Universidade Católica do Salvador
Bacharelado em Engenharia de Software

Antônio Victor de Almeida Palma,
Brian Friedrich dos Santos Schultz,
Jean Alves Silva,
Lucas Nascimento da Silva,
Yuri Figueiredo Ribeiro.

O Impacto dos Agentes de Desenvolvimento Baseados em Inteligência Artificial no Processo de Criação de Software: Um Estudo Comparativo Entre Abordagens Vibe Coding e Tradicionais.

Salvador
2025

Antônio Victor de Almeida Palma,
Brian Friedrich dos Santos Schultz,
Jean Alves Silva,
Lucas Nascimento da Silva,
Yuri Figueiredo Ribeiro.

O Impacto dos Agentes de Desenvolvimento Baseados em Inteligência Artificial no Processo de Criação de Software: Um Estudo Comparativo Entre Abordagens Vibe Coding e Tradicionais.

Trabalho de Conclusão de Curso apresentado à Universidade Católica do Salvador como parte dos requisitos necessários para a obtenção do Título de Engenheiro de Software.
Orientador: Prof. Haroldo Peon

Universidade Católica do Salvador

Salvador
2025

Antônio Victor de Almeida Palma,
Brian Friedrich dos Santos Schultz,
Jean Alves Silva,
Lucas Nascimento da Silva,
Yuri Figueiredo Ribeiro.

O Impacto dos Agentes de Desenvolvimento Baseados em Inteligência Artificial no Processo de Criação de Software: Um Estudo Comparativo Entre Abordagens Vibe Coding e Tradicionais.

Trabalho de Conclusão de Curso apresentado à Universidade Católica do Salvador como requisito parcial para a obtenção do título de Engenheiro de Software.

Salvador, 5 de dezembro de 2025
Banca Examinadora:

Prof. Haroldo Peon
Universidade Católica do Salvador
Orientador

Universidade Católica do Salvador

Prof. Me. Terceiro professor
Universidade Católica do Salvador

Agradecimentos

Eu, Antônio Victor de Almeida Palma, Agradeço a Deus pela força, sabedoria e fé que me sustentaram nesta jornada. Aos meus pais, por todo amor, paciência e apoio incondicional, que me ensinaram o valor do esforço e da dedicação. A vocês, devo não apenas este trabalho, mas todas as minhas conquistas.

Eu, Brian Friedrich dos Santos Schultz, gostaria de expressar meus sentimentos de gratidão aos meus colegas de turma e aos meus professores por terem compartilhado tanto conhecimento até aqui. Considero essa contribuição fundamental para todos que buscam impactar positivamente a sociedade. Agradeço ao caro leitor pelo tempo dedicado e desejo que faça bom proveito dos conhecimentos aqui apresentados.

Eu, Jean Alves Silva, agradeço primeiramente a Deus, por me conceder força, fé e sabedoria para superar os desafios e alcançar esta conquista. À minha família, expresso profunda gratidão pelo amor, apoio e incentivo incondicional em todos os momentos. Aos professores, agradeço pela dedicação, paciência e compromisso em compartilhar conhecimento e contribuir para minha formação.

Eu, Lucas Nascimento da Silva, gostaria de agradecer, primeiramente, a Deus, que me permitiu chegar nesse momento, agradecer também à minha família, que me deu todo o apoio necessário para que eu pudesse estar aqui, aos meus amigos, que tornaram essa jornada um pouco mais leve, e não menos importante, à própria UCSal.

Eu, Yuri Figueiredo Ribeiro, agradeço ao meu orientador, Professor Haroldo Peon, pela sua orientação precisa, paciência inestimável e pelo conhecimento compartilhado. Suas críticas construtivas e seu incentivo constante foram fundamentais para que nós pudéssemos superar os desafios e concluir esta pesquisa com rigor e qualidade. Agradeço de forma muito especial à minha família, o alicerce de toda a minha jornada. Em particular, à minha mãe, Patricia Figueiredo Ribeiro, por seu amor incondicional, por acreditar em mim em todos os momentos. À minha avó, Eliete Andrade Figueiredo, por todo o carinho, sabedoria e pelo apoio fundamental que sempre me ofereceu e por ser minha maior fonte de inspiração e força. Sem o suporte emocional e a compreensão de vocês, esta conquista não seria uma realidade.

Gostaríamos de agradecer a todos professores que nos acompanharam nessa jornada e fizeram a diferença em nossa vida e acredito que na vida de todos os alunos que por eles passaram. Por fim, nosso coordenador, Prof. Osvaldo Requião Melo.

”Somos pelegrinos nesta terra...não sabemos até quando! Devemos encarar a vida, não com tristeza, mas com seriedade e esperança” (São João Paulo II)

Resumo

Os avanços tecnológicos têm gerado incertezas quanto ao futuro do desenvolvimento de software tradicional, ao mesmo tempo em que abrem espaço para novas abordagens ainda pouco exploradas. Este documento apresenta um estudo comparativo entre tecnologias consolidadas no mercado e soluções ainda pouco conhecidas, como os agentes de inteligência artificial e o desenvolvimento de software vibe coding. O objetivo principal é analisar o impacto dessas abordagens no processo de criação de software, identificando potenciais benefícios, limitações e caminhos para o futuro do desenvolvimento tecnológico, tanto pela elaboração de uma aplicação simples e funcional utilizando ambos os métodos, comparando-os e documentando todo o processo, quanto com a participação de profissionais da área e suas percepções sobre o tema através de um questionário com perguntas diretas e objetivas.

Palavras-Chave: 1. Vibe Coding. 2. Agentes de IA. 3. Estudo Comparativo.

Abstract

Technological advances have generated uncertainties regarding the future of traditional software development, while also opening space for new and still little-explored approaches. This document presents a comparative study between well-established market technologies and lesser-known solutions, such as artificial intelligence agents and vibe coding software development. The main objective is to analyze the impact of these approaches on the software creation process, identifying potential benefits, limitations, and pathways for the future of technological development. This is achieved both through the creation of a simple and functional application using both methods—comparing them and documenting the entire process—and through the participation of professionals in the field and their perceptions on the topic, gathered via a questionnaire with direct and objective questions.

Keywords: 1. Vibe Coding. 2. AI agents. 3. Comparative Study.

Lista de ilustrações

Lista de Siglas e Abreviaturas

API	*Application Programming Interface*
BES	Bacharelado em Engenharia de Software
IA	Inteligência Artificial
LLM	*Large Language Models*
MVP	*Minimum Viable Product*

Sumário

1 Introdução

O processo de criação de software atravessa uma transformação crucial para o futuro, impulsionada pela maturação da Inteligência Artificial (IA) Generativa. Historicamente, o cenário do desenvolvimento foi marcado por uma dicotomia clara: de um lado, as abordagens tradicionais, baseadas em codificação manual, que oferecem flexibilidade e poder ilimitados, mas exigem alto conhecimento técnico e longos ciclos de produção; de outro, a ascensão das plataformas *low-code* e *vibe coding*, que visam a ”democratização”do desenvolvimento, permitindo que usuários de negócios (os chamados *citizen developers*) criem aplicações funcionais através de interfaces visuais intuitivas (Baldow et al. 2024).

Contudo, a recente emergência de agentes de desenvolvimento baseados em IA, como modelos de linguagem de larga escala (LLMs) treinados para código, age como um catalisador que impacta *ambos* os paradigmas simultaneamente.

No desenvolvimento tradicional, ferramentas como o GitHub Copilot já demonstram um impacto empírico significativo. Elas atuam como “colaboradores” que aumentam a produtividade, automatizando tarefas repetitivas, gerando blocos de código complexos e liberando os desenvolvedores para se concentrarem em desafios mais criativos. Por exemplo, em um experimento controlado, desenvolvedores com acesso ao Copilot completaram uma tarefa 55,8% mais rápido do que aqueles sem a ferramenta (Peng et al. 2023).

Em outro estudo realizado com estudantes trabalhando em código legado, o Copilot permitiu reduzir o tempo de codificação e diminuir o esforço necessário para escrever manualmente partes do sistema (Shihab et al. 2025).

Paralelamente à evolução dos assistentes de código convencionais, agentes de desenvolvimento mais avançados têm impulsionado o surgimento e a popularização do vibe coding, um paradigma em que o programador atua principalmente como descritor de intenções enquanto o agente de IA gera, refatora e valida o código de forma autônoma. Esse modelo representa uma ruptura significativa em relação ao fluxo tradicional, no qual o desenvolvedor precisa manipular o código diretamente, linha a linha. Segundo análises recentes, a combinação entre agentes generativos e ambientes de desenvolvimento está ”elevando o patamar de automação e velocidade nunca antes visto no ciclo de produção de software”(Microsoft 2024). Dessa forma, tarefas antes altamente técnicas, como integração de módulos, escrita de interfaces complexas ou otimização de algoritmos, podem ser delegadas ao agente enquanto o humano supervisiona, valida e ajusta o comportamento desejado.

Esse movimento levanta uma questão crítica: estamos diante de uma convergên-

cia entre o desenvolvimento tradicional e o *vibe coding*, ou de uma divergência que estabelece dois paradigmas distintos? Alguns analistas argumentam que os agentes de IA estão tornando o processo tradicional tão acelerado e automatizado que ele pode se aproximar da experiência de *vibe coding*, reduzindo drasticamente a quantidade de código escrito manualmente (Peng et al. 2023). Por outro lado, há quem sustente que o *vibe coding* representa um salto qualitativo: enquanto o desenvolvimento tradicional assistido por IA ainda exige participação ativa em decisões arquiteturais, o *vibe coding* desloca o foco quase inteiramente para a descrição de requisitos e para a interação em linguagem natural com o agente (Cavalcante 2025). Essa tensão evidencia uma ”paisagem de transformação acelerada”(TechRadar 2024) que precisa ser compreendida.

Assim, o objetivo central deste Trabalho de Conclusão de Curso é comparar, de forma sistemática, o impacto dos agentes de desenvolvimento baseados em IA nos processos *vibe coding* e tradicionais, com foco em aspectos como eficiência, qualidade do código gerado e curva de aprendizado para desenvolvedores. A relevância deste estudo está na necessidade emergente de organizações e profissionais decidirem qual paradigma — quando amplificado por agentes de IA — apresenta maior potencial estratégico para o desenvolvimento de software no futuro próximo, considerando tanto ganhos de produtividade quanto riscos associados, como dependência tecnológica, segurança e manutenibilidade.

1.1 Objetivo Geral

O objetivo geral deste presente trabalho de pesquisa é analisar e comparar o impacto dos Agentes de Desenvolvimento Baseados em Inteligência Artificial (AI-Powered Development Agents) no processo de criação de software, contrastando as metodologias de desenvolvimento tradicional e *vibe coding*.

Para atingir este objetivo, o estudo será centrado no desenvolvimento de um MVP (Produto Mínimo Viável) de um aplicativo genérico, que será construído em dois cenários distintos: na abordagem tradicional, utilizando linguagens de programação convencionais com o suporte dos agentes de IA, e na abordagem *vibe coding*, empregando plataformas que eliminam a necessidade de escrever código, também com o auxílio da Inteligência Artificial.

Este estudo busca explorar como a intervenção dos agentes de IA influencia a produtividade e a velocidade de entrega em ambas as metodologias, em comparação com os fluxos de trabalho tradicionais de desenvolvimento de software. Além disso, pretende-se validar a viabilidade e a eficácia de cada abordagem, avaliando métricas cruciais como o tempo de desenvolvimento, a complexidade da manutenção, a qualidade do código/estrutura gerada e a usabilidade do produto final para o usuário

comum. Todo o projeto será orientado pela experiência do usuário, garantindo uma comparação justa e abrangente da aplicabilidade e das limitações de cada metodologia no contexto da engenharia de software.

1.2 Objetivos Específicos

- Desenvolver um aplicativo genérico em dois ambientes distintos: um utilizando uma abordagem de desenvolvimento tradicional, e o outro utilizando plataforma *vibe coding* com o auxílio de agentes de IA também com o suporte desses agentes.
- Mensurar e comparar a produtividade e a eficiência do processo de criação de software em ambas as abordagens (*vibe coding* e tradicional), registrando o tempo total de desenvolvimento, o esforço da equipe e a complexidade de implementação das funcionalidades básicas do aplicativo.
- Analisar a qualidade e a manutenibilidade das soluções desenvolvidas, avaliando a robustez da arquitetura gerada pelo ambiente *vibe coding* e pelo código tradicional, bem como a facilidade e o custo para a aplicação de futuras modificações e updates.
- Avaliar a confiabilidade de ambasmversões do aplicativo genérico, por meio de testes automatizados, para determinar se a metodologia de desenvolvimento (*vibe coding* Vs. tradicional) impõe limitações ou oferece vantagens significativas na interação final com o usuário.

1.3 Estrutura do Trabalho

Neste capítulo foram apresentados o contexto, a motivação, objetivos, estratégias de pesquisa, e contribuições associados a este trabalho de conclusão de curso.

2 Fundamentação Teórica

Este capítulo aborda os conceitos fundamentais que servem de base para o desenvolvimento deste trabalho de pesquisa. Nele, encontram-se referências e inspirações que orientaram tanto a redação do tema quanto a criação da solução proposta do tema ”O Impacto dos Agentes de Desenvolvimento Baseados em Inteligência Artificial no Processo de Criação de Software: Um estudo comparativo entre abordagens vibe-coding e tradicionais.”.

2.1 Histórico das formas de programação

O desenvolvimento de software é um campo ”jovem adulto”, mas suas raízes se estendem desde os primórdios do conceito de máquina programável. A evolução das formas de programação é um reflexo direto da busca humana por maior abstração e eficiência, um caminho que resulta nas abordagens modernas de *vibe coding* baseados em IA.

A primeira grande aceleração no desenvolvimento de máquinas de calcular e, consequentemente, das formas de as programar, foi impulsionada por necessidades militares. Como Ceruzzi (Ceruzzi 2003) aponta, o computador digital eletrônico moderno nasce em um contexto de conflito. A Segunda Guerra Mundial (1939-1945) exigiu máquinas capazes de realizar cálculos balísticos complexos e, crucialmente, de quebrar códigos criptográficos em tempo hábil, como ilustra o filme (Tyldum 2014). Na Grã-Bretanha, a criação do Colossus foi essencial para decifrar as comunicações alemãs, exigindo uma forma inicial de ”programação”através da configuração de chaves e painéis. Nos Estados Unidos, o ENIAC (Electronic Numerical Integrator and Computer) foi desenvolvido para calcular tabelas de tiro. Inicialmente, a ”programação”do ENIAC era um processo laborioso e físico, configurado manualmente. Essas máquinas, criadas para fins bélicos, estabeleceram a necessidade urgente de desenvolver formas mais rápidas e lógicas de comunicação com o hardware. Campbell-Kelly e Aspray (Campbell-Kelly e Aspray 1996) destacam que a pressão do tempo de guerra não apenas financiou a pesquisa, mas também criou uma cultura de colaboração e desenvolvimento acelerado, cujo legado se tornou a base da indústria da informação.

O desenvolvimento de software é, essencialmente, a história da migração de uma interação de baixo nível (próxima ao hardware) para uma interação de alto nível (próxima à linguagem humana). A forma mais primitiva de programação utilizava sequências binárias de 0 e 1 (linguagem de máquina). A verdadeira revolução na produtividade veio com as Linguagens de Alto Nível (Terceira Geração), como o FORTRAN

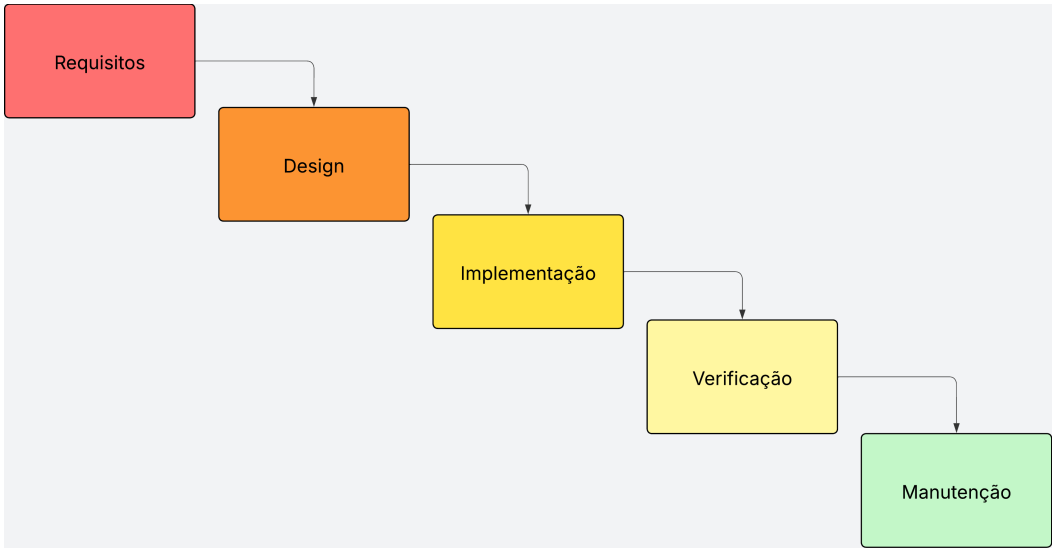
e o COBOL, que permitiram que os desenvolvedores expressassem a lógica de programação utilizando sintaxe próxima ao inglês. Esta é a essência do que se convencionou chamar de abordagem de desenvolvimento tradicional. O paradigma tradicional evoluiu significativamente com a adoção da Programação Orientada a Objetos (POO) na década de 1980, aumentando a complexidade gerenciável do código, mas ainda exigindo conhecimento aprofundado de sintaxe e lógica de programação. (Campbell-Kelly e Aspray 1996)

No século XXI, a complexidade crescente das demandas de software e a necessidade de facilitar a criação de aplicações levaram ao questionamento sobre o limite da abstração. A resposta se manifesta na ascensão das plataformas low-code e no-code. Estas abordagens representam o passo final no caminho histórico da abstração, permitindo que a lógica de negócios seja expressa por meio de interfaces visuais. Este movimento não é um abandono do tradicional, mas sim uma bifurcação onde o no-code visa a produtividade e a inclusão. A chegada dos Agentes de Desenvolvimento Baseados em Inteligência Artificial e o *vibe coding* é o próximo e mais radical capítulo. Eles não apenas abstraem a sintaxe (como o no-code), mas buscam abstrair o próprio processo de desenvolvimento, transformando uma simples descrição de necessidade em um produto funcional. Estes agentes representam a mais alta forma de automatização na história da programação, com desafios ainda pouco conhecidos e explorados pela comunidade da área (Ray 2025; 10; Meske et al. 2025; Sarkar e Drosos 2025), sendo o foco comparativo deste estudo.

2.2 Metodologias Tradicionais e o Desenvolvimento de Software Clássico

As metodologias tradicionais de desenvolvimento de software, frequentemente categorizadas como modelos de processo quase obrigatórios, representam a aplicação dos princípios da engenharia clássica ao domínio da criação de programas. Esta abordagem é fundamentalmente orientada por planejamento, sequência lógica e pela busca de previsão e controle de todo o projeto. O modelo mais conhecido desta filosofia é o Modelo Cascata (Waterfall), formalizado por Winston W. Royce (Royce 1970). Embora Royce tenha alertado sobre os riscos de uma aplicação pura do modelo, sua descrição de fases sequenciais e distintas tornou-se, com o tempo, a base paradigmática para inúmeros projetos. A premissa central, como detalhado por Roger S. Pressman (Pressman e Maxim 2020), que cada fase deve ser completamente finalizada e documentada antes do início da seguinte, criando um contrato claro entre etapas e minimizando ambiguidades, não permitindo a ”flutuação”entre etapas, ocasionando em rigidez interna.

Figura 1 – Exemplo Modelo Cascata



Autoria Própria

Este foco na documentação e no planejamento meticuloso não se limitou ao Cascata. Uma evolução significativa foi o Modelo em Espiral, proposto por Barry Boehm (Boehm 1988). Este modelo introduziu ciclos iterativos, mas manteve um núcleo fortemente alinhado com a gestão de riscos e o planejamento deliberado, características centrais do desenvolvimento clássico. Segundo Boehm, a tomada de decisão econômica e a avaliação sistemática de riscos são atividades fundamentais para o sucesso de projetos de software complexos.(Boehm 1981)

No entanto, a rigidez inerente a estas abordagens tornou-se alvo de críticas contundentes, particularmente com o advento de metodologias mais ágeis. Autores como Robert C. Martin em ”Código Limpo”e Kent Beck argumentam que, em ambientes de negócio voláteis, a natureza sequencial e de longo ciclo de feedback dos modelos tradicionais frequentemente resulta em produtos finalizados que já não correspondem às necessidades reais do utilizador (Martin 2009; Beck 2004). A concentração da fase de testes no final do ciclo frequentemente leva à descoberta tardia de erros fundamentais, cuja correção se torna proibitivamente custosa. Assim, as metodologias tradicionais e o desenvolvimento de software clássico representam um legado duradouro de disciplina e rigor, cujo valor é inquestionável em projetos de alta criticidade e requisitos fixos, mas cuja aplicação cega em contextos de incerteza. Dessa forma, faz-se necessário o uso de tecnologias ainda mais específicas para cada cenário e um estudo prévio cada vez mais detalhado.

2.2.1 O Presente e o Futuro das Metodologias de Desenvolvimento de Software

O panorama atual do desenvolvimento de software é caracterizado por uma controversia entre a disciplina dos processos tradicionais e a flexibilidade exigida pelo mercado contemporâneo. As metodologias clássicas, com foco em planejamento sequencial e documentação abrangente, enfrentam críticas por sua rigidez em ambientes onde a mudança é constante. Críticas de autores renomados já são comuns, como Robert C. Martin apresenta em ”Código Limpo”(Martin 2009). Esta rigidez manifesta-se principalmente na dificuldade de incorporar mudanças tardias de requisitos e na concentração das atividades de validação nas fases finais do ciclo, tornando a correção de erros proibitivamente custosa - um princípio que Barry Boehm já havia demonstrado em ”Software Engineering Economics”. (Boehm 1981)

O futuro, no entanto, está a ser redefinido pela convergência entre desenvolvimento de software e a inteligência artificial. O processo de desenvolvimento apresenta uma interrupção do curso normal, que questiona os fundamentos tanto das abordagens tradicionais quanto das ágeis. Estas ferramentas prometem acelerar drasticamente a produção de código, mas simultaneamente levam a questões críticas sobre a manutenção da qualidade e a possível erosão do conhecimento fundamental, segundo Martin Fowler. O legado das abordagens tradicionais permanecerá vital em domínios de alta criticidade, porém o desafio emergente será integrar ferramentas de IA em processos estruturados, garantindo que a velocidade e a flexibilidade não comprometam a robustez, segurança e qualidade final do software. (Fowler 2018)

2.3 Fundamentos da Inteligência Artificial e Agentes Inteligentes

A Inteligência Artificial (IA) constitui um campo da computação dedicado ao desenvolvimento de sistemas capazes de executar tarefas que exigem algum tipo de inteligência humana, como raciocínio, aprendizagem e tomada de decisão. A compreensão moderna da IA se apoia na definição apresentada por Russell e Norvig (2010), segundo a qual o objetivo central desse domínio é criar agentes racionais, ou seja, sistemas que percebem o ambiente, processam informações e agem da forma como foram ensinados. Esse conceito evidência a autonomia e a capacidade adaptativa, elementos essenciais para entender o papel da IA no desenvolvimento de software. (Russell e Norvig 2010)

No percurso histórico da área, Alan Turing teve papel importante ao propor, já em meados do século XX, que máquinas poderiam demonstrar comportamento inteligente observável, contribuindo para uma visão da IA focada na emulação da cognição hu-

mana. Apesar dessa inspiração inicial, a evolução técnica — especialmente com os avanços em aprendizado de máquina e aprendizado profundo — ampliou significativamente o potencial dos sistemas atuais. Goodfellow, Bengio e Courville (2016) mostram que o *deep learning* permitiu o reconhecimento de padrões complexos e a generalização eficiente, criando condições para o surgimento de agentes mais robustos e capazes de operar em tarefas de alta complexidade.(14; Turing 1950)

Nesse contexto, os agentes inteligentes tornaram-se a base das estratégias modernas de criação de software. Mais do que simples assistentes, esses agentes conseguem interpretar instruções em linguagem natural, gerar código, executar análises, propor correções e ajustar soluções de maneira contínua. Shoham destaca que, em arquiteturas de múltiplos agentes, tais sistemas podem dividir responsabilidades, cooperar entre si e coordenar processos completos (Shoham e Leyton-Brown 2008). Características essas que sustenta práticas contemporâneas como o vibe coding . Assim, os fundamentos da IA e dos agentes inteligentes explicam por que ferramentas baseadas nesse paradigma têm produzido impactos significativos nas formas tradicionais de desenvolvimento, ao permitir que o processo seja conduzido por modelos autônomos que interpretam intenções e operacionalizam soluções.

2.3.1 Programação Tradicional vs. Programação Assistida por IA

A programação tradicional caracteriza-se por um processo totalmente controlado pelo desenvolvedor, que precisa dominar linguagens, algoritmos e práticas de engenharia para conduzir cada etapa do projeto. Essa abordagem, amplamente descrita por (Sommerville 2016), envolve um fluxo sequencial e manual, no qual as ferramentas servem apenas como apoio operacional. Com o avanço da Inteligência Artificial, surgiu uma fase intermediária conhecida como programação assistida por IA, marcada principalmente por sistemas capazes de sugerir trechos de código e automatizar tarefas repetitivas. Modelos como o GitHub Copilot, analisados por (Chen et al. 2021), introduziram a capacidade de prever padrões e acelerar a escrita, embora sua atuação fosse limitada a recomendações locais. Os estudos iniciais desses autores mostram que os desenvolvedores aceitavam apenas 20% a 30% das sugestões geradas, o que evidencia que a IA ainda funcionava como ferramenta complementar.

Esse avanço foi possibilitado pelo progresso do aprendizado profundo, discutido por Goodfellow, Bengio e Courville (14), que permitiu que modelos compreendessem estruturas de código de maneira mais sofisticada. No entanto, essa etapa pré-vibe coding não envolvia agentes autônomos. Como destaca Shoham (Shoham 1993), agentes inteligentes exigem capacidades de percepção, raciocínio e tomada de decisão mais amplas, características que ainda não estavam presentes nos assistentes dessa fase. Assim, a programação assistida por IA representou um momento de transição,

reduzindo esforço e acelerando tarefas, mas mantendo o programador como centro de todas as decisões. Esse período preparou o terreno para abordagens mais avançadas, nas quais agentes de IA passaram a desempenhar funções de planejamento e execução mais complexas.

2.4 *Vibe Coding*

O conceito de *vibe coding* surge na evolução das práticas de desenvolvimento de software mediadas por IA, representando uma mudança em relação às abordagens tradicionais e mesmo às primeiras formas de programação assistida por IA. Em termos gerais, o *vibe coding* pode ser definido como um modelo de desenvolvimento no qual programadores e não programadores interagem com agentes inteligentes por meio de linguagem natural, expressando intenções de maneira fluida, enquanto a IA ”traduz”essas intenções em softwares, com arquitetura, código, testes e documentação. Essa forma de interação, baseada em conversação contínua, desloca o foco do programador da escrita manual para a mediação de objetivos e revisões, transformando o processo em uma experiência mais intuitiva e centrada na intenção do usuário.

2.4.1 Ferramentas e Mecanismos

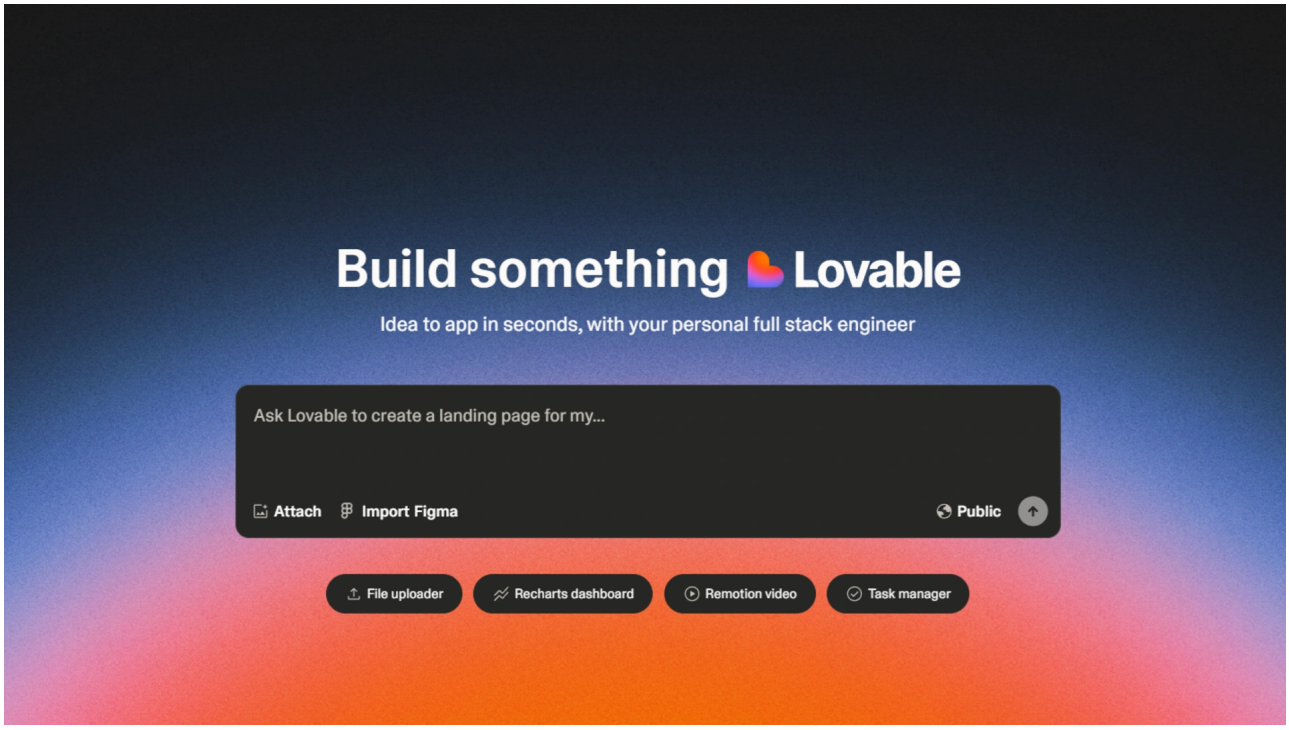
Esse novo paradigma de criação de software em que a interação entre desenvolvedores e sistemas de inteligência artificial ocorre predominantemente por meio de linguagem natural. Diferentemente da programação tradicional, na qual o programador manipula diretamente o código, esse mecanismo delega grande parte da mediação entre intenção e implementação aos agentes de IA (Sarkar e Drosos 2025). Esse modelo equivale a “programar por conversa”, pois o usuário descreve o que deseja e os agentes inteligentes convertem essas diretrizes em artefatos concretos de software. Cientistas ampliam essa perspectiva ao argumentar que o *vibe coding* representa uma reconfiguração fundamental do processo de mediação de intenção, deslocando o foco do como se programa para o que deve ser construído. Assim, ele não se limita a incorporar modelos de linguagem às IDEs, mas inaugura uma nova forma de produzir sistemas digitais. (Meske et al. 2025)

Esse paradigma é possibilitado por um conjunto sofisticado de ferramentas e mecanismos internos. (Ray 2025) explica que o *vibe coding* é sustentado por agentes de desenvolvimento capazes de atuar como arquitetos, planejadores, geradores de código e verificadores de consistência. Esses agentes trabalham de forma autônoma, decompondo objetivos de alto nível em tarefas menores e produzindo implementações estruturadas, muitas vezes incluindo diretórios completos, APIs, interfaces, testes automatizados e documentação. Em vez de apenas sugerir pequenas porções de có-

digo, os agentes executam o *scaffolding* integral de aplicações, aproximando-se mais de um “colega de equipe digital” do que de uma ferramenta de apoio. (Horvat 2025; Geng et al. 2025)

Entre as plataformas que operam desta forma, o Lovable aparece como um exemplo notável. Ele funciona como um ambiente completo de desenvolvimento orientado por IA que permite ao usuário construir sistemas inteiros descrevendo apenas suas intenções em linguagem natural. Ao receber uma solicitação, o Lovable elabora a arquitetura, gera os arquivos, popula diretórios, configura o *backend* e o *frontend* e estabelece padrões de projeto coerentes com o objetivo proposto. O usuário interage com o sistema por meio de uma conversa contínua, e a IA mantém o histórico contextual necessário para modificar, expandir ou refatorar o projeto conforme novas instruções surgem. Esse funcionamento é coerente com o modelo descrito por (Sarkar e Drosos 2025), no qual a criação de software passa a ser guiada por diálogo, e também com a visão de (Ray 2025), que destaca os agentes especializados atuando em conjunto para gerar soluções completas.

Figura 2 – Interface do Lovable



Automasix 2025

2.4.2 Vantagens e Desafios

Um conjunto de vantagens expressivas justificam seu rápido crescimento no ecossistema de desenvolvimento de software, mas também traz desafios que ainda limitam

sua consolidação plena. Entre os principais benefícios, destaca-se a drástica redução da carga cognitiva do desenvolvedor, já observada em estudos empíricos. (10) mostram que o *vibe coding* possibilita ao programador concentrar-se na concepção do sistema e na formulação de objetivos, enquanto agentes inteligentes assumem atividades mecânicas e estruturais, como geração de código, configuração de ambientes e produção de testes. Essa dinâmica acelera o ciclo de desenvolvimento, reduz gargalos e facilita experimentação rápida, o que Ray identifica como um dos pilares para o aumento de produtividade observado em equipes que utilizam agentes baseados em IA. Além disso, a interação por linguagem natural democratiza o acesso ao desenvolvimento, permitindo que iniciantes ou profissionais de áreas distintas contribuam para a criação de software sem domínio avançado de sintaxe ou frameworks, como argumentam Sarkar e Drosos ao descrever como uma forma de "programação conversacional". (Ray 2025; Sarkar e Drosos 2025)

No entanto, apesar de suas vantagens, o *vibe coding* enfrenta desafios significativos. Um deles é a dependência da clareza e precisão das instruções fornecidas pelo usuário, uma vez que ambiguidades textuais podem levar a interpretações incorretas e implementações inadequadas. (Meske et al. 2025) alertam que a mediação de intenção, embora poderosa, é sensível a nuances linguísticas, o que cria novas responsabilidades para o desenvolvedor na formulação dos requisitos. Outro desafio recorrente envolve a necessidade de validação rigorosa do código gerado. Embora os agentes sejam capazes de produzir sistemas funcionais, ainda podem ocorrer falhas estruturais, inconsistências lógicas e desvios arquiteturais, exigindo supervisão humana constante. Além disso, a manutenção de contexto em projetos longos continua sendo um ponto de limitação, conforme destacado por Ray , que observa que mesmo pipelines multiagente podem perder coerência ao longo de ciclos extensos de modificação. Por fim, a manutenibilidade é um fator a ser considerado, muitos desenvolvedores alegam não conseguir realizar manutenções em sistemas gerados por agentes de IA, seja más práticas de programação, seja pela abstração gerada. (Ray 2025)

3 Metodologia

3.1 Objetivos da Pesquisa

O objetivo desta pesquisa é analisar como agentes de desenvolvimento baseados em Inteligência Artificial influenciam o processo de criação de software, comparando a abordagem do *vibe coding* com métodos tradicionais de programação. Busca-se compreender as percepções, experiências e formas de trabalho de desenvolvedores que utilizam agentes de IA, identificando benefícios, limitações e impactos percebidos no fluxo de desenvolvimento.

A pesquisa adota uma perspectiva exploratória, concentrada na investigação das interações entre humanos e agentes inteligentes. Pretende-se compreender como a mediação da IA afeta a tomada de decisão, a autonomia no desenvolvimento, a compreensão do código e a dinâmica de co-criação entre humano e sistema inteligente.

Os resultados esperados incluem a ampliação do entendimento sobre a transformação das práticas de engenharia de software na era da IA, contribuindo para que profissionais, pesquisadores e instituições de ensino possam avaliar e aperfeiçoar o uso dessas tecnologias em diferentes contextos.

3.2 Natureza da Pesquisa

Esta pesquisa possui natureza aplicada, pois busca gerar conhecimento orientado à prática e ao aprimoramento de processos reais de desenvolvimento de software. A investigação concentra-se em compreender como ferramentas atuais de IA podem modificar e potencializar abordagens tradicionais e emergentes, como o *vibe coding*.

3.3 Abordagem do Trabalho

A abordagem adotada é qualitativa, adequada para investigar fenômenos complexos que envolvem percepções, experiências subjetivas e interações humanas com tecnologia. Segundo (Creswell 2013), a pesquisa qualitativa permite analisar fenômenos em seu contexto natural, interpretando significados atribuídos pelos participantes.

Esse tipo de abordagem é apropriado para compreender como desenvolvedores vivenciam a utilização de agentes de IA, quais desafios e benefícios percebem e como essas ferramentas alteram práticas tradicionais de desenvolvimento. Não se busca quantificação de desempenho, mas sim interpretação aprofundada das experiências relatadas.

3.4 Procedimentos

3.4.1 População e Amostra

- **População-alvo**: Desenvolvedores de software atuantes ou em formação.
- **Critérios de Inclusão**: Participantes com experiência prévia em desenvolvimento de software e contato com ferramentas de IA generativa ou agentes de desenvolvimento.
- **Amostragem**: Amostragem intencional, buscando diversidade de perfis quanto a nível de habilidade, área de atuação, tempo de experiência e grau de exposição a práticas tradicionais e ao *vibe coding*.

3.4.2 Instrumentos de Coleta

A coleta de dados será realizada por meio de:

- **Entrevistas semiestruturadas**, permitindo explorar percepções e experiências individuais;
- **Questionário aberto**, para ampliar o alcance da coleta e obter múltiplas perspectivas;
- **Testes Automatizados**, registrando observações sobre práticas o funcionamento da aplicação em ambos os modelos de desenvolvimento.

3.4.3 Análise dos Dados

Os dados serão analisados mediante **análise temática**, permitindo identificar padrões de significado, categorias e temas recorrentes nas falas dos participantes. O processo envolverá codificação inicial, agrupamento semântico e interpretação dos temas emergentes.

3.5 Discussão Esperada

A pesquisa pretende compreender como agentes de desenvolvimento baseados em IA influenciam o processo de criação de software, comparando percepções, práticas e impactos entre usuários do *vibe coding* e do desenvolvimento tradicional. Espera-se identificar como esses agentes afetam o fluxo de trabalho, a autonomia, a compreensão do código e o processo de tomada de decisão, contribuindo para o debate sobre o futuro das práticas de engenharia de software.

4 Construção da Aplicação

4.1 Aplicação Genérica

Um sistema genérico de gerenciamento de finanças pessoais foi desenvolvido com o objetivo de servir como base comparativa entre o desenvolvimento tradicional e o desenvolvimento baseado em *vibe coding*. Esse sistema permite que o usuário registre despesas, receitas, metas financeiras e projeções, oferecendo uma experiência completa de organização e acompanhamento das finanças do dia a dia. Trata-se de uma aplicação mobile robusta, com interface intuitiva e análise visual dos dados, permitindo que o usuário compreenda como seus recursos estão sendo distribuídos ao longo do tempo.

Em ambas as abordagens, o sistema permite realizar uma comparação entre os modos de desenvolvimento tradicional e *vibe coding*, já que ambas as versões possuem as mesmas funcionalidades, requisitos e cobertura de testes automatizados. Dessa forma, torna-se possível analisar diferenças significativas no fluxo de trabalho, no esforço exigido do desenvolvedor, na velocidade de implementação, na qualidade do código gerado e na experiência geral de construção da aplicação. Ao partir de um mesmo produto final, a análise se concentra não no resultado, mas nos caminhos percorridos para atingi-lo, permitindo avaliar com maior clareza as vantagens, limitações e características distintas de cada abordagem.

4.2 Prototipação das telas

O projeto teve início com a elaboração do design da aplicação no Figma. Essa etapa não se limitou aos aspectos estéticos, pois consistiu no momento de organização das ideias, definição dos fluxos da aplicação e visualização preliminar de como cada componente seria posteriormente convertido em código. Tal processo demandou uma atuação full-stack, ao integrar a concepção visual com o desenvolvimento lógico em Dart e a implementação das interfaces em Flutter. Verificou-se, ao final dessa etapa, a satisfação em observar o layout projetado sendo reproduzido no dispositivo móvel de maneira fiel ao planejado.

4.2.1 Construção Tradicional

4.2.1.1 Ferramentas e Tecnologias

Para o desenvolvimento do projeto, foi escolhido utilizar o framework Flutter. Essa decisão foi motivada por sua simplicidade, flexibilidade e ampla adoção no mercado. O Flutter permite criar interfaces dinâmicas e modernas com eficiência por meio de sua arquitetura baseada em blocos e widgets reutilizáveis, que facilitam a manutenção e evolução do código. Além disso, seu mecanismo de renderização próprio garante alto desempenho e fluidez nativa, enquanto a possibilidade de compilar para Android, iOS, Web e Desktop a partir de uma única base de código reduz significativamente o esforço de desenvolvimento. Esse framework comporta uma solução robusta para a criação rápida de aplicações sofisticadas e escaláveis. (Flutter Dev 2024)

4.2.1.2 Aspectos do desenvolvimento

O desenvolvimento do aplicativo demonstrou que a escrita manual de código oferece um maior entendimento da aplicação, permitindo ao desenvolvedor controlar cada etapa do processo, desde a estrutura dos widgets até as decisões de arquitetura. Mesmo com experiência prévia e certificações, o projeto exigiu constantes pesquisas, consultas à documentação oficial, fóruns e comunidades, o que contribuiu para o fortalecimento da capacidade analítica e da autonomia técnica da equipe.

Essa abordagem tradicional favoreceu uma maior personalização do produto também, já que cada funcionalidade foi construída de acordo com as necessidades específicas do aplicativo, sem limitações impostas por ferramentas automáticas, como o *vibe coding*. Além disso, o domínio sobre o código facilitou a antecipação de riscos, o planejamento mais consciente de futuras funcionalidades e a construção de um software mais consistente e escalável e de relativa facilidade de manutenção.

4.2.1.3 Conclusão

Embora o processo envolva desafios, como a configuração do ambiente, a resolução de erros e a curva de aprendizado do Flutter, essas etapas se mostraram fundamentais para consolidar boas práticas e amadurecer a visão sobre arquitetura e desenvolvimento. Assim, a prática do desenvolvimento manual não apenas aprimora a qualidade técnica do produto, mas também contribui para o crescimento profissional, tornando o desenvolvedor mais preparado para criar soluções sólidas, eficientes e alinhadas às necessidades reais do usuário e do negócio.

4.2.2 Construção Vibe Coding

4.2.2.1 Ferramentas e Tecnologias

Para o desenvolvimento do projeto também foi utilizada a plataforma Lovable, uma solução voltada para criação assistida de software por meio de IA. A escolha se deu por sua praticidade, flexibilidade e crescente adoção como ferramenta de apoio ao desenvolvimento moderno. O Lovable permite gerar e evoluir aplicações completas de forma dinâmica, a partir de instruções em linguagem natural, reduzindo a necessidade de intervenção manual em tarefas repetitivas. Além disso, sua capacidade de sugerir, refatorar e ajustar trechos de código em tempo real acelera o ciclo de desenvolvimento e facilita a manutenção contínua do projeto. Outro diferencial é a possibilidade de integrar testes automatizados e ajustes incrementais durante o processo, promovendo maior qualidade e consistência no resultado final. Dessa forma, o Lovable se apresenta como uma solução eficiente e robusta para criação rápida de aplicações completas, sofisticadas e escaláveis.(Lovable.dev 2025)

4.2.2.2 Aspectos do desenvolvimento

O desenvolvimento do aplicativo Expense Pixel Perfect evidenciou a importância da escrita manual de código como ferramenta central para o domínio técnico do projeto. Ao optar pela construção do sistema de forma artesanal sem dependência de geradores automáticos ou plataformas no-code o desenvolvedor pode compreender profundamente cada camada da aplicação, desde a organização estrutural dos componentes visuais até as decisões que impactam diretamente a arquitetura do software. Esse processo proporciona um controle maior sobre a lógica e sobre o comportamento da interface, resultando em um produto mais personalizado.

Mesmo possuindo experiência prévia em desenvolvimento de aplicações, o projeto exigiu um ciclo contínuo de aprendizado e atualização. Foram necessárias diversas pesquisas, análises de documentações oficiais, consultas a fóruns especializados e interações com comunidades de desenvolvimento. Esse movimento colaborativo contribuiu para o fortalecimento da capacidade analítica, resolução de problemas e autonomia técnica, competências essenciais em projetos de software.

4.2.2.3 Conclusão

Assim, o domínio sobre o código-fonte de forma tradicional favorece a antecipação de problemas futuros, como falhas estruturais, gargalos de desempenho e riscos de segurança. Esse entendimento permitiu um planejamento mais estratégico das próximas funcionalidades, bem como a construção de uma base sólida para escalabilidade. O resultado é um software mais organizado, modular, sustentável e com maior facili-

dade de manutenção, fatores determinantes para a longevidade e evolução contínua do projeto.

5 Coleta de Dados e Resultados

5.1 Metodologia de Coleta de Dados

Para a validação das hipóteses levantadas neste estudo, foi realizada uma coleta de dados quantitativa por meio da aplicação de um *survey* (questionário estruturado online). O instrumento de pesquisa foi distribuído estrategicamente em grupos acadêmicos de tecnologia e comunidades de desenvolvedores, com o objetivo de alcançar um público com conhecimento pertinente sobre os paradigmas de desenvolvimento de software abordados.

É importante ressaltar que a pesquisa seguiu rigorosos padrões éticos. A primeira seção do formulário apresentou o Termo de Consentimento Livre e Esclarecido (TCLE), garantindo aos participantes o anonimato, a confidencialidade das informações e a ciência de que os dados seriam utilizados exclusivamente para fins acadêmicos nesta análise comparativa entre o Desenvolvimento Tradicional e o Vibe Coding. Somente foram considerados válidos os questionários cujos participantes concordaram explicitamente com o termo.

5.2 Formulário avaliativo para entendimento sobre o assunto entre profissionais da área

O formulário foi desenvolvido com o propósito de coletar informações fundamentais para analisar a adoção, percepção e efetividade tanto do *Vibe Coding* quanto ao uso das ferramentas tradicionais no processo de desenvolvimento de software. Seu objetivo central é compreender como estudantes e profissionais da área utilizam essa abordagem, avaliando desde o impacto na produtividade e criatividade até a aplicabilidade em projetos reais de maior complexidade. O questionário encontra-se disponível para acesso através do link: <https://forms.gle/ejGxUcnyoXMgsZvn8>.

A estrutura do formulário contempla diversas seções que investigam, de forma sistemática, múltiplos aspectos do *Vibe Coding*. Entre eles estão a experiência prévia dos participantes, a percepção da produtividade diária, o impacto da ferramenta no foco e na motivação, bem como sua eficácia em fases específicas do desenvolvimento, como levantamento de requisitos, planejamento e manutenção de software. Além disso, o instrumento busca identificar o grau de confiança dos respondentes na utilização de ferramentas emergentes em projetos, permitindo uma visão abrangente sobre o potencial e os limites dessa abordagem emergente.

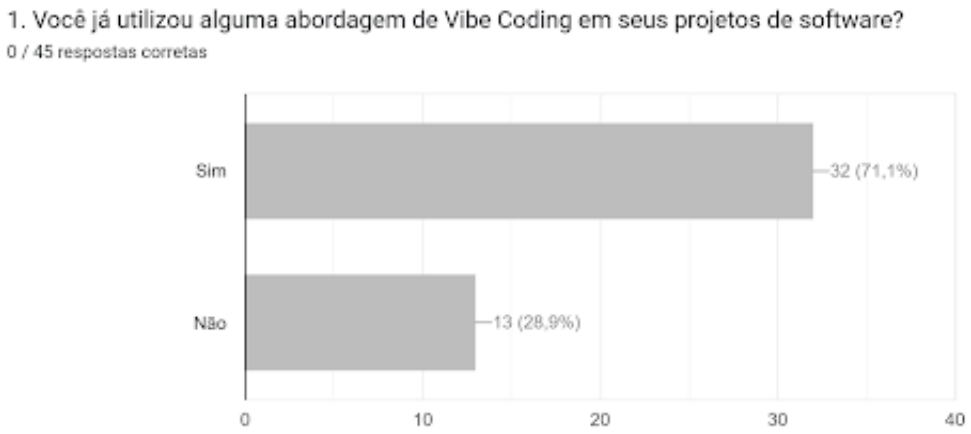
Os dados obtidos por meio dessa coleta serão essenciais para validar as hipóteses levantadas sobre o papel da IA generativa no ciclo de desenvolvimento de software. A análise das respostas permitirá identificar padrões de uso, percepções de valor e possíveis resistências, contribuindo para compreender de que forma essas tecnologias modernas podem ser integradas de maneira eficiente às práticas tradicionais. Ao mesmo tempo, os resultados auxiliarão na compreensão do cenário atual e na projeção de caminhos futuros para o uso híbrido entre métodos clássicos e ferramentas assistidas por IA.

Para contextualizar os respondentes e facilitar a interpretação das questões, o formulário foi acompanhado de explicações introdutórias sobre os conceitos-chave do estudo. A partir dessa coleta estruturada, busca-se construir uma base sólida para discutir a evolução do desenvolvimento de software na era da inteligência artificial, destacando de que forma essa nova abordagem pode otimizar processos, acelerar implementações e transformar a dinâmica entre desenvolvedores e ferramentas digitais.

5.3 Análise dos Gráficos

5.3.1 Gráfico 1: Adoção do Vibe Coding

Figura 3 – Gráfico 1: Adoção do Vibe Coding



Gerado pelo *Google Forms*

Pergunta: Você já utilizou alguma abordagem de Vibe Coding em seus projetos de software?

- **Sim:** 71,1% (32 respostas)

- **Não:** 28,9% (13 respostas)

Análise: Os dados revelam uma alta adesão das ferramentas de IA no cotidiano dos desenvolvedores entrevistados. Com mais de 70% dos respondentes afirmando já ter utilizado *Vibe Coding*, infere-se que essa tecnologia ultrapassou a fase de "inovação inicial" e já faz parte do ferramental prático da maioria dos estudantes e profissionais da área. Isso valida a relevância do estudo, dado que a tecnologia não é apenas teórica, mas amplamente aplicada.

5.3.2 Gráfico 2: Percepção de Produtividade Diária

Figura 4 – Gráfico 2: Percepção de Produtividade Diária

Gerado pelo *Google Forms*

Pergunta: Qual abordagem você considera mais produtiva no seu dia a dia?

- **Ambos:** 62,2%
- **Desenvolvimento Tradicional:** 26,7%
- **Vibe Coding:** 11,1%

Análise: Este resultado é crucial para o estudo. A maioria expressiva (62,2%) aponta que a produtividade máxima não é alcançada pela substituição total, mas pela hibridização. O *Vibe Coding* isolado é visto como menos produtivo (11,1%) do que o método tradicional isolado (26,7%), sugerindo que a IA atua melhor como um "copiloto" que potencializa o desenvolvedor, e não como um substituto autônomo. O cenário ideal, segundo os dados, é a integração das duas abordagens.

5.3.3 Gráfico 3: Impacto no Foco e Motivação

Figura 5 – Gráfico 3: Impacto no Foco e Motivação

Gerado pelo *Google Forms*

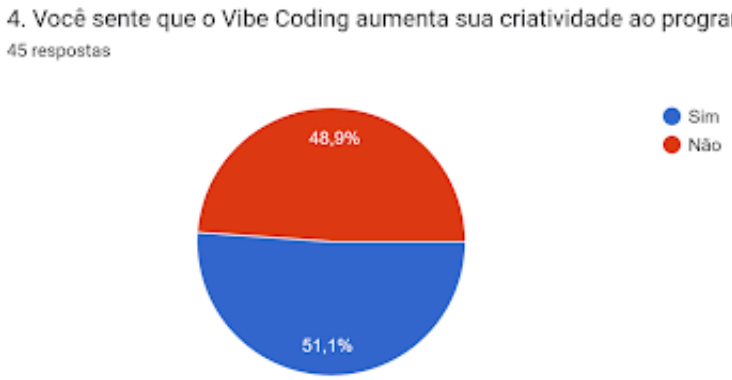
Pergunta: O quanto o Vibe Coding ajuda a manter seu foco e motivação durante o desenvolvimento? (Escala de 1 a 5)

- **Nível 3 (Neutro/Médio):** 35,6%
- **Nível 2 (Baixo):** 24,4%
- **Nível 4 (Alto):** 20%
- **Nível 5 (Muito Alto):** 11,1%
- **Nível 1 (Muito Baixo):** 8,9%

Análise: As opiniões mostram-se divididas e tendendo à neutralidade. Somando os níveis 1 e 2 (33,3%) contra os níveis 4 e 5 (31,1%), percebe-se um equilíbrio técnico. Isso indica que o Vibe Coding não é um fator determinante isolado para a motivação. Para um terço dos usuários, a ferramenta pode até dispersar ou frustrar (níveis baixos), enquanto para outro terço, ela atua como impulsionadora. A maior fatia (Nível 3) sugere que a motivação depende mais do contexto do projeto do que da ferramenta em si.

5.3.4 Gráfico 4: Impacto na Criatividade

Figura 6 – Gráfico 4: Impacto na Criatividade



Gerado pelo *Google Forms*

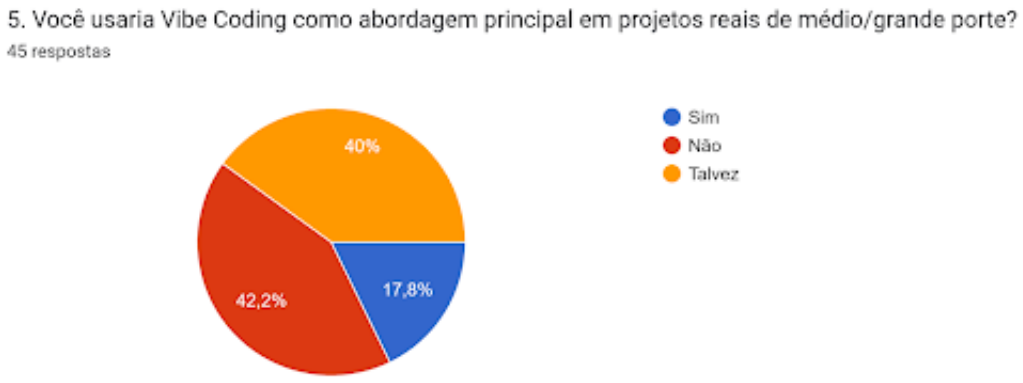
Pergunta: Você sente que o Vibe Coding aumenta sua criatividade ao programar?

- **Não:** 51,1%
- **Sim:** 48,9%

Análise: O resultado apresenta uma polarização quase exata, demonstrando uma dicotomia na percepção da IA. Para uma ligeira maioria (51,1%), a automação de códigos pode estar gerando uma sensação de passividade ou mecanização, inibindo o processo criativo (“apenas aceitar o código sugerido”). Por outro lado, 48,9% sentem que, ao eliminar tarefas repetitivas, a IA libera espaço mental para a criatividade em níveis mais altos de abstração.

5.3.5 Gráfico 5: Aplicabilidade em Projetos de Médio/Grande Porte

Figura 7 – Gráfico 5: Aplicabilidade em Projetos de Médio/Grande Porte



Gerado pelo *Google Forms*

Pergunta: Você usaria Vibe Coding como abordagem principal em projetos reais de médio/grande porte?

- **Não:** 42,2%
- **Talvez:** 40%
- **Sim:** 17,8%

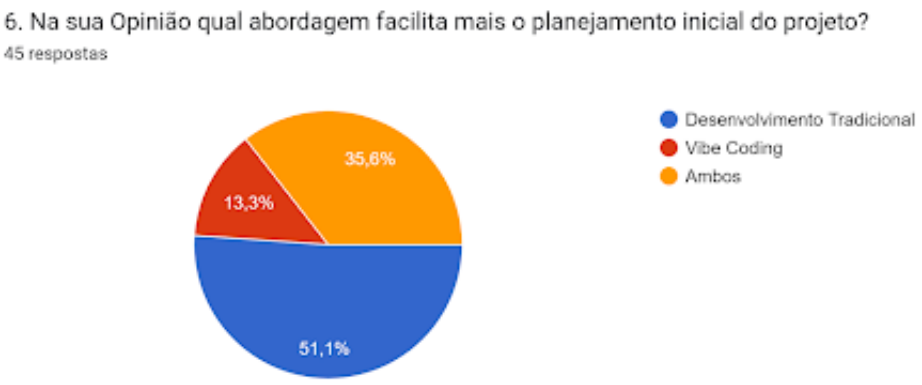
Análise: Aqui observa-se o maior índice de ceticismo. Apenas 17,8% confiariam no Vibe Coding como abordagem principal para sistemas complexos. A soma de "Não" e "Talvez" (82,2%) indica uma forte percepção de risco. Isso sugere que, embora útil para scripts e funções isoladas, a IA ainda carece da confiança necessária para arquitetar e sustentar grandes bases de código, onde a coesão e a estrutura sistêmica são vitais.

5.3.6 Gráfico 6: Planejamento Inicial do Projeto

Pergunta: Na sua opinião, qual abordagem facilita mais o planejamento inicial do projeto?

- **Desenvolvimento Tradicional:** 51,1%
- **Ambos:** 35,6%
- **Vibe Coding:** 13,3%

Figura 8 – Gráfico 6: Planejamento Inicial do Projeto

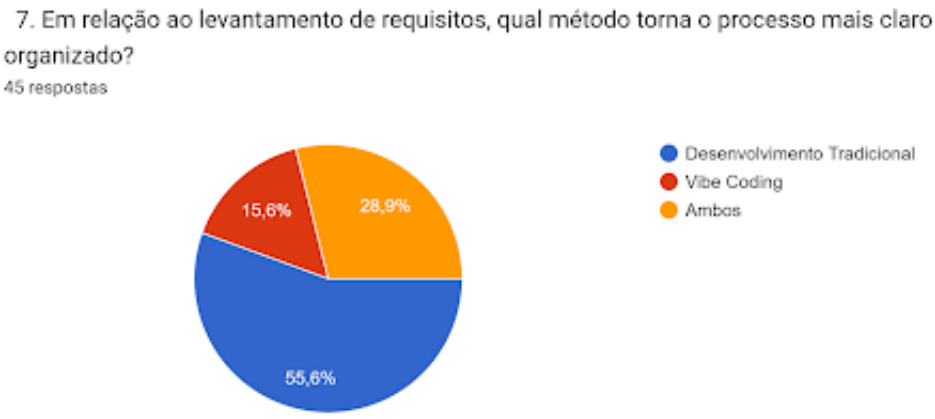


Gerado pelo *Google Forms*

Análise: O Desenvolvimento Tradicional mantém sua hegemonia (51,1%) na fase de planejamento. Isso corrobora a literatura que afirma que a engenharia de software exige pensamento crítico humano para definições arquiteturais. A IA (Vibe Coding) tem pouca penetração isolada nesta fase (13,3%), reforçando que a capacidade de abstração e visão de negócio ainda é uma competência predominantemente humana.

5.3.7 Gráfico 7: Levantamento de Requisitos

Figura 9 – Gráfico 7: Levantamento de Requisitos



Gerado pelo *Google Forms*

Pergunta: Em relação ao levantamento de requisitos, qual método torna o processo mais claro e organizado?

- **Desenvolvimento Tradicional:** 55,6%

- **Ambos:** 28,9%
- **Vibe Coding:** 15,6%

Análise: Seguindo a tendência do planejamento, o levantamento de requisitos é visto como um território do método tradicional (55,6%). A clareza e organização dependem de comunicação humana e entendimento de nuances que a IA atual ainda luta para interpretar com precisão sem supervisão constante. O Vibe Coding é visto como menos eficaz para estruturar as "regras do jogo" antes da codificação começar.

5.3.8 Gráfico 8: Velocidade na Implementação

Figura 10 – Gráfico 8: Velocidade na Implementação

Gerado pelo *Google Forms*

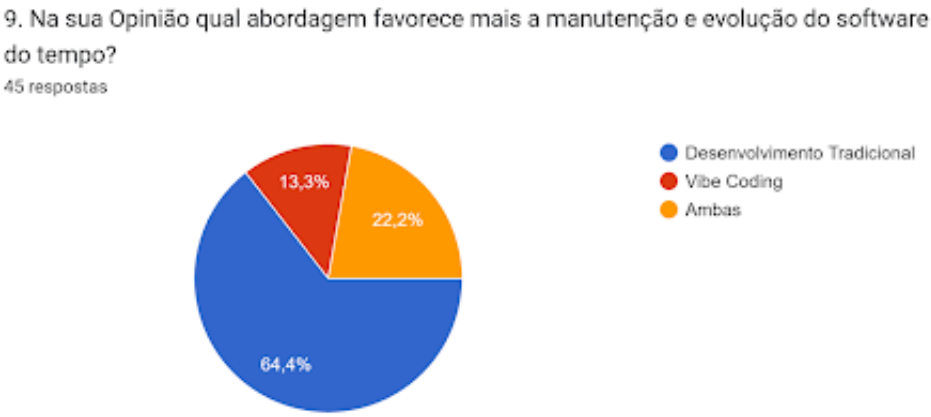
Pergunta: Na sua opinião, o Vibe Coding impacta a fase de implementação positivamente em termos de velocidade?

- **Sim:** 80%
- **Não:** 20%

Análise: Este é o ponto de maior consenso do estudo. Uma maioria esmagadora (80%) concorda que o Vibe Coding entrega velocidade. Contrastando com os gráficos anteriores (planejamento e requisitos), fica evidente a função principal da IA na visão dos desenvolvedores: ela é uma aceleradora de implementação. Quando o "o que fazer" já está decidido (fase tradicional), a IA é imbatível no "fazer rápido".

5.3.9 Gráfico 9: Manutenção e Evolução do Software

Figura 11 – Gráfico 9: Manutenção e Evolução do Software



Gerado pelo *Google Forms*

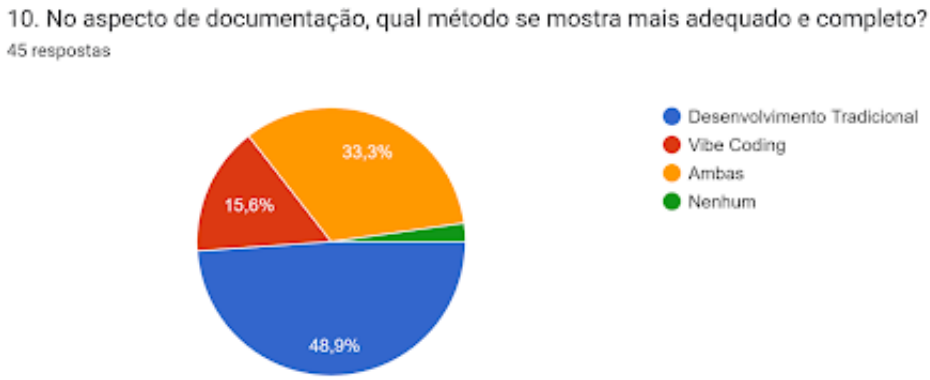
Pergunta: Na sua opinião, qual abordagem favorece mais a manutenção e evolução do software ao longo do tempo?

- **Desenvolvimento Tradicional:** 64,4%
- **Ambos:** 22,2%
- **Vibe Coding:** 13,3%

Análise: A preocupação com a dívida técnica é evidente. Quase 65% dos respondentes acreditam que o método tradicional gera códigos mais fáceis de manter e evoluir. Isso reflete o receio de que o código gerado por IA "código caixa-preta" possa ser difícil de entender ou refatorar futuramente. A velocidade ganha na implementação (Gráfico 8) pode cobrar seu preço na fase de manutenção, caso não haja rigor humano na revisão.

5.3.10 Gráfico 10: Qualidade da Documentação

Figura 12 – Gráfico 10: Qualidade da Documentação



Gerado pelo *Google Forms*

Pergunta: No aspecto de documentação, qual método se mostra mais adequado e completo?

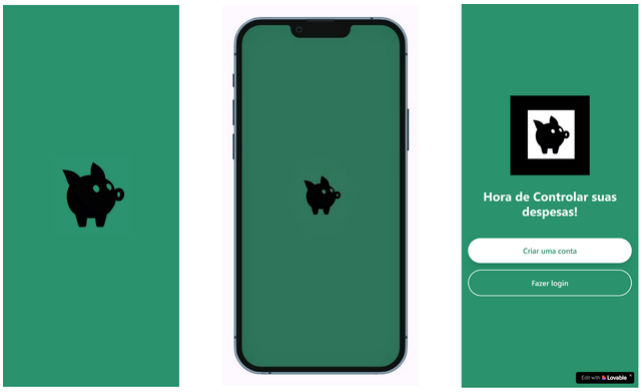
- **Desenvolvimento Tradicional:** 48,9%
- **Ambos:** 33,3%
- **Vibe Coding:** 15,6%

Análise: Apesar da capacidade das IAs de gerar textos, o método tradicional ainda lidera (48,9%) na preferência por documentação adequada. No entanto, a opção “Ambos” tem uma representatividade significativa (33,3%), indicando um potencial crescente das IAs em auxiliar na geração de *boilerplate* de documentação, desde que revisada por humanos para garantir a precisão contextual que muitas vezes a IA perde.

5.4 Comparativo Prático

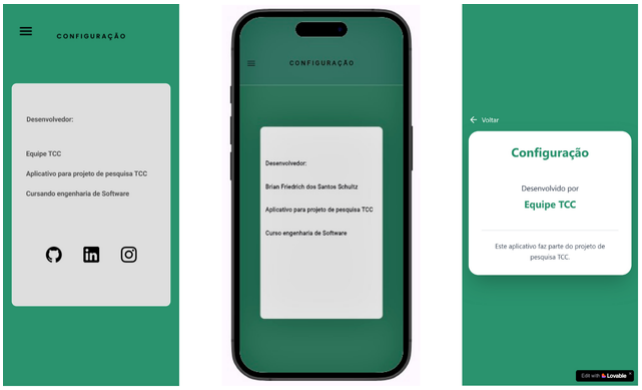
5.4.1 Resultado das telas

Figura 13 – Telas de carregamento



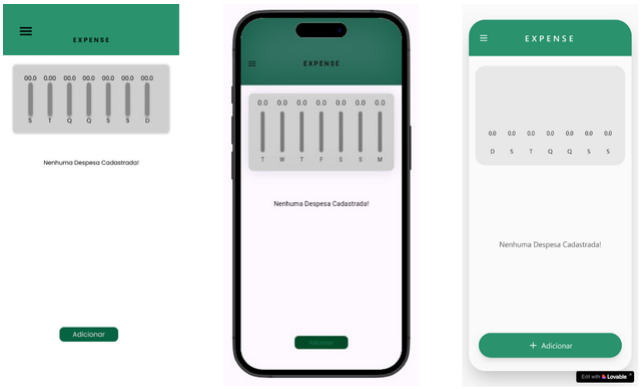
Autoria Própria

Figura 14 – Telas de Configurações



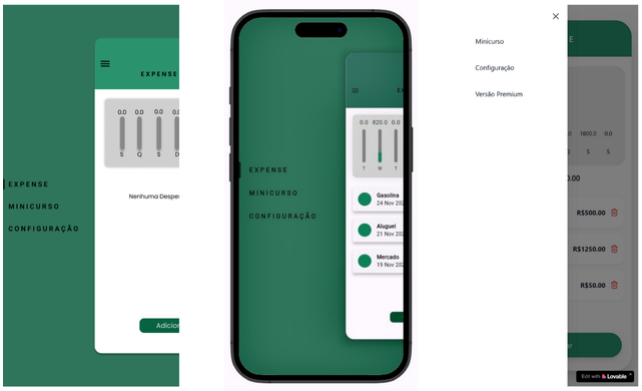
Autoria Própria

Figura 15 – Telas de Início



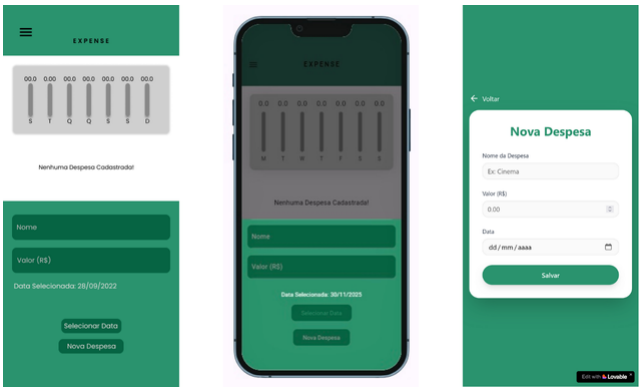
Autoria Própria

Figura 16 – Telas Menu Hambúrguer Ativado



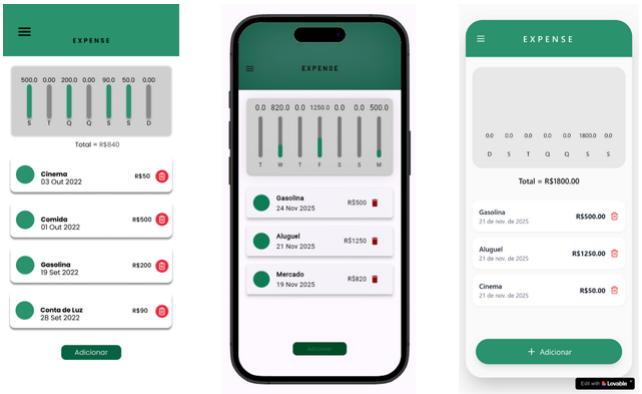
Autoria Própria

Figura 17 – Telas de Nova Despesa



Autoria Própria

Figura 18 – Telas Total de Gastos



Autoria Própria

5.4.1.1 Conclusão Resultados dos Protótipos

Ao longo do desenvolvimento dos dois protótipos, buscou-se constantemente manter fidelidade às telas projetadas no Figma, garantindo coerência visual e uma expe-

riência uniforme entre as plataformas. No entanto, tornou-se evidente que, utilizando métodos tradicionais de codificação, a personalização e o controle fino da interface são maiores. Esse método permite reproduzir com precisão cada detalhe do protótipo, atendendo com rigor às expectativas de usabilidade e identidade visual definidas.

Por outro lado, a abordagem baseada em *vibe coding* mostrou limitações em relação à personalização. Embora a ferramenta permita acelerar o processo de desenvolvimento e gerar interfaces visualmente próximas ao protótipo, ela não alcança o nível de refinamento necessário para reproduzir a proposta original. Esse aspecto representa uma fragilidade do método quando o projeto exige alto grau de detalhamento.

Assim, se o foco do cliente é a personalização, a fidelidade ao design e o maior controle sobre o comportamento da interface, o desenvolvimento tradicional apresenta desempenho superior. Por outro lado, para entregas rápidas, prototipagem funcional e desenvolvimento orientado à produtividade, o método emergente oferece vantagens, ainda que com limitações perceptíveis no aspecto da personalização.

5.5 Análise da Qualidade e Manutenibilidade do Código

Para complementar a análise perceptiva apresentada no Gráfico 9, foi conduzida uma avaliação técnica quantitativa baseada em métricas clássicas de engenharia de software. O objetivo foi comparar diretamente o código gerado via método Tradicional e o produzido por Vibe Coding quanto à complexidade, clareza, e estrutura.

Foram utilizadas métricas reconhecidas na engenharia de software, amplamente aplicadas em ferramentas como SonarQube, SonarCloud, que avaliam aspectos diretamente ligados à manutenibilidade:

Tabela 1 – Métricas de Manutenibilidade

Métrica	**Significado**	**Por que afeta análise**
Linhas do código	Linhas totais	Códigos maiores podem ser mais difíceis de revisar
Confiabilidade	Nível de confiança	Mede a probabilidade do software falhar e a robustez
Manutenibilidade	Facilidade de manutenção	Facilidade com que o código-fonte pode ser compreendido
Hotspots Reviewed	potencial risco de segurança	Avalia se trechos de códigos representam uma ameaça
Duplicações	Redundâncias	Código sem repetição é sinal de limpeza

Autoria Própria

5.5.1 Resultados Obtidos

Tabela 2 – Comparação entre Código Tradicional e Vibe Coding

Critério	Tradicional	Vibe Coding	Impacto
Linhas do código	**1.186 linhas**	**4.900 linhas**	Vibe Coding gera mais código boilerplate
Confiabilidade	**0 - A**	**20 - C**	Código da IA apresenta uma maior zona de bugs
Manutenibilidade	**4 - A**	**36 - A**	Ambos apresentam dívida técnica menos de 5% do esforço total estimado de desenvolvimento
Hotspots Reviewed	**100% - A**	**0,0% - E**	IA resprenta código que pode ser arriscado
Duplicações	**8,8%**	**0,4%**	Método tradicional apresenta maior repetição de código

Autoria Própria

A análise das métricas revela que o código desenvolvido manualmente apresenta desempenho superior em praticamente todos os aspectos avaliados. Embora o *Vibe Coding* tenha produzido um volume significativamente maior de linhas de código, resultando em um projeto mais extenso e com maior presença de *boilerplate*, o desenvolvimento tradicional mostrou-se mais enxuto e objetivo.

Do ponto de vista da confiabilidade, o código manual obteve nota A por não apresentar bugs detectados, enquanto o código gerado pela IA recebeu nota C devido ao registro de 20 ocorrências, demonstrando menor robustez inicial. Em relação à manutenibilidade, ambos obtiveram nota A, indicando baixa dívida técnica proporcional ao tamanho do projeto; contudo, o código da IA acumulou 36 *code smells*, enquanto o manual não apresentou nenhum, evidenciando maior clareza estrutural na abordagem tradicional. No critério *Hotspots Reviewed*, o código manual alcançou 100% de revisão, garantindo avaliação humana sobre pontos sensíveis de segurança, enquanto o *Vibe Coding* permaneceu com 0%, deixando potenciais riscos sem verificação. Por fim, na métrica de duplicações, o código gerado manualmente apresentou apenas 8,6%, ao passo que o da IA atingiu 21,4%, indicando maior repetição de trechos e menor qualidade interna. Em conjunto, esses resultados mostram que, apesar da IA entregar código funcional e com boa manutenibilidade geral segundo o SonarQube, o desenvolvimento tradicional produz um código mais confiável, seguro, limpo e eficiente.

5.6 Conclusão da Seção

A análise conjunta dos gráficos e tabelas permite concluir que o *Vibe Coding* não está substituindo o Desenvolvimento Tradicional, mas sim criando uma nova camada de produtividade focada na fase de implementação.

O estudo aponta para um consenso de que as etapas cognitivas “nobres” (Planejamento, Requisitos, Arquitetura e Manutenção a longo prazo) ainda exigem a robustez do método Tradicional. Em contrapartida, a execução e a codificação bruta são amplamente beneficiadas pela velocidade da IA. O modelo “Híbrido” (apontado como o mais produtivo no Gráfico 2) forte candidato paro o novo padrão de referência na engenharia.

6 Conclusões

O estudo demonstrou que o avanço dos agentes de desenvolvimento baseados em Inteligência Artificial não elimina o valor das práticas tradicionais de engenharia de software, mas cria um novo cenário impactando ambos os paradigmas, tradicionais e emergentes, coexistem e se complementam. A análise comparativa entre as duas abordagens, aliada à construção prática da aplicação e à pesquisa quantitativa, revelou que o *Vibe Coding* se destaca principalmente pela velocidade na implementação, simplificação de tarefas repetitivas e redução da carga operacional do desenvolvedor. Isso ficou evidente nos resultados do *survey*, em que 80% dos participantes afirmaram que o *Vibe Coding* acelera significativamente a fase de implementação e durante a nossa experiência em desenvolver uma aplicação com ambos os métodos.

Entretanto, o Desenvolvimento Tradicional se manteve superior em aspectos críticos como planejamento inicial, levantamento de requisitos, manutenção e documentação, elementos considerados “etapas nobres” do ciclo de engenharia de software. Mais da metade dos participantes da pesquisa, por exemplo, indicaram que o método tradicional proporciona maior clareza no planejamento do projeto (51,1%) e melhor manutenibilidade (64,4%).Esses dados indicam que, enquanto o *Vibe Coding* impulsiona a execução, ele ainda não substitui a profundidade analítica, a precisão arquitetural e a previsibilidade oferecidas pelos métodos tradicionais.

Assim, a principal contribuição original deste estudo é a constatação de que o modelo emergente como uma ferramenta e não como parte principal do desenvolvimento. A combinação equilibrada entre práticas tradicionais e ferramentas de IA, conforme também indicado pelos próprios participantes, com 62,2% apontando o uso conjunto como o cenário mais produtivo. A integração entre autonomia humana e automação inteligente não representa apenas uma tendência tecnológica, mas um novo paradigma que pode redefinir como equipes projetam, constroem e mantêm sistemas no futuro próximo.

6.1 Trabalhos futuros

Nesta seção, apresentam-se possibilidades de continuidade e aprimoramento do estudo, considerando limitações identificadas e oportunidades de aprofundamento sobre o *Vibe Coding*, agentes de IA e seu impacto no desenvolvimento de software.

No contexto de aplicações reais, faz-se necessário investigar sistemas mais complexos, a fim de avaliar melhor os limites dos agentes de IA além do escopo de um MVP simples. Dado que muitas ferramentas de *Vibe Coding* priorizam velocidade em

detrimento de arquitetura e documentação, trabalhos futuros podem explorar modelos de IA treinados especificamente para auxiliar decisões arquiteturais, sugerir padrões adequados e reduzir a geração de código pouco transparente. A melhoria na interpretabilidade dessas ferramentas pode aumentar a confiança dos desenvolvedores e mitigar riscos de dependência tecnológica.

Por fim, expandir a análise para outras plataformas de *Vibe Coding*, comparando produtividade, clareza das instruções e qualidade das entregas. Além disso, investigações no âmbito educacional podem esclarecer como o as tecnologias emergentes influenciam o aprendizado de programação e o desenvolvimento de autonomia técnica nos estudantes, contribuindo para uma compreensão mais ampla do papel da IA na engenharia de software.

Referências

BALDOW, S. R. et al. Low-code e no-code: democratizando o desenvolvimento de software. *Revista Multidisciplinar do Nordeste Mineiro*, p. 1–13, 2024. Disponível em: <https://remunom.ojsbr.com/multidisciplinar/article/view/3032>. 15

BECK, K. *Extreme Programming Explained: Embrace Change*. 2. ed. Boston, MA: Addison-Wesley, 2004. 20

BOEHM, B. W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice Hall, 1981. 20, 21

BOEHM, B. W. A spiral model of software development and enhancement. *Computer*, v. 21, n. 5, p. 61–72, May 1988. 20

CAMPBELL-KELLY, M.; ASPRAY, W. *Computer: A History of the Information Machine*. New York: Basic Books, 1996. 18, 19

CAVALCANTE, S. A case study with github copilot and other artificial intelligence assistants. In: *Proceedings of the 2025 International Conference on Software Engineering Research*. [S.l.]: ScitePress, 2025. Estudo comparativo sobre o uso de assistentes no desenvolvimento. 16

CERUZZI, P. E. *A History of Modern Computing*. Second edition. Cambridge, MA: MIT Press, 2003. ISBN 978-0262532037. 18

CHEN, M. et al. Evaluating large language models trained on code. *CoRR*, 2021. Available at: OpenAI/GitHub documentation. 22

CRESWELL, J. W. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 4. ed. Thousand Oaks, CA: Sage Publications, 2013. 26

FAWZY, A.; TAHIR, A.; BLINCOE, K. Vibe coding in practice: Motivations, challenges, and a future outlook – a grey literature review. In: *Conference'17*. New York, NY, USA: ACM, 2025. p. 1–12. 19, 25

Flutter Dev. *Documentação oficial do Flutter*. [S.l.], 2024. Acesso em: 27 nov. 2024. Disponível em: <https://flutter.dev>. 29

FOWLER, M. *Refactoring: Improving the Design of Existing Code*. 2. ed. Boston, MA: Addison-Wesley Professional, 2018. ISBN 978-0134757599. 21

GENG, F. et al. *Exploring Student-AI Interactions in Vibe Coding*. 2025. Disponível em: <https://arxiv.org/abs/2507.22614>. 24

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. 22

HORVAT, M. What is vibe coding and when should you use it (or not)? *TechRxiv*, Institute of Electrical and Electronics Engineers (IEEE), ago. 2025. Disponível em: <http://dx.doi.org/10.36227/techrxiv.175459780.03758839/v1>. 24

LOVABLE.DEV. *Lovable Documentation*. [S.l.], 2025. 30

MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. [S.l.]: Alta Books, 2009. Tradução de: Código Limpo: Um Manual de Agile Software Craftsmanship. 20, 21

MESKE, C. et al. *Vibe Coding as a Reconfiguration of Intent Mediation in Software Development: Definition, Implications, and Research Agenda*. 2025. Disponível em: <https://arxiv.org/abs/2507.21928>. 19, 23, 25

MICROSOFT. *Vibe Coding and Other Ways artificial intelligence is Changing Who Can Build*. 2024. Online article. Discussão sobre automação e impacto do vibe coding. 15

PENG, B. et al. The impact of artificial intelligence on developer productivity. *arXiv preprint arXiv:2306.????*, 2023. Experimento que analisa produtividade no desenvolvimento assistido por inteligência artificial. Disponível em: <https://arxiv.org>. 15, 16

PRESSMAN, R. S.; MAXIM, B. R. *Engenharia de Software: Uma Abordagem Profissional*. 9. ed. Porto Alegre: AMGH, 2020. 19

RAY, P. P. A review on vibe coding: Fundamentals, state-of-the-art, challenges and future directions. *TechRxiv*, Institute of Electrical and Electronics Engineers (IEEE), maio 2025. Disponível em: <http://dx.doi.org/10.36227/techrxiv.174681482.27435614/v1>. 19, 23, 24, 25

ROYCE, W. W. Managing the development of large software systems. In: *Proceedings of IEEE WESCON*. [S.l.: s.n.], 1970. p. 1–9. 19

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3. ed. [S.l.]: Prentice Hall, 2010. 21

SARKAR, A.; DROSOS, I. *Vibe coding: programming through conversation with artificial intelligence*. 2025. Disponível em: <https://arxiv.org/abs/2506.23253>. 19, 23, 24, 25

SHIHAB, E. et al. The effects of github copilot on computing students' programming effectiveness, efficiency, and processes in brownfield programming tasks. *arXiv preprint arXiv:2506.10051*, 2025. Disponível em: <https://arxiv.org/abs/2506.10051>. 15

SHOHAM, Y. Agent-oriented programming. *Artificial Intelligence*, Elsevier, v. 60, n. 1, p. 51–92, 1993. 22

SHOHAM, Y.; LEYTON-BROWN, K. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. [S.l.]: Cambridge University Press, 2008. 22

SOMMERVILLE, I. *Software Engineering*. 10. ed. [S.l.]: Pearson, 2016. 22

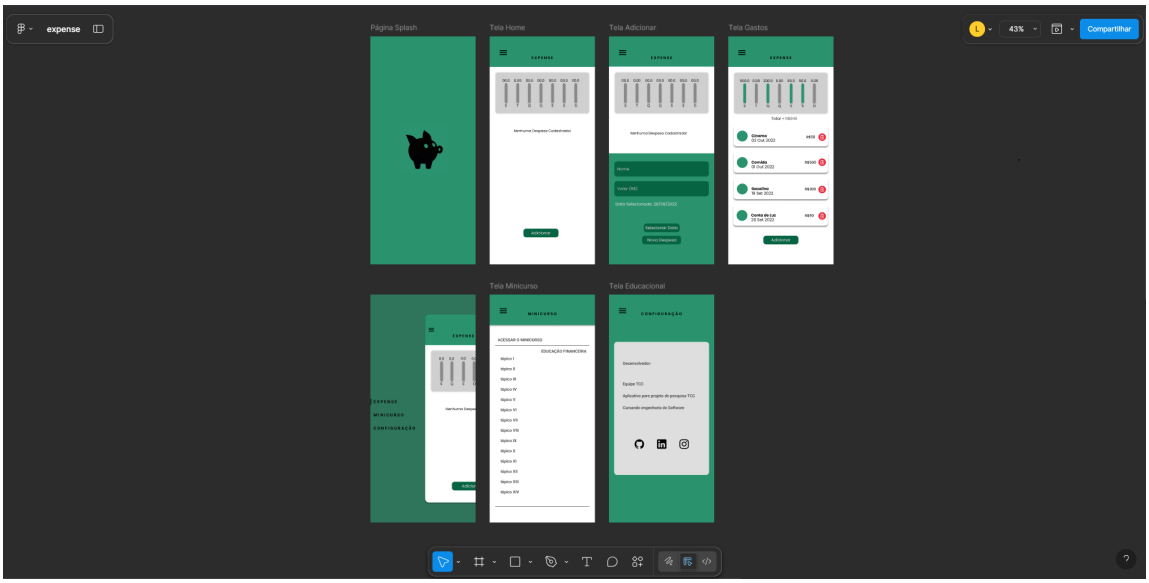
TECHRADAR. Aws kiro: New agentic development tool and its implications for software engineering. *TechRadar*, 2024. Análise da evolução de agentes de desenvolvimento e suas implicações. 16

TURING, A. M. Computing machinery and intelligence. *Mind*, Oxford University Press, v. 59, n. 236, p. 433–460, 1950. 22

TYLDUM, M. *The Imitation Game*. 2014. Filme. United Estated Of America: Black Bear Pictures. 113 min. 18

A Apêndice

Figura 19 – Prototipação Completa



Autoria Própria

A.1 Repositórios do Projeto

O código-fonte completo utilizado neste trabalho está disponível nos repositórios abaixo:

- `Tradicional (Flutter Manual)`
- `Vibe Coding (Lovable)`