua disponibilidade permite que empresas rodem assistentes de codificação em infraestrutura própria (*on-premise*), garantindo privacidade de dados e mitigando riscos de vazamento de propriedade intelectual.

2.3 Ferramentas Selecionadas para o Estudo

Apesar da diversidade de opções apresentada acima, este trabalho delimita seu escopo experimental ao uso do **GitHub Copilot (GitHub, 2024)** e do **ChatGPT (OpenAI, 2024)** (baseado no GPT-4). A seleção destas ferramentas não é arbitrária, mas justifica-se por três fatores principais que alinham o experimento à realidade da indústria:

1. **Maturidade e Penetração de Mercado:** O GitHub Copilot é a ferramenta de *AI Pair Programming* mais amplamente adotada na indústria, tornando seus resultados mais representativos da prática cotidiana dos desenvolvedores de software (DOHMKE; IANSITI; RICHARDS, 2023).

2. **Integração de Ambiente:** Diferente de modelos que operam apenas via interface de chat (como Claude ou Gemini na web), o Copilot possui integração profunda com o Visual Studio Code (VS Code). Isso permite o acesso ao contexto do *workspace* (arquivos abertos e estrutura do projeto), característica fundamental para a metodologia de refatoração proposta neste estudo.

3. **Benchmark Acadêmico:** O GPT-4 permanece como o modelo de referência (*baseline*) na maioria dos estudos acadêmicos comparativos sobre qualidade de código, facilitando a correlação dos resultados deste trabalho com a literatura existente.

A seguir, detalham-se as características específicas das ferramentas selecionadas e seus papéis no experimento.

2.3.1 GitHub Copilot: O Programador em Par Contextual

O GitHub Copilot (GitHub, 2024) atua como um assistente integrado ao Ambiente de Desenvolvimento Integrado (IDE). Treinado com bilhões de linhas de código público, ele analisa o contexto imediato — o código adjacente, comentários e abas abertas — para oferecer sugestões de autocompletar em tempo real. O impacto desta ferramenta na produtividade e na qualidade do código é objeto de estudos seminais, como os de Irlbeck et al. (2023) e Peng et al. (2023), que investigam a redução do tempo de codificação em tarefas repetitivas.

2.3.2 ChatGPT: O Parceiro Estratégico

Enquanto o Copilot atua na micro-implementação (linha a linha), o ChatGPT (OpenAI, 2024) é utilizado neste estudo como um parceiro estratégico para tarefas de nível superior, tais como:

- **Arquitetura e Design:** Discussão de padrões arquiteturais, como a aplicação da Arquitetura Limpa e princípios SOLID.
- **Refatoração Guiada:** Análise de blocos de código para sugerir melhorias de legibilidade e redução de complexidade ciclomática.
- **Geração de Testes:** Criação de casos de teste unitários e de integração baseados em regras de negócio descritas em linguagem natural.

2.4 Qualidade de Software: Conceitos e Métricas de Análise

Para avaliar objetivamente o impacto da IA, utiliza-se a norma ISO/IEC 25010 como referência de qualidade. Este trabalho foca em três características mensuráveis via análise estática (SonarQube): Manutenibilidade, Confiabilidade e Segurança.

2.4.1 Manutenibilidade

Define a facilidade com que o software pode ser modificado. Conforme apontado por Irlbeck et al. (2023), a facilidade de geração de código pela IA pode, paradoxalmente, prejudicar a manutenibilidade se não for monitorada, gerando código verborrágico ou difícil de entender. As métricas adotadas para monitorar esse risco são:

- **Dívida Técnica (*Technical Debt*):** O custo estimado do retrabalho necessário para corrigir problemas de qualidade no futuro.

- **Code Smells:** Sintomas no código que indicam possíveis problemas de design ou implementação, dificultando a manutenção.
- **Complexidade Ciclomática:** Medida da complexidade estrutural de uma função, baseada no número de caminhos independentes através do código.
- **Duplicação de Código:** Violação do princípio DRY (*Don't Repeat Yourself*), comum em código gerado por IA que tende a repetir padrões.

2.4.2 Confiabilidade

Refere-se à capacidade do sistema de funcionar sem falhas sob condições específicas. A métrica principal analisada é a presença de **Bugs**, definidos como padrões de código que levam a erros de execução com alta probabilidade.

2.4.3 Segurança

A segurança mede a proteção do software contra ameaças e acessos não autorizados. O trabalho de Sobania et al. (2023) alerta que LLMs podem reproduzir vulnerabilidades presentes em seus dados de treinamento (como injeção de SQL ou exposição de segredos). As métricas monitoradas são:

- **Vulnerabilidades:** Falhas confirmadas que podem ser exploradas por atacantes para comprometer o sistema.
- **Hotspots de Segurança:** Áreas sensíveis do código (ex: criptografia, autenticação) que, embora não sejam falhas confirmadas, exigem revisão manual obrigatória.

2.5 Impactos da Inteligência Artificial no Ciclo de Vida do Software

A literatura acadêmica e a indústria convergem para um debate sobre os efeitos da adoção em massa da IA. Esta seção sintetiza essa discussão, abordando produtividade, qualidade e a evolução do perfil profissional.

2.5.1 Impacto na Produtividade e na Colaboração Humano-IA

A promessa de produtividade é o principal motor da adoção da IA (NADELLA, 2025). Evidências empíricas suportam essa percepção: Peng et al. (2023) demonstrou, em experimento controlado, que usuários do Copilot concluíram tarefas de codificação 55,8% mais rápido que o grupo de controle. Estudos internos no Google corroboram esses dados, apontando uma redução de 21% no tempo de codificação (PARADIS et al., 2024).

Contudo, o ganho não é uniforme. A pesquisa da METR (METR, 2024) observou que, em tarefas de engenharia complexas e mal definidas, o uso da IA pode **atrasar a conclusão em até 19%**, sugerindo que a ferramenta acelera a execução mecânica, mas pode atrapalhar o raciocínio profundo se não for bem gerenciada. Além disso, a taxa de aceitação de sugestões de código na indústria estabilizou-se em torno de 30% (DOHMKE; IANSITI; RICHARDS, 2023), indicando que os desenvolvedores atuam intensamente como curadores e revisores do conteúdo gerado, e não apenas como consumidores passivos.

2.5.2 A Faca de Dois Gumes da Qualidade e Segurança

Em contrapartida aos ganhos de produtividade, emerge a preocupação com a degradação da qualidade a longo prazo. O relatório da GitClear, analisando 153 milhões de linhas de código, identificou um aumento no *code churn* (código descartado logo após a escrita) e o fenômeno da **"dívida técnica induzida pela IA"** (THOMPSON, 2024). A facilidade de gerar código pode levar a soluções funcionais, porém verborrágicas e difíceis de manter.

No aspecto da segurança, a preocupação é ainda maior. Yetiştiren, Özkaya e Ayaydın (2023) e Sobania et al. (2023) destacam que, embora funcional, o código gerado por IA frequentemente carece de tratamentos de borda e sanitização de segurança adequados, exigindo que o desenvolvedor humano mantenha um papel vigilante de auditoria para evitar a introdução de vulnerabilidades conhecidas.

2.5.3 O Novo Papel do Engenheiro de Software

A integração da IA redefine o perfil do engenheiro de software. A pesquisa de Li et al. (2024) sugere que a carga cognitiva se desloca da sintaxe (como escrever o código) para a semântica e a estratégia (o que escrever e por quê). O engenheiro do futuro deixa de ser apenas um "codificador"para se tornar um arquiteto de sistemas e um estrategista de *prompts*, utilizando a IA para amplificar sua capacidade de resolução de problemas complexos, e não apenas para aumentar a velocidade de digitação (SALVADOR, 2024).

3 Metodologia

Este capítulo detalha o desenho da pesquisa e o plano metodológico adotado para atingir os objetivos propostos. A abordagem combina uma fundamentação teórica, construída a partir da revisão de literatura, com um estudo de caso prático de caráter experimental. Serão descritos os procedimentos da pesquisa, o desenho do experimento, os critérios para a seleção do objeto de estudo, as ferramentas utilizadas, as métricas de avaliação e o protocolo de execução que guia a coleta e a análise dos dados. Ao final, são discutidas as limitações inerentes a este desenho metodológico.

Nota sobre o uso de Inteligência Artificial

Em conformidade com as diretrizes acadêmicas de integridade e transparência, declara-se que ferramentas de Inteligência Artificial Generativa (**ChatGPT, GitHub Copilot e Google Gemini**) foram utilizadas na elaboração deste trabalho. Seu uso restringiu-se à revisão gramatical, sugestão de reestruturação de parágrafos para clareza, aprimoramento textual, geração de *snippets* de código demonstrativos e auxílio na estruturação de ideias. Toda a concepção teórica, a análise crítica dos dados, a discussão dos resultados e as conclusões apresentadas são de autoria intelectual original dos pesquisadores humanos.

3.1 Abordagem da Pesquisa

Este trabalho adota uma abordagem de métodos mistos. A primeira fase, de natureza qualitativa, consiste em uma **Revisão da Literatura** para consolidar o entendimento sobre os impactos de ferramentas de IA no desenvolvimento de software. A segunda fase, de natureza quantitativa, é um **Estudo de Caso Empírico** com desenho experimental, que busca gerar evidências concretas sobre os efeitos da IA na qualidade do código.

A revisão da literatura informa o estudo de caso, especialmente na seleção das métricas relevantes (detalhadas na Seção 3.4.7), enquanto o experimento prático oferece os dados que serão discutidos à luz do conhecimento teórico estabelecido.

3.2 Fase 1: Revisão da Literatura

A construção da base teórica desta pesquisa foi realizada por meio de uma revisão focada em cinco artigos acadêmicos recentes, selecionados por sua relevância direta com os objetivos deste trabalho. O propósito foi estabelecer um framework conceitual para guiar o estudo empírico, definindo quais dimensões de impacto — qualidade, produtividade, segurança e experiência do desenvolvedor — são mais proeminentes na literatura atual.

Tabela 1 – Comparação entre os artigos selecionados para a fundamentação teórica.

3.3 Cenário Atual e Seleção das Ferramentas de IA

Embora o mercado de Inteligência Artificial Generativa tenha se expandido rapidamente com o surgimento de modelos robustos como o **Claude 3.5 Sonnet** (Anthropic), conhecido por sua alta capacidade de raciocínio lógico, e o **Gemini 1.5 Pro** (Google), que oferece integração nativa com o ecossistema Google e ampla janela de contexto, este trabalho optou por focar nas ferramentas da OpenAI e GitHub.

A escolha do **ChatGPT (modelo GPT-4)** como o "Arquiteto"justifica-se por sua posição consolidada como *benchmark* de mercado, ampla disponibilidade de documentação e capacidade comprovada em tarefas de *Zero-Shot Learning* para design de sistemas.

Para a função de "Copiloto"na codificação, a escolha do **GitHub Copilot** deve-se à sua integração profunda com o ambiente de desenvolvimento (VS Code) e sua liderança no segmento de *AI Pair Programming*. Diferente de ferramentas baseadas apenas em chat, o Copilot possui acesso ao contexto do *workspace* (arquivos abertos), característica fundamental para a metodologia de refatoração proposta neste estudo.

3.4 Fase 2: Estudo de Caso Empírico

Esta fase consiste na execução do experimento prático para coletar dados quantitativos sobre o impacto do uso das ferramentas GitHub Copilot e ChatGPT no desenvolvimento de software.

3.4.1 Desenho do Experimento

O experimento segue um desenho de **análise pré e pós-intervenção**, conforme ilustrado na Figura 1. Esta abordagem foi escolhida por sua capacidade de isolar e quantificar o impacto de uma intervenção específica. O processo envolve três etapas centrais:

1. Estabelecimento de uma **linha de base (baseline)** de qualidade através da medição do estado inicial do sistema;

2. Execução de uma **intervenção controlada** de desenvolvimento de novas funcionalidades com assistência integral de IA;

3. Uma nova medição para **avaliação comparativa** dos artefatos de código gerados.

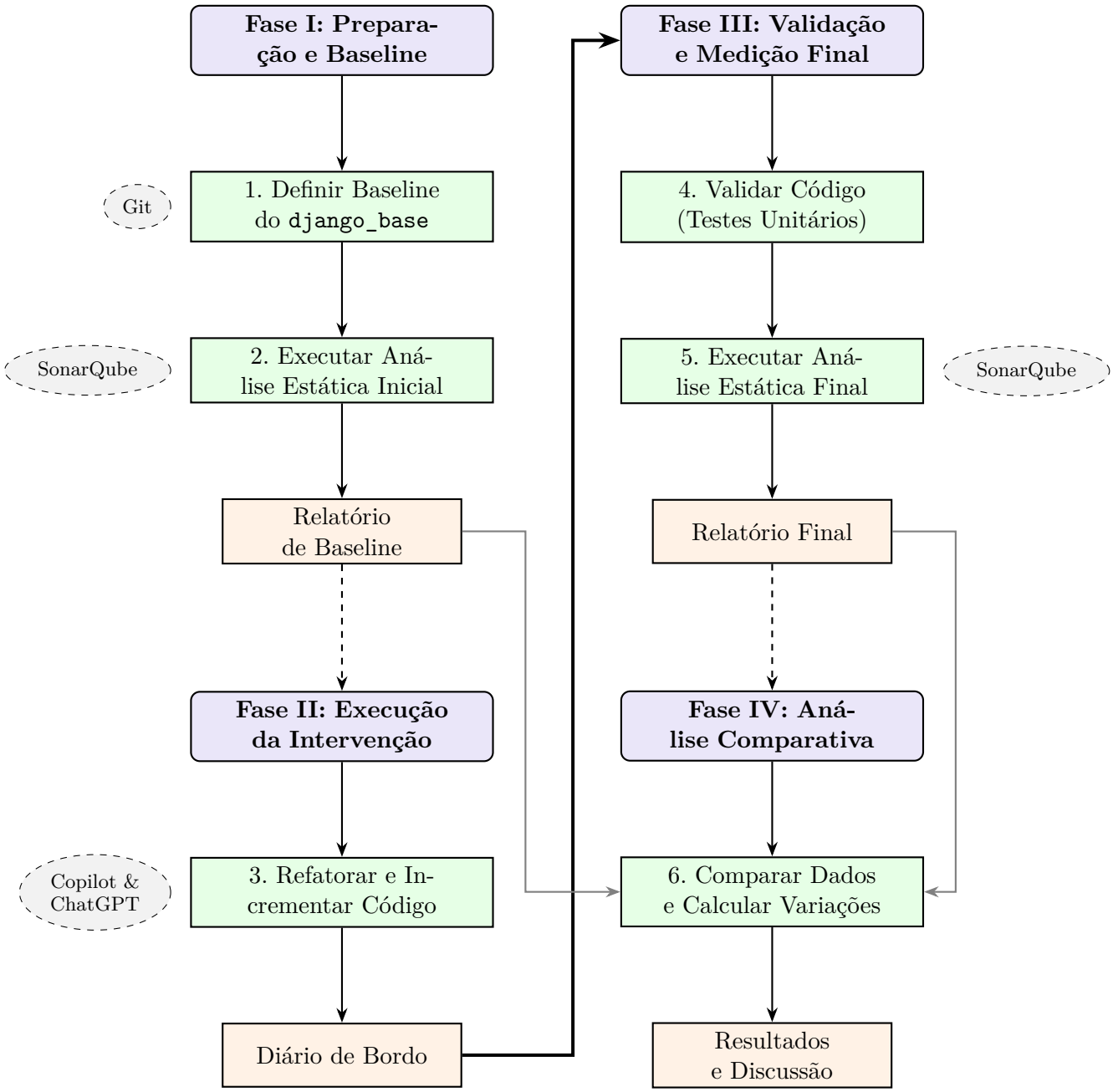


Figura 1 – Fluxograma do desenho experimental pré e pós-intervenção, ilustrando o fluxo entre as fases de análise.

3.4.2 Evolução do Objeto de Estudo e Histórico

Durante a fase inicial da pesquisa (TCC 1), o projeto *open-source* **Zulip** foi selecionado como objeto de estudo para a aplicação das intervenções de IA. No entanto, durante a execução preliminar, identificou-se que a complexidade arquitetural do Zulip e, especificamente, a latência de seu pipeline de CI/CD (com tempos de execução de testes superiores a 30 minutos) inviabilizavam o ciclo rápido de *feedback* necessário para medir a produtividade da IA em tempo real.

Além disso, a análise estática inicial do módulo selecionado no Zulip revelou uma cobertura de testes de apenas 22%, o que criava uma linha de base ruidosa e dificultava a distinção entre o impacto da intervenção e a dívida técnica preexistente. Devido a esses impedimentos técnicos, o Zulip foi descontinuado como foco da intervenção ativa, sendo mantido neste trabalho apenas como registro das limitações metodológicas encontradas em projetos legados de alta complexidade.

3.4.3 Seleção do Novo Objeto de Estudo: `django_base`

Para garantir a viabilidade e o rigor do experimento, o objeto de estudo foi substituído pelo projeto *open-source* **`django_base`** (v2.1.0), um template profissional para projetos Django desenvolvido pelos autores. Esta escolha foi fundamentada em:

- **Linha de Base Robusta:** O projeto apresenta uma *baseline* de alta maturidade (classificação "A"no SonarCloud e 93% de cobertura de testes). Isso permite que ele sirva como um ambiente experimental controlado, onde o impacto da IA pode ser medido com precisão (sinal claro sem ruído).
- **Controle do Ambiente:** O domínio completo do ambiente elimina barreiras de infraestrutura, permitindo a execução rápida de ciclos de análise (Prompt $\rightarrow$ Código $\rightarrow$ Teste $\rightarrow$ Refatoração).
- **Arquitetura Modular:** A estrutura de Arquitetura Limpa do projeto é ideal para o desenho experimental de "evolução de software", permitindo que novos módulos sejam acoplados sem interferência no código legado.

Os resultados apresentados neste trabalho referem-se, portanto, exclusivamente às intervenções realizadas no `django_base`.

3.4.4 Ambiente e Ferramentas de Apoio

Foi definido um ambiente de trabalho padronizado para garantir a consistência do experimento.

3.4.5 Definição do Escopo Experimental (Requisitos)

Para avaliar o impacto da IA em cenários de complexidade distintos, o experimento consiste em duas intervenções de desenvolvimento "greenfield"(código novo). A seguir, são especificados os requisitos para cada intervenção, que formarão o *Product Backlog* do experimento.

3.4.5.1 Intervenção A: Módulo de Lógica de Negócios (Carrinho de Compras)

O objetivo é avaliar a capacidade da IA em lidar com lógica de negócios complexa e gerenciamento de estado.

3.4.5.1.1 Histórias de Usuário (Requisitos Funcionais)

- **HU-A1:** Como um usuário anônimo, eu quero adicionar um produto ao meu carrinho (baseado em sessão), para que eu possa comprá-lo mais tarde.
- **HU-A2:** Como um usuário anônimo, eu quero visualizar todos os itens e o valor total do meu carrinho, para que eu possa revisar meu pedido.
- **HU-A3:** Como um usuário anônimo, eu quero atualizar a quantidade de um item no meu carrinho, para que a lógica de cálculo do total seja re-executada.
- **HU-A4:** Como um usuário anônimo, eu quero remover um item do meu carrinho, para que eu não o compre.

3.4.5.1.2 Requisitos Não Funcionais

- **RNF-A1 (Testabilidade):** A lógica de negócios (cálculo de subtotal, total, adição e remoção) deve ser validada por testes unitários.
- **RNF-A2 (Manutenibilidade):** As funções e métodos do módulo devem manter uma Complexidade Ciclomática baixa (inferior a 10).
- **RNF-A3 (Confiabilidade):** A intervenção não deve introduzir nenhum ‘Bug‘ (classificação do SonarQube) no código novo.

3.4.5.2 Intervenção B: Módulo de Segurança (Login Social OAuth2)

O objetivo é avaliar a capacidade da IA em implementar um fluxo de segurança, uma área onde a literatura aponta riscos significativos.

3.4.5.2.1 Histórias de Usuário (Requisitos Funcionais)

- **HU-B1:** Como um novo usuário, eu quero me cadastrar/autenticar no sistema usando minha conta do Google, para não precisar criar uma conta local com senha.
- **HU-B2:** Como um usuário existente, eu quero fazer login usando minha conta do Google, para acessar minha conta rapidamente e de forma segura.

3.4.5.2.2 Requisitos Não Funcionais

- **RNF-B1 (Segurança):** O fluxo deve implementar o protocolo 'Authorization Code Grant' do OAuth2.
- **RNF-B2 (Segurança):** As chaves secretas (API keys, client secrets) não devem estar expostas no código-fonte (*hardcoded*), devendo ser lidas de variáveis de ambiente.
- **RNF-B3 (Segurança):** O código novo gerado não deve conter nenhuma 'Vulnerabilidade' ou 'Hotspot de Segurança' de criticidade alta ou média (classificação do SonarQube).

3.4.5.3 Backlog do Produto Experimental

A Tabela 2 consolida as *features* (épicos) e suas respectivas histórias de usuário.

Tabela 2 – Backlog do Produto para o experimento de intervenção.

Feature (Épico)	**ID da História**	**Descrição (História de Usuário)**
Gerenciamento de Carrinho de Compras	HU-A1	Adicionar produto ao carrinho (sessão).
	HU-A2	Visualizar carrinho e total.
	HU-A3	Atualizar quantidade de item no carrinho.
	HU-A4	Remover item do carrinho.
Autenticação via Google (OAuth2)	HU-B1	Cadastrar-se no sistema com conta Google.
	HU-B2	Fazer login no sistema com conta Google.

Fonte: os Autores.

3.4.6 Arquitetura da Aplicação

Foi definida e estruturada, com o auxílio do assistente de IA ChatGPT, a arquitetura final para a aplicação. O objetivo desse processo é entender como o agente de IA interpreta os requisitos e contribui para a definição arquitetural.

3.4.6.1 Arquitetura sugerida pelo Agente

A arquitetura para a intervenção no projeto `django_base` foi desenhada para otimizar a testabilidade e a segurança. A estrutura segue o padrão nativo do Django, utilizando a separação por *Apps* de domínio e introduzindo uma **Camada de Serviço** para isolar a lógica de negócios.

3.4.6.1.1 Estrutura e Isolamento por Apps:

O projeto será composto pelos seguintes Apps, garantindo o Princípio da Responsabilidade Única (SRP):

- `shopping_cart`: App responsável pela Intervenção A (Carrinho de Compras).
- `accounts`: App responsável pela Intervenção B (Segurança/OAuth2).

3.4.6.1.2 Camada de Serviço (Intervenção A):

Para cumprir o RNF-A1 (Testabilidade) e RNF-A2 (Complexidade), toda a lógica complexa de gerenciamento é encapsulada em uma camada de serviço (`shopping_cart/services.py`).

- A **Camada de Serviço** manipula o estado do carrinho.
- As Views HTTP apenas delegam chamadas a esta camada e lidam com a resposta.

3.4.6.1.3 Segurança e Configuração (Intervenção B):

Para atender aos requisitos de segurança (RNF-B1 a RNF-B3):

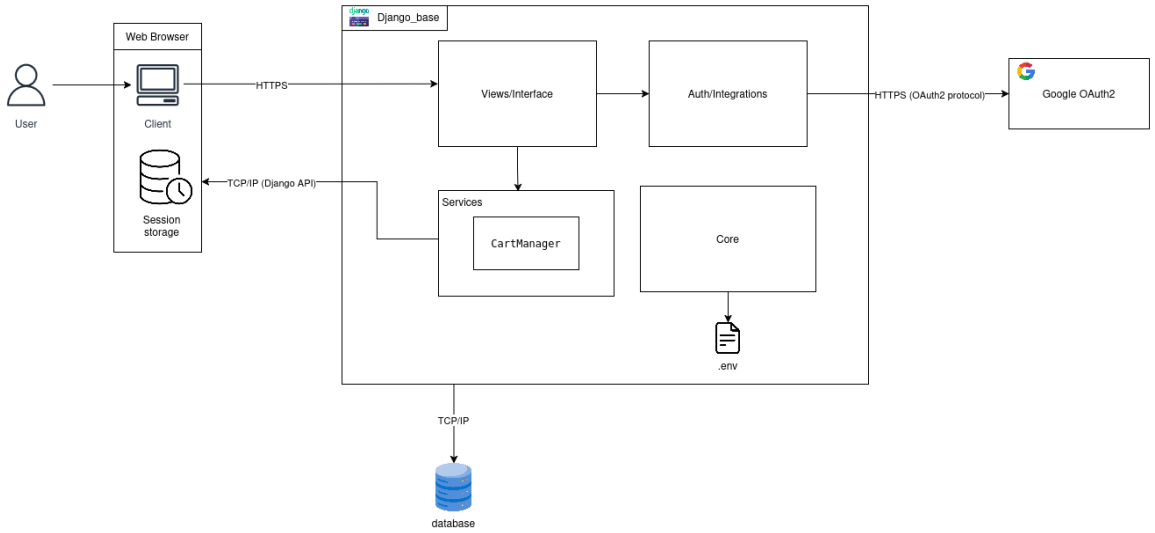
- **Autenticação:** Utilização de bibliotecas consolidadas para implementar o fluxo `Authorization Code Grant`.
- **Segredos:** Todas as chaves devem ser lidas via **variáveis de ambiente** (`.env`), prevenindo a exposição no código-fonte.

3.4.6.2 Desenho da Arquitetura Definida

O desenho da arquitetura pode ser visualizado na Figura 2.

O diagrama ilustra o isolamento dos domínios da intervenção (Carrinho e Segurança) e a separação de responsabilidades (SRP) dentro do projeto `django_base`.

Figura 2 – Diagrama de Arquitetura de Software.



Fonte: os Autores.

3.4.6.2.1 Níveis de Abstração e Componentes:

1. **Contêiner Principal (`django_base`):** Representa o limite da aplicação.
2. **Módulos Internos:**
   - **Views/Interface:** Camada de entrada/saída que recebe requisições HTTP.
   - **Lógica de Negócios (Serviços):** Contém as classes puras do `shopping_cart/services.py` É isolada e altamente testável.
   - **Autenticação/Integração:** Lida com o fluxo de segurança do `accounts` App.
3. **Serviços Externos:** Componentes de infraestrutura necessários.

3.4.6.2.2 Comunicações Críticas:

- **Fluxo Carrinho (Intervenção A):** A camada de Interface delega à Lógica de Negócios, que se comunica com o Sistema de Sessão.
- **Fluxo Segurança (Intervenção B):** A camada de Autenticação se comunica com o provedor OAuth2. A Configuração obtém chaves estritamente das Variáveis de Ambiente, cumprindo o RNF-B2.

3.4.7 Definição das Métricas de Análise

A seleção das métricas foi um processo deliberado...

3.4.8 Critérios de Interpretação dos Resultados

3.4.9 Protocolo de Execução do Experimento

O protocolo experimental foi revisado para se adequar ao novo objeto de estudo. O processo será dividido em duas intervenções paralelas, executadas por pesquisadores distintos.

3.4.9.1 Intervenção A: Módulo de Lógica de Negócios (Carrinho de Compras)

- **Pesquisador Responsável:** Lude Ribeiro.
- **Objetivo:** Implementar um novo módulo de "Carrinho de Compras"anônimo.
- **Hipótese a Validar:** Avaliar a capacidade da IA (Copilot e ChatGPT) em lidar com lógica de negócios complexa, gerenciamento de estado e geração de testes.
- **Indicadores-Chave:** Complexidade Ciclomática e Cobertura de Testes.

3.4.9.2 Intervenção B: Módulo de Segurança (Login Social OAuth2)

- **Pesquisador Responsável:** Eric Chagas.
- **Objetivo:** Implementar a integração com um provedor de "Login Social"via OAuth2.
- **Hipótese a Validar:** Avaliar a eficácia da IA na implementação de fluxos de segurança, verificando a inserção de vulnerabilidades.
- **Indicadores-Chave:** Vulnerabilidades e Hotspots de Segurança (CodeQL/Sonar).

Fases de Execução (Aplicadas a cada Intervenção)

Cada intervenção seguirá rigorosamente o mesmo protocolo de 4 fases em *feature branches* Git separadas:

Fase I: Preparação

1. **Isolamento do Código-Fonte:** Criação de uma *feature branch* a partir da *main* (baseline).
2. **Planejamento da Intervenção:** Definição dos requisitos específicos.

Fase II: Execução da Intervenção (Desenvolvimento Assistido)

3. **Desenvolvimento Assistido por IA:** Realização da codificação com o auxílio integral do GitHub Copilot e ChatGPT.
4. **Registro Sistemático (Diário de Bordo):** Manutenção de um diário de bordo (disponível no Apêndice B) registrando os *prompts*, sugestões acatadas/rejeitadas e justificativas.

Fase III: Validação e Medição Final (Pós-Intervenção)

5. **Validação Funcional:** Verificação de que o código compila e os testes passam.
6. **Execução da Análise Estática:** Execução do SonarQube e 'pytest-cov' na *feature branch*.
7. **Coleta de Dados Finais:** Arquivamento dos relatórios de qualidade.

Fase IV: Análise Comparativa dos Resultados

8. **Tabulação dos Dados:** Organização dos dados pós-intervenção.
9. **Análise Crítica:** Avaliação da qualidade intrínseca do código gerado "do zero"pela IA, utilizando a *baseline* de 93% de cobertura como padrão-ouro.
10. **Interpretação:** Análise dos resultados quantitativos, correlacionando-os com o Diário de Bordo.

3.4.10 Limitações da Metodologia

É imperativo reconhecer as limitações deste novo desenho de pesquisa.

- **Generalização dos Resultados:** Por se tratar de um estudo de caso focado em um único projeto (`django_base`), os resultados representam uma evidência pontual e aprofundada, não uma conclusão universal para todos os contextos de software.
- **Viés de Autoria e Familiaridade:** A principal limitação é o uso de um projeto de autoria de um dos pesquisadores. Isso introduz um viés de familiaridade que pode otimizar a interação com o código. No entanto, esta limitação é mitigada pela adoção de uma linha de base de alta maturidade (padrão rigoroso a ser mantido) e pelo uso de ferramentas de análise estática (SonarQube) como "árbitro"imparcial da qualidade.

4 Resultados

Este capítulo apresenta os dados coletados e a análise dos resultados obtidos a partir do experimento controlado realizado no projeto `django_base`. A exposição está organizada de forma a responder aos objetivos específicos do trabalho, iniciando pela definição da linha de base (*baseline*) de qualidade do projeto, seguida pela apresentação dos dados das intervenções realizadas nas frentes de Lógica de Negócios e Segurança, e finalizando com uma análise comparativa consolidada.

4.1 Estabelecimento da Linha de Base (Baseline)

A primeira etapa do protocolo experimental consistiu na execução de uma auditoria completa do estado inicial do projeto `django_base` (versão 2.1.0, branch `main`). O objetivo foi validar a qualidade do objeto de estudo e estabelecer os parâmetros quantitativos de comparação (o "padrão-ouro") para as intervenções subsequentes.

Para garantir uma visão holística, foram utilizadas duas ferramentas complementares: o SonarQube, focado em qualidade geral e manutenibilidade, e o GitHub Advanced Security (CodeQL), focado em vulnerabilidades de segurança profundas.

4.1.1 Análise de Qualidade e Manutenibilidade (SonarQube)

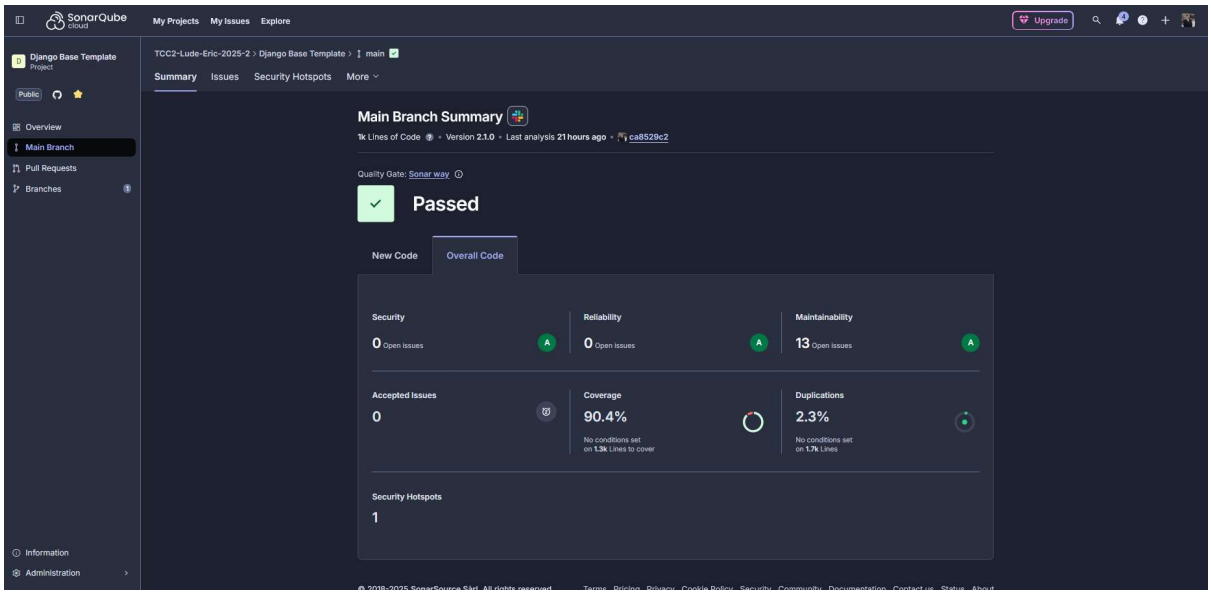
A análise estática realizada pelo SonarQube confirmou a hipótese de que o projeto selecionado possui um alto grau de maturidade técnica, servindo como um ambiente controlado ideal para o experimento. Conforme ilustrado na Figura 3, o projeto obteve aprovação no *Quality Gate* padrão (*Sonar way*).

Os dados brutos coletados, resumidos na Tabela 3, demonstram um código limpo e bem testado. Destaca-se a cobertura de testes de **90.4%**, um valor significativamente superior à média da indústria, e a classificação "A"em todas as categorias principais (Segurança, Confiabilidade e Manutenibilidade). O baixo índice de duplicação (2.3%) e o número reduzido de *code smells* (13) indicam uma base de código coesa e aderente aos princípios de Arquitetura Limpa.

4.1.2 Análise de Segurança Avançada (CodeQL)

Enquanto o SonarQube apresentou um cenário de segurança quase perfeito (0 vulnerabilidades), a análise semântica profunda realizada pelo GitHub Advanced Security (CodeQL) revelou uma realidade distinta e mais complexa.

Figura 3 – Dashboard do SonarQube exibindo a Baseline da branch `main`.



Fonte: Dados extraídos do SonarCloud (2025).

Tabela 3 – Métricas de Baseline coletadas via SonarQube (Branch `main`).

Métrica	**Valor Aferido**	**Classificação/Status**
Quality Gate	Passed	Aprovado
Linhas de Código	1.3k	-
Cobertura de Testes	**90.4%**	Excelente
Duplicação de Código	2.3%	Baixo
Dívida Técnica (*Code Smells*)	13 issues	Classificação A (Baixa)
Bugs	0	Classificação A (Confiável)
Vulnerabilidades (Sonar)	0	Classificação A (Seguro)
Security Hotspots	1	Revisão Necessária

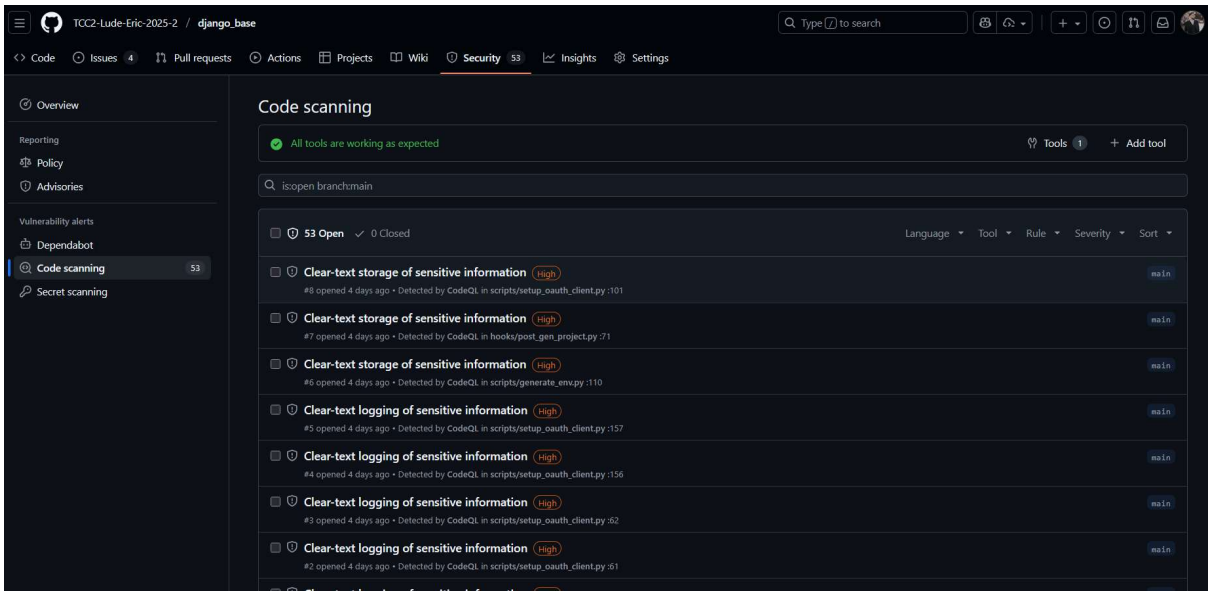
Fonte: os Autores.

Conforme evidenciado na Figura 4, a ferramenta detectou um total de **53 alertas de segurança latentes** na base de código. Deste total, **18 alertas** foram classificados com severidade **Alta (High)**, indicando riscos reais que não foram capturados pela análise estática tradicional.

A análise detalhada dos alertas permitiu categorizar as principais fragilidades arquiteturais do projeto base:

- **Log Injection (High):** A maior incidência de alertas críticos (ex: `views/user.py`) refere-se à injeção de logs, onde dados de entrada de usuários não confiáveis são gravados nos registros do sistema sem a devida sanitização, permitindo que atacantes falsifiquem entradas de log.

Figura 4 – Painel de Segurança do GitHub (CodeQL) revelando alertas na Baseline.



Fonte: Dados extraídos do GitHub Security (2025).

- **Armazenamento de Dados Sensíveis em Texto Claro (High):** O CodeQL identificou que scripts de configuração e *hooks* do projeto (ex: `setup_oauth_client.py`) podem estar manipulando ou armazenando informações sensíveis sem criptografia.
- **Logging de Dados Sensíveis (High):** Alertas indicando que informações confidenciais podem estar sendo enviadas para a saída padrão ou arquivos de log durante a execução de scripts de automação.

Esta discrepância entre as ferramentas (0 vulnerabilidades no Sonar vs. 53 alertas no CodeQL) é um achado preliminar relevante. Ela demonstra que métricas tradicionais de qualidade de código, embora essenciais, não garantem a segurança semântica da aplicação. Para o experimento de intervenção, estes 53 alertas constituem a "linha de base de segurança real"que não deve ser degradada pelo código gerado pela IA.

5 Cronograma de Execução e Gestão de Riscos

Este capítulo apresenta o planejamento temporal e o registro da execução das atividades do TCC 2. Diferente de um cronograma linear tradicional, este plano de trabalho reflete a **natureza adaptativa** da pesquisa experimental. O cronograma foi reestruturado durante o curso do semestre para acomodar a decisão estratégica de substituição do objeto de estudo (Pivot), mitigando os riscos técnicos que ameaçavam a viabilidade do prazo final.

O cronograma final consolida-se em um regime de execução intensiva (*Sprint*), visando a entrega do documento completo em **14 de Dezembro** e a defesa na data prevista de **16 de Dezembro**.

5.1 Detalhamento das Fases e Adaptação do Plano

A execução do projeto foi segmentada em quatro fases distintas. A Tabela 4 detalha a transição crítica da Fase 1 para a Fase 2, onde ocorreu a mudança de escopo técnico, e a Tabela 5 apresenta o planejamento final de escrita e defesa.

Tabela 4 – Cronograma Executivo TCC 2 - Fase Exploratória e Pivot (Ago/25 a Nov/25)

Fase	Atividade e Descrição Detalhada	Período (Semanas)	Entregável / Marco
Fase 1	**Exploração e Baseline (Projeto Zulip):** Configuração de ambiente complexo, tentativa de isolamento de módulos e execução da análise estática inicial. Identificação de bloqueios de infraestrutura.	18/08 a 31/10 (Sem. 1-10)	Relatório de Impedimentos Técnicos.
	Análise de Risco e Tomada de Decisão (Pivot): Avaliação da inviabilidade do ciclo de feedback no Zulip. Seleção do `django_base` como novo objeto de estudo.	01/11 a 07/11 (Sem. 11)	**Marco de Decisão:** Troca de Objeto de Estudo.
	Setup do Novo Ambiente (django_base): Configuração do pipeline de CI/CD simplificado, Docker Compose e estabelecimento da nova Baseline de qualidade (SonarQube).	08/11 a 17/11 (Sem. 12-13)	Nova Baseline ("Padrão Ouro") definida.
Fase 2	**Intervenção Assistida por IA (Sprint de Desenvolvimento):** Implementação dos módulos de **Carrinho de Compras** (Lógica) e **Autenticação OAuth2** (Segurança) utilizando Copilot e ChatGPT.	**18/11 a 24/11 (Sem. 14)**	Código funcional, Testes e Diário de Bordo.
	Validação e Coleta de Métricas: Execução de testes de regressão, testes de carga e nova análise estática (Pós-Intervenção). Coleta dos relatórios de cobertura.	25/11 a 27/11 (Sem. 15)	Dados Brutos para Análise.
	Análise Quantitativa e Visualização: Processamento dos dados, cálculo de delta (Antes vs. Depois) e geração dos gráficos comparativos.	28/11 a 30/11 (Sem. 15)	Gráficos do Cap. de Resultados.
	Escrita do Capítulo de Resultados: Redação técnica descrevendo os achados, correlacionando métricas com o Diário de Bordo.	01/12 a 02/12 (Sem. 16)	Capítulo 4 Finalizado.

Tabela 5 – Cronograma Executivo TCC 2 - Finalização e Defesa (Dezembro/25)

Fase	Atividade e Descrição Detalhada	Período (Semanas)	Entregável / Marco
Fase 3	**Consolidação e Discussão:** Interpretação crítica dos resultados à luz do referencial teórico. Redação das Considerações Finais e Trabalhos Futuros.	03/12 a 05/12 (Sem. 16)	Texto completo (Draft).
	Revisão Geral e Normalização ABNT: Leitura completa, correção de citações, formatação de apêndices (Diário de Bordo e Protocolos) e revisão textual.	06/12 a **13/12** (Sem. 17)	Versão Final para Depósito.
	Entrega Formal do Documento (DATA LIMITE): Envio do documento final ao orientador e banca examinadora.	**14/12 (Sem. 17)**	**MARCO FINAL: ENTREGA.**
Fase 4	**Preparação para Banca:** Elaboração dos slides, roteiro de apresentação e revisão dos principais pontos de arguição.	10/12 a 15/12 (Sem. 17-18)	Material de Apresentação.
	Defesa Pública: Apresentação do trabalho à banca examinadora.	**16/12 (Confirmado)** (Sem. 18)	**Conclusão do Curso.**

5.2 Visualização da Execução (Gantt)

A Figura 5 ilustra a distribuição temporal das atividades, destacando a extensão do prazo de escrita e revisão até a entrega final em 14/12.

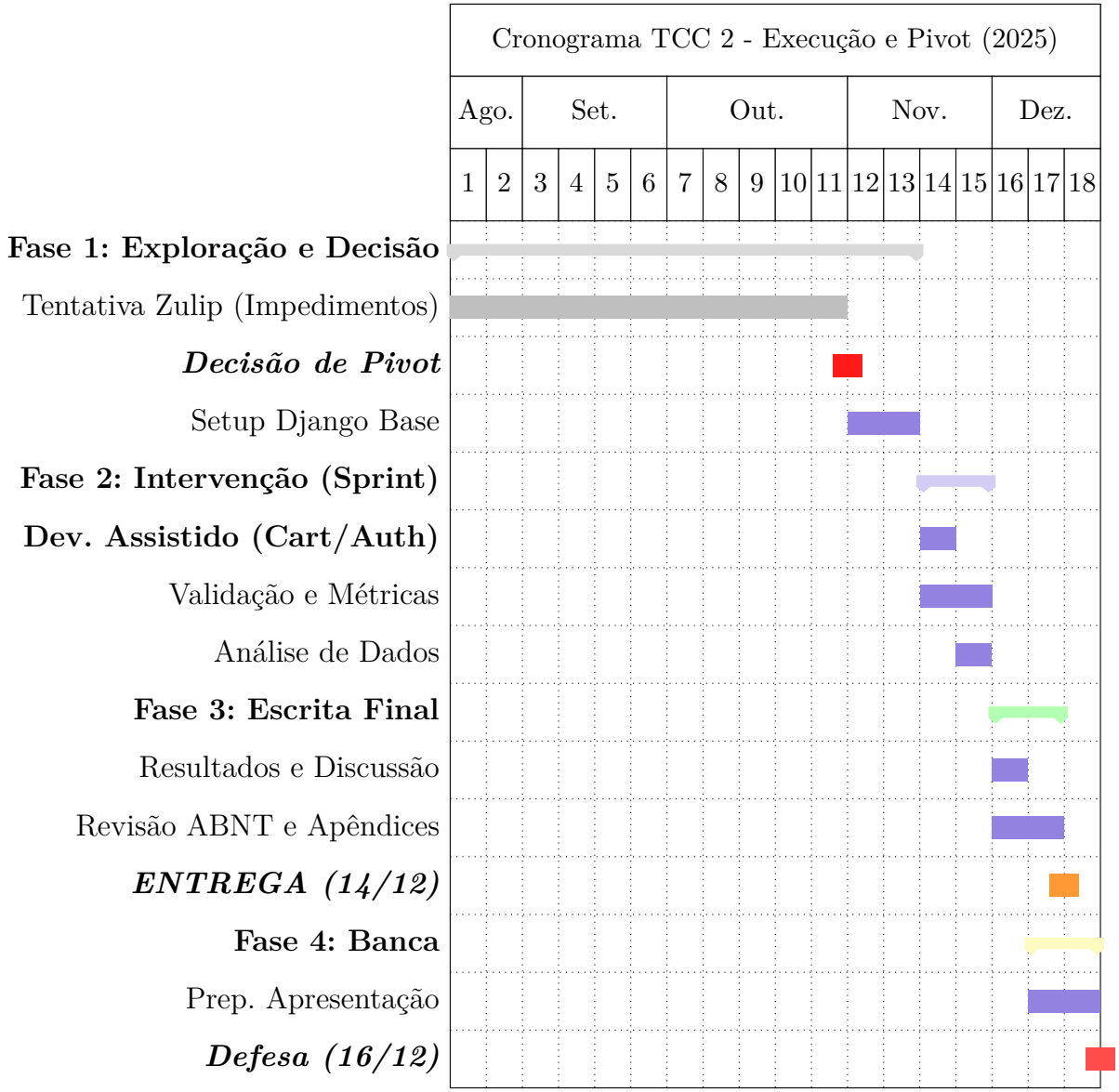


Figura 5 – Linha do tempo do projeto, evidenciando o momento de decisão (Pivot) e o cronograma final até a defesa em 16/12.

Referências

ALTMAN, S. *The Best Generative AI Quotes*. 2023. Acessado em: 17 jul. 2025. Disponível em: <https://www.supplychaintoday.com/generative-ai-quotes/>. Citado na página 9.

Anthropic. *The Claude 3 Model Family: Opus, Sonnet, Haiku*. 2024. Acessado em: 08 dez. 2025. Disponível em: <https://www.anthropic.com/news/claude-3-family>. Citado na página 13.

BASILIO, P. *Quem será responsabilizado quando os agentes de IA errarem?* 2025. Publicado em: Época Negócios. Acessado em: 23 jul. 2025. Disponível em: <https://epocanegocios.globo.com/inteligencia-artificial/noticia/2025/05/quem-sera-responsabilizado-quando-os-agentes-de-ia-errarem.ghtml>. Citado na página 10.

BIANCHIN, V. *Os jovens programadores já não sabem programar: a IA está provocando a mesma revolução que a calculadora causou há meio século*. 2025. Publicado em: Xataka Brasil. Acessado em: 17 jul. 2025. Disponível em: <https://www.xataka.com.br/informatica/os-jovens-programadores-ja-nao-sabem-programar-a-ia-esta-provocando-a-mesma-revolucao-que-a-calc
Citado na página 10.

DOHMKE, T.; IANSITI, M.; RICHARDS, G. *A Sea Change in Software Development: Economic and Productivity Analysis of the AI-Powered Developer Lifecycle*. [S.l.], 2023. Disponível em: <https://arxiv.org/abs/2306.15033>. Citado 2 vezes nas páginas 14 e 17.

G1. *O impacto da inteligência artificial na carreira de engenheiros de software*. 2024. Acessado em: 17 jul. 2025. Disponível em: <https://g1.globo.com/pr/parana/especial-publicitario/uniopet/opet-inovacao-em-rede/noticia/2024/10/04/o-impacto-da-inteligencia-artificial-na-carreira-de-engenheiros-de-software.ghtml>. Citado na página 9.

GitHub. *GitHub Copilot: Your AI pair programmer*. 2024. Acessado em: 08 dez. 2025. Disponível em: <https://github.com/features/copilot>. Citado 2 vezes nas páginas 14 e 15.

Google. *Gemini 1.5: Our next-generation model*. 2024. Acessado em: 08 dez. 2025. Disponível em: <https://blog.google/technology/ai/google-gemini-next-generation-model-february-2024/>. Citado na página 14.

HUANG, J. *The Best Generative AI Quotes*. 2023. Acessado em: 17 jul. 2025. Disponível em: <https://www.supplychaintoday.com/generative-ai-quotes/>. Citado 2 vezes nas páginas 8 e 13.

IRLBECK, M. et al. Assessing the impact of ai-based pair programmers on code quality. In: *Proceedings of the 2023 Conference on Empirical Methods in Natural Language*

Processing. Association for Computational Linguistics, 2023. p. 13190–13200. Disponível em: <https://aclanthology.org/2023.emnlp-main.819>. Citado na página 15.

ISO/IEC. *Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models*. Geneva, 2023. Citado 3 vezes nas páginas 38, 39 e 40.

LI, C.-L. et al. Tapping into the cognitive world of novice programmers: Evaluating gpt-4 for pair programming. In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. New York, NY, USA: ACM, 2024. (CHI '24). Citado na página 17.

MCFARLAND, A. *Vibe Coding: Como a IA está mudando o desenvolvimento de software para sempre*. 2025. Acessado em: 17 jul. 2025. Disponível em: <https://www.unite.ai/vibe-coding-como-a-ia-esta-mudando-o-desenvolvimento-de-software-para-sempre/>. Citado na página 10.

Meta AI. *Introducing Meta Llama 3: The most capable openly available LLM to date*. 2024. Acessado em: 08 dez. 2025. Disponível em: <https://ai.meta.com/blog/meta-llama-3/>. Citado na página 14.

METR. *Measuring the Impact of LLMs on Software Engineering*. 2024. Acessado em: 17 jul. 2025. Disponível em: <https://metr.org/blog/2024-05-21-measuring-llm-impact-on-se>. Citado na página 17.

Moneycontrol News. *'None of us will have a job': Elon Musk makes big claims about AI taking over human jobs*. 2024. Acessado em: 17 jul. 2025. Disponível em: <https://www.moneycontrol.com/technology/none-of-us-will-have-a-job-elon-musk-makes-big-claims-about-ai-taking-over-human-jobs-article-12977 html>. Citado na página 9.

NADELLA, S. *Satya Nadella says as much as 30% of Microsoft code is written by AI*. 2025. Publicado em: CNBC. Acessado em: 17 jul. 2025. Disponível em: <https://www.cnbc.com/2025/04/29/satya-nadella-says-as-much-as-30percent-of-microsoft-code-written-by-ai.html>. Citado 2 vezes nas páginas 8 e 16.

OpenAI. *ChatGPT: Optimizing Language Models for Dialogue*. 2024. Versão GPT-4. Acessado em: 08 dez. 2025. Disponível em: <https://openai.com/chatgpt>. Citado 2 vezes nas páginas 14 e 15.

PARADIS, E. et al. How much does ai impact development speed? a controlled experiment at google. *arXiv preprint arXiv:2403.00382*, 2024. Disponível em: <https://arxiv.org/abs/2403.00382>. Citado na página 16.

PENG, S. et al. The impact of ai on developer productivity: Evidence from github copilot. *arXiv preprint arXiv:2302.06590*, 2023. Acessado em: 17 jul. 2025. Disponível em: <https://arxiv.org/abs/2302.06590>. Citado 2 vezes nas páginas 15 e 16.

SALVADOR, E. *Seis estratégias para que desenvolvedores adotem a IA mais rápido*. 2024. Acessado em: 17 jul. 2025. Disponível em: <https://about.gitlab.com/pt-br/the-source/ai/6-strategies-to-help-developers-accelerate-ai-adoption/>. Citado 2 vezes nas páginas 10 e 17.

SOBANIA, D. et al. An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653*, 2023. Disponível em: <https://arxiv.org/abs/2301.08653>. Citado 2 vezes nas páginas 16 e 17.

THOMPSON, B. *AI-Generated Code Raises Quality Concerns*. 2024. Acessado em: 17 jul. 2025. Disponível em: <https://blog.gitclear.com/code-churn-and-ai-a-looming-problem/>. Citado na página 17.

YETIşTIREN, B.; ÖZKAYA, M.; AYAYDıN, Y. Evaluating the code quality, performance, and security of ai-assisted code generation tools: A comparative study of github copilot, amazon codewhisperer, and chatgpt. *arXiv preprint arXiv:2305.11225*, 2023. Disponível em: <https://arxiv.org/abs/2305.11225>. Citado na página 17.

ZUCKERBERG, M. *Mark Zuckerberg says in 18 months coding will be done by AI*. 2025. Publicado em: India Today. Acessado em: 17 jul. 2025. Disponível em: <https://www.indiatoday.in/technology/news/story/mark-zuckerberg-says-in-18-months-coding-will-be-done-by-ai-it-will-be-better-than-work-of-most-eng
Citado na página 9.

APÊNDICE A – Mapeamento das Métricas de Manutenibilidade

Tabela 6 – Relação entre as métricas de Manutenibilidade do estudo e as subcaracterísticas da norma ISO/IEC 25010:2023 (ISO/IEC, 2023).

Subcaracterística	**Métrica (SonarQube)**	**Relação e Alinhamento**
Analisabilidade	**Dívida Técnica**	Medida em tempo de remediação, indica o esforço para entender e consertar o código. Alto nível de dívida compromete a analisabilidade.
	Code Smells	Tornam o código complexo de diagnosticar. A redução melhora a facilidade de análise.
Modificabilidade	**Complexidade Ciclomática**	Funções complexas são difíceis de modificar. O controle garante modificações seguras.
	Código Duplicado	Exige mudanças em múltiplos locais, aumentando risco de erros.
Testabilidade	(Cobertura de Testes)	Embora crítica, a cobertura complementa a análise de manutenibilidade.

APÊNDICE B – Mapeamento da Métrica de Confiabilidade

Tabela 7 – Relação entre a métrica de Confiabilidade do estudo e as subcaracterísticas da norma ISO/IEC 25010:2023 (ISO/IEC, 2023).

Subcaracterística	**Métrica (SonarQube)**	**Relação e Alinhamento**
Tolerância a Falhas	**Bugs**	Bugs são a manifestação direta de problemas de confiabilidade. Sua redução aumenta a confiança no sistema.

APÊNDICE C – Mapeamento das Métricas de Segurança

Tabela 8 – Relação entre as métricas de Segurança do estudo e as subcaracterísticas da norma ISO/IEC 25010:2023 (ISO/IEC, 2023).

Subcaracterística	**Métrica (SonarQube)**	**Relação e Alinhamento**
Integridade	**Vulnerabilidades**	Correção de falhas que permitem exploração do sistema.
Confidencialidade	**Hotspots de Segurança**	Pontos sensíveis que exigem revisão manual para garantir proteção de dados.

APÊNDICE D – Protocolo de Reproducibilidade Experimental

Este apêndice documenta os artefatos de configuração utilizados. O objetivo é permitir a reprodução do experimento, contrastando a complexidade do projeto inicial (Zulip) com a solução otimizada (Django Base).

D.1 Ambiente 1: Projeto Zulip (Fase Exploratória)

A complexidade da infraestrutura de testes do Zulip inviabilizou o ciclo de feedback rápido. Abaixo, o trecho do arquivo `zulip-ci.yml` que demonstra a dependência de contêineres customizados e scripts proprietários.

```
jobs:
  tests:
    strategy:
      matrix:
        include:
          - docker_image: zulip/ci:jammy
            name: Ubuntu 22.04 (Python 3.10)
    container: ${{ matrix.docker_image }}
    steps:
      - name: Install dependencies
        run: |
          ./tools/ci/setup-backend --skip-dev-db-build
      - name: Run backend tests
        run: |
          ./tools/test-backend --coverage
```

D.2 Ambiente 2: Projeto Django Base (Ambiente Ativo)

Ambiente onde as intervenções foram executadas. Utiliza padrões de mercado para garantir reprodutibilidade.

D.2.1 Containerização (Docker Compose)

Trecho do arquivo `docker-compose.dev.yml` utilizado:

```
services:
    project:
        container_name: project_dev
        build:
            context: .
            dockerfile: Dockerfile.dev
        ports:
            - "8000:8000"
        depends_on:
            project_db:
                condition: service_healthy

    project_db:
        image: postgres:13-alpine
        healthcheck:
            test: ["CMD-SHELL", "pg_isready -U postgres"]
            interval: 5s
```

D.2.2 Automação (Makefile)

Comandos utilizados para padronizar a execução dos testes e análises. *Nota: Os emojis originais foram removidos para compatibilidade com a compilação do documento.*

```
# Execucao da suite de testes completa
test:
    @echo "[TEST] Executando testes..."
    @cd $(PROJECT_DIR) && . ../$(VENV)/bin/activate && \
    export PYTHONPATH=$$PWD && $(PYTEST) -v core cart

# Medicao de cobertura para o Sonar
test-coverage:
    @echo "[COVERAGE] Executando testes com cobertura..."
    @cd $(PROJECT_DIR) && . ../$(VENV)/bin/activate && \
    export PYTHONPATH=$$PWD && \
    $(PYTEST) --cov=core --cov=cart --cov-report=xml
```

```
# Analise de Seguranca
security-check:
    @echo "[SECURITY] Verificando vulnerabilidades..."
    @cd $(PROJECT_DIR) && . ../$(VENV)/bin/activate && pip-audit
```

D.2.3 Pipeline CI/CD (GitHub Actions)

Configuração do *workflow* para validação contínua.

```
name: CI/CD Pipeline
on: [push, pull_request]
jobs:
  test-and-quality:
    runs-on: ubuntu-latest
    steps:
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.12"
      - name: Run tests
        run: |
          cd project
          pytest --cov=core --cov=cart --cov-report=xml:coverage.xml
      - name: SonarCloud Scan
        uses: SonarSource/sonarcloud-github-action@master
```

APÊNDICE E – Diário de Bordo Experimental

Este apêndice apresenta o registro estruturado das sessões de pareamento com as ferramentas de IA, detalhando os desafios de contexto e as decisões metodológicas.

Entrada 1: Definição Arquitetural e Configuração Inicial

Pesquisador	Lude Ribeiro
Requisito	Arquitetura do Módulo (Modular Monolith)
Ferramenta	ChatGPT 4.0 (Web)
Tempo	3 horas

1. Objetivo: Obter a estrutura de pastas correta para o módulo `cart` e scripts de automação.

2. Interação (Prompt):

> "Atue como um Arquiteto de Software. Preciso criar um NOVO módulo isolado chamado `cart`. Forneça um script bash para criar a árvore de diretórios."

3. Resultado e Análise: A IA gerou um script que criava pastas na raiz errada do projeto (falha de contexto espacial).

- **Decisão:** Modificado. O pesquisador interveio manualmente para executar o script dentro da pasta `project/`.
- **Observação:** Um erro de infraestrutura (Docker) causou desvio de foco, levando a IA a "esquecer"a arquitetura definida inicialmente.

Entrada 2: Crise de Contexto e Migração de Ferramenta

Pesquisador	Lude Ribeiro
Requisito	Implementação da API (Views/Serializers)
Ferramenta	ChatGPT → Migração para GitHub Copilot

1. Objetivo: Implementar os endpoints da API do carrinho.

2. Problema (Crise): O ChatGPT entrou em contradição cíclica ao tentar corrigir um erro 400 Bad Request, sugerindo mudanças desnecessárias na herança das classes.

3. Decisão Metodológica (Pivot): Interrupção do uso do ChatGPT Web. Migração para o **GitHub Copilot Agent** no VS Code para aproveitar o contexto local (`@workspace`).

Entrada 3: Implementação de Lógica e Refatoração

Pesquisador	Lude Ribeiro
Requisito	Regras de Negócio (Estoque, Preço)
Ferramenta	GitHub Copilot Agent

1. Interação (Prompt):

> "@workspace Implemente a lógica de `add_item` com validação de estoque e garanta atomicidade."

2. Resultado: A IA implementou corretamente o uso de `transaction.atomic` e `select_for_update`, além de extrair a validação para um método privado, melhorando a manutenibilidade.

Entrada 4: Testes e Cobertura

Pesquisador	Lude Ribeiro
Requisito	Testabilidade e Cobertura
Resultado	92% de Cobertura no Módulo

1. Incidente: Erro de configuração nos testes (`RuntimeError`) pois o app não estava no `settings_test.py`.

2. Correção: O Copilot identificou a ausência do app e sugeriu o patch correto. Após a correção, a cobertura do módulo `cart` atingiu 92%, superando a baseline.