

UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**A IA Generativa na Engenharia de
Software**

Um estudo de caso

Cássio Azevedo Cancio

MONOGRAFIA FINAL

MAC 499 — TRABALHO DE
FORMATURA SUPERVISIONADO

Supervisor: Prof. Dr. Paulo Roberto Miranda Meirelles
Cossupervisor: Arthur Pilone Maia da Silva
Cossupervisor: Carlos Eduardo Santos

*O conteúdo deste trabalho é publicado sob a licença CC BY 4.0
(Creative Commons Attribution 4.0 International License)*

*Aos meus pais, que sempre incentivaram meus estudos.
Aos meus professores, que tornaram este trabalho possível.*

Resumo

Cássio Azevedo Cancio. **A IA Generativa na Engenharia de Software: *Um estudo de caso***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo.

[illegible]

Palavras-chave: Palavra-chave1. Palavra-chave2. Palavra-chave3.

Abstract

Cássio Azevedo Cancio. **Generative AI in Software Engineering: A case study.** Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo.

[illegible]

Keywords: Keyword1. Keyword2. Keyword3.

Lista de abreviaturas

IA	Inteligência Artificial (<i>Artificial Intelligence</i>)
IME	Instituto de Matemática e Estatística
LLM	Modelo de Linguagem de Grande Escala (<i>Large Language Model</i>)
USP	Universidade de São Paulo

Lista de figuras

Lista de tabelas

Lista de programas

Sumário

Introdução	1
Contexto	1
Motivação	1
Objetivos	2
1 Referencial Teórico	3
1.1 Engenharia de Software	3
1.1.1 Etapas do Desenvolvimento de Software	3
1.1.2 Metodologias de Desenvolvimento	6
1.1.3 Ferramentas de Desenvolvimento	6
1.2 Inteligência Artificial	6
1.2.1 Conceitos Básicos	6
1.2.2 Redes Neurais	6
1.2.3 Aprendizado de Máquina	6
1.3 IA Generativa	6
1.3.1 Modelos de Linguagem	6
1.3.2 Transformers	6
1.3.3 LLMs (Large Language Models)	6
2 Metodologia	7
2.1 Abordagem de Pesquisa	7
2.1.1 Tipo de Pesquisa	7
2.1.2 Procedimentos Metodológicos	7
2.2 Coleta de Dados	7
2.2.1 Fontes de Dados	7
2.2.2 Instrumentos de Coleta	7
2.2.3 Processo de Coleta	7
2.3 Análise de Dados	7

2.3.1	Métodos de Análise	7
2.3.2	Ferramentas Utilizadas	7
2.3.3	CrITÉrios de AvaliaÇão	7
3	Resultados	9
3.1	Análise dos Dados	9
3.2	Avaliação do Sistema	9
3.2.1	Desempenho	9
3.2.2	Eficiência	9
3.2.3	Usabilidade	9
3.3	Discussão	9
3.3.1	Limitações Identificadas	9
3.3.2	Melhorias Propostas	9
4	Conclusão	11
4.1	Resumo dos Resultados	11
4.1.1	Principais Descobertas	11
4.1.2	Objetivos Alcançados	11
4.1.3	Contribuições	11
4.2	Trabalhos Futuros	11
4.2.1	Direções de Pesquisa	11
4.2.2	Melhorias Propostas	11
4.2.3	Desafios Identificados	11
4.3	Considerações Finais	11
Apêndices		
Anexos		
Referências		13
Índice remissivo		15

Introdução

Contexto

A engenharia de *software* é um campo da computação que surgiu e se desenvolveu através da crescente demanda da sociedade do fim do século XX até a atualidade por sistemas computacionais cada vez mais complexos. Neste contexto, ferramentas, métodos e processos foram criados para possibilitar o atendimento dessa demanda.

Nas últimas décadas, os estudos em inteligência artificial (IA) avançaram rapidamente, de modo que um novo paradigma em IA surgiu, a IA generativa. Diferentemente da IA tradicional, a IA generativa pode criar conteúdos novos e originais baseados no que aprendeu, em vez de apenas copiar, imitar e reproduzir algo que já existe.

Dada a flexibilidade e a abertura de diversas possibilidades com essa nova tecnologia, é natural que uma de suas aplicações fosse a engenharia de *software*. Nos últimos anos, diversos estudos foram publicados de modo a analisar essas aplicações, suas consequências e propor diferentes abordagens para tais aplicações. (JOHNSON e MENZIES, 2024 e TERRAGNI *et al.*, 2025)

Desta forma, este projeto se propõe a levantar os resultados observados em diversos artigos que tratam do estudo do impacto da IA generativa na engenharia de *software*. Além de realizar um estudo de caso sobre a aplicação da IA generativa ao longo das fases do desenvolvimento de *software*, avaliando sua utilidade, limitações e impacto na qualidade e produtividade.

Como estudo de caso, foi realizado o desenvolvimento de uma aplicação web na área de investimentos, com banco de dados, *backend* em *Java Spring Boot* e *frontend* em *Angular*, utilizando ferramentas de IA generativa nas diferentes fases do desenvolvimento de um sistema. As fases analisadas foram: coleta e análise de requisitos, estudo de viabilidade, *design* de *software*, codificação e testes.

Motivação

Este trabalho se faz relevante no contexto em que o uso de ferramentas de IA generativa vem crescendo com o passar dos anos, desde o surgimento de ferramentas como *ChatGPT* e *GitHub Copilot*. Segundo dados da *Stack Overflow 2024 Developer Survey* (STACK OVERFLOW, 2024), 63,2% dos desenvolvedores profissionais já utilizam ferramentas de IA no seu processo de desenvolvimento, enquanto que 13,5% desse mesmo grupo planeja utilizá-las em

breve. Além disso, entre os desenvolvedores que responderam usar inteligência artificial, 82% a utiliza para escrever código.

Desta maneira, é evidente que uma nova tecnologia com amplo uso no mercado de *software* e que abre possibilidade para diversas aplicações, terá impactos sobre como os desenvolvedores escrevem seus códigos. Assim, é de suma importância buscar avaliar e compreender melhor de que maneira esses impactos vêm ocorrendo nas bases de código, inclusive através de um estudo de caso.

Objetivos

Os principais objetivos do trabalho são:

- Compreender de que maneira a IA generativa tem impactado na produção de código das empresas de *software*, através de um levantamento de dados disponíveis em outros artigos;
- Desenvolver um sistema com todo seu processo voltado ao uso de ferramentas de IA generativa ao longo das suas diferentes fases;
- Documentar os resultados gerados pela IA durante o processo de desenvolvimento do sistema, incluindo os *prompts* utilizados;
- Analisar os resultados obtidos, a fim de mensurar a qualidade das respostas geradas.

Capítulo 1

Referencial Teórico

1.1 Engenharia de Software

Segundo a definição do *Institute of Electrical and Electronics Engineers* (IEEE) (1990), a engenharia de *software* é a aplicação de uma sistemática, disciplinada e quantificável abordagem para o desenvolvimento, operação e manutenção de um *software*. Para que a engenharia de *software* seja viável, dada a complexidade dos sistemas demandados na atualidade, foram desenvolvidas etapas, metodologias e ferramentas que dessem suporte aos atores envolvidos no projeto, como desenvolvedores, analistas, investidores, clientes, entre outros.

1.1.1 Etapas do Desenvolvimento de Software

O Ciclo de Vida de Desenvolvimento de Software (SDLC) consiste numa sequência de processos pelos quais o desenvolvimento de um *software* ocorre, de modo a produzir um resultado eficaz e de alta qualidade. Existe alguma variação no número de passos descritos por diferentes fontes, mas, em geral, há sete fases essenciais: planejamento, análise de requisitos, *design*, codificação, testes, implantação e manutenção.

Planejamento

A fase inicial envolve definir o propósito e o escopo do *software*. Durante esta etapa, a equipe de desenvolvimento deve levantar as tarefas necessárias, elaborar estratégias para cumpri-las e colaborar de modo a compreender as necessidades dos usuários finais. Neste processo, os objetivos do *software* e qual problema ele se propõe a resolver precisam ficar claros a todos os envolvidos.

Além disso, nesta fase também ocorre o estudo de viabilidade, ou seja, desenvolvedores e outros atores do projeto avaliam desafios técnicos e financeiros que possam impactar a evolução ou o sucesso do *software*. Ao fim desta fase, um plano de projeto é criado, com o intuito de detalhar as funções do sistema, os recursos necessários, possíveis riscos e o cronograma do projeto. Ao definir papéis, responsabilidades e expectativas claras, o planejamento estabelece uma base sólida para um processo eficiente de desenvolvimento.

Análise de Requisitos

A segunda fase do SDLC visa identificar e registrar os requisitos dos usuários finais. Nesta etapa, a equipe de projeto realiza o levantamento dos requisitos, por meio da coleta de informações das partes interessadas, como analistas, usuários e clientes. São empregadas técnicas como entrevistas, pesquisas e grupos de foco para compreender as necessidades e expectativas dos usuários.

Após a coleta, os dados são analisados, diferenciando os requisitos essenciais dos desejáveis. Essa análise possibilita a definição das funcionalidades, desempenho, segurança e interfaces do *software*. Neste momento, são definidos os requisitos funcionais e não funcionais. Os requisitos funcionais especificam as funções que o *software* deve realizar, ou seja, o que o sistema deve fazer. Já os requisitos não funcionais tratam de como o sistema deve se comportar, incluindo aspectos como desempenho, segurança, usabilidade e escalabilidade.

O resultado desse processo é o Documento de Especificação de Requisitos (DER), que descreve o propósito, as funcionalidades e características do *software*, servindo como guia para a equipe de desenvolvimento e fornecendo estimativas de custo, quando necessário. O êxito desta fase é crucial para o sucesso do projeto, pois assegura que a solução desenvolvida atenda às expectativas dos usuários.

Design

A fase de *design* é responsável pela definição da estrutura do *software*, abrangendo sua funcionalidade e aparência. A equipe de desenvolvimento detalha a arquitetura do sistema, a navegação, as interfaces de usuário e a modelagem do banco de dados, assegurando que o *software* tenha boa usabilidade e seja eficiente.

Entre as atividades desta fase, destaca-se a elaboração de diagramas de fluxo de dados, de entidade-relacionamento, de classes, protótipos de interface e diagramas arquiteturais. O objetivo é garantir que as estruturas projetadas sejam suficientes para dar suporte a todas as funcionalidades do sistema. Também são identificadas dependências, pontos de integração e eventuais restrições, como limitações do equipamento físico e requisitos de desempenho.

O resultado desta fase é o Documento de *Design* de Software (DDS) que estrutura formalmente as informações do projeto e trata preocupações de *design*. Neste documento, são adicionados os artefatos produzidos, servindo como guia estável para coordenar equipes grandes e garantir que todos os componentes do sistema funcionem de maneira integrada.

Codificação

Na fase de codificação, os engenheiros e desenvolvedores transformam o *design* do *software* em código executável. O objetivo é produzir um *software* funcional, eficiente e com boa usabilidade. Para isso, utilizam-se linguagens de programação adequadas, seguindo o DDS e diretrizes de codificação estabelecidas pela organização e pela legislação local.

Durante esta fase, são realizadas revisões de código, nas quais os membros da equipe examinam o trabalho uns dos outros para identificar erros ou inconsistências, garantindo

elevados padrões de qualidade. Além disso, testes preliminares internos são conduzidos para garantir que as funcionalidades básicas do sistema foram atendidas.

Ao final da fase de codificação, o *software* passa a existir como um produto funcional, representando a materialização dos esforços das etapas anteriores, mesmo que ainda sejam necessários refinamentos e ajustes subsequentes. O resultado desta fase é o código-fonte.

Testes

A fase de testes consiste em verificar a qualidade e a confiabilidade do *software* antes de sua entrega aos usuários finais. Seu objetivo é identificar falhas, erros e vulnerabilidades, assegurando que o sistema atenda aos requisitos especificados.

Inicialmente, são definidos parâmetros de teste alinhados aos requisitos do *software* e casos de teste que contemplem diferentes cenários de uso. Em seguida, são conduzidos testes de diversos níveis e tipos, incluindo testes de unidade, de integração, de sistema, de segurança e de aceitação, permitindo a avaliação tanto de componentes individuais quanto da operação do sistema na sua totalidade.

Quando um erro é identificado, ele é registrado detalhadamente, incluindo seu comportamento, métodos de reprodução e impacto sobre o sistema. As falhas são encaminhadas para correção e o *software* retorna à fase de testes para validação. Este ciclo de teste e correção se repete até que o sistema esteja conforme os critérios previamente estabelecidos. O resultado desta fase é um código-fonte mais robusto e menos propenso a falhas.

Implantação

A fase de implantação ou *deployment* consiste em disponibilizar o *software* aos usuários finais, garantindo sua operacionalidade no ambiente de produção. Este processo ocorre tanto no primeiro lançamento do sistema, quanto quando ele já está em uso pelos usuários e passando por atualizações. Por isso, o processo deve minimizar interrupções e impactos negativos nos acessos dos usuários.

A escolha da estratégia de implantação é feita conforme as características do sistema e de seus usuários. As estratégias mais comuns são:

- *Rolling*: a atualização ocorre de forma gradual, substituindo instâncias antigas por novas até que todo o sistema esteja atualizado;
- *Blue-Green*: dois ambientes paralelos são mantidos, um em produção e outro em preparação, permitindo a troca imediata entre eles;
- *Canary*: a nova versão é liberada primeiramente para um grupo de usuários, monitorando o comportamento do sistema antes de expandir a implantação para todos.

Além de colocar o *software* em operação, esta fase envolve assegurar que os usuários compreendam seu funcionamento. Para isso, podem ser fornecidos manuais, treinamentos e suporte técnico. Desta maneira, a fase de implantação marca a transição do *software* de projeto para produto, iniciando efetivamente o cumprimento de seus objetivos e a entrega de valor ao usuário.

Manutenção

A fase de manutenção é caracterizada por suporte contínuo e por melhorias incrementais, de modo a garantir que o *software* mantenha seu funcionamento adequado, acompanhe as necessidades dos usuários e as demandas de mercado. Nesta fase, são realizadas atualizações, correções de falhas e suporte ao usuário.

Considerando o horizonte de longo prazo, a manutenção inclui estratégias de modernização ou substituição do *software*, buscando manter sua relevância e adequação às evoluções tecnológicas.

1.1.2 Metodologias de Desenvolvimento

Continuous Deployment (CD): Um processo automatizado onde cada alteração aprovada no código é automaticamente disponibilizada para os utilizadores.

1.1.3 Ferramentas de Desenvolvimento

1.2 Inteligência Artificial

1.2.1 Conceitos Básicos

1.2.2 Redes Neurais

1.2.3 Aprendizado de Máquina

1.3 IA Generativa

1.3.1 Modelos de Linguagem

1.3.2 Transformers

1.3.3 LLMs (Large Language Models)

Capítulo 2

Metodologia

2.1 Abordagem de Pesquisa

2.1.1 Tipo de Pesquisa

2.1.2 Procedimentos Metodológicos

2.2 Coleta de Dados

2.2.1 Fontes de Dados

2.2.2 Instrumentos de Coleta

2.2.3 Processo de Coleta

2.3 Análise de Dados

2.3.1 Métodos de Análise

2.3.2 Ferramentas Utilizadas

2.3.3 Critérios de Avaliação

Capítulo 3

Resultados

3.1 Análise dos Dados

3.2 Avaliação do Sistema

3.2.1 Desempenho

3.2.2 Eficiência

3.2.3 Usabilidade

<https://forum.cursor.com/t/cursor-tab-hides-all-code-in-the-line/49675/10>

3.3 Discussão

3.3.1 Limitações Identificadas

3.3.2 Melhorias Propostas

Capítulo 4

Conclusão

4.1 Resumo dos Resultados

4.1.1 Principais Descobertas

4.1.2 Objetivos Alcançados

4.1.3 Contribuições

4.2 Trabalhos Futuros

4.2.1 Direções de Pesquisa

4.2.2 Melhorias Propostas

4.2.3 Desafios Identificados

4.3 Considerações Finais

Referências

- [“IEEE Standard Glossary of Software Engineering Terminology” 1990] “Ieee standard glossary of software engineering terminology”. *IEEE Std 610.12-1990* (1990), pp. 1–84. DOI: [10.1109/IEEESTD.1990.101064](https://doi.org/10.1109/IEEESTD.1990.101064) (citado na pg. 3).
- [JOHNSON e MENZIES 2024] Brittany JOHNSON e Tim MENZIES. “ AI Over-Hype: A Dangerous Threat (and How to Fix It) ”. *IEEE Software* 41.06 (nov. de 2024), pp. 131–138. ISSN: 1937-4194. DOI: [10.1109/MS.2024.3439138](https://doi.org/10.1109/MS.2024.3439138). URL: <https://doi.ieeecomputersociety.org/10.1109/MS.2024.3439138> (citado na pg. 1).
- [STACK OVERFLOW 2024] STACK OVERFLOW. *Stack Overflow Developer Survey 2024*. 2024. URL: <https://survey.stackoverflow.co/2024/> (acesso em 19/08/2025) (citado na pg. 1).
- [TERRAGNI *et al.* 2025] Valerio TERRAGNI, Annie VELLA, Partha ROOP e Kelly BLINCOE. “The future of ai-driven software engineering”. *ACM Trans. Softw. Eng. Methodol.* 34.5 (mai. de 2025). ISSN: 1049-331X. DOI: [10.1145/3715003](https://doi.org/10.1145/3715003). URL: <https://doi.org/10.1145/3715003> (citado na pg. 1).

Índice remissivo

C

Captions, *veja* Legendas

Código-fonte, *veja* Floats

E

Equações, *veja* Modo matemático

F

Figuras, *veja* Floats

Floats

Algoritmo, *veja* Floats, ordem

Fórmulas, *veja* Modo matemático

I

Inglês, *veja* Língua estrangeira

P

Palavras estrangeiras, *veja* Língua es-

trangeira

R

Rodapé, notas, *veja* Notas de rodapé

S

Subcaptions, *veja* Subfiguras

Sublegendas, *veja* Subfiguras

T

Tabelas, *veja* Floats

V

Versão corrigida, *veja* Tese/Dissertação,
versões

Versão original, *veja* Tese/Dissertação,
versões