REVISTA TÓPICOS
ISSN 2965-6672 | Qualis A2

INTELIGÊNCIA ARTIFICIAL GENERATIVA NA ENGENHARIA DE SOFTWARE: UMA ANÁLISE DOS IMPACTOS NA PRODUTIVIDADE, QUALIDADE DO CÓDIGO E SEGURANÇA NO DESENVOLVIMENTO DE SISTEMAS

GENERATIVE ARTIFICIAL INTELLIGENCE IN SOFTWARE ENGINEERING: AN ANALYSIS OF ITS IMPACTS ON PRODUCTIVITY, CODE QUALITY, AND SECURITY IN SYSTEMS DEVELOPMENT

Ciências Exatas e da Terra, Engenharias • 18/11/2026

REGISTRO DOI: 10.70773/revistatopicos/794751162

Gabriel Victor Martins Carvalho[1]

Yago da Costa Jardim Alves Braga[2]

Daniel Monteiro de Carvalho[3]

Caio Cezar Jotta Nogueira[4]

Christiane Martins de Vasconcellos Silveira[5]

Wellington Ávila[6]

RESUMO

A ascensão na Inteligência Artificial Generativa (IAGen) e popularização de Grandes Modelos de Linguagem (LLMs) revolucionaram a área de Engenharia de Software, trazendo assistentes de programação virtuais para todos os estágios do ciclo de vida de desenvolvimento de sistemas. Nesse contexto, este trabalho explora os efeitos que a IAGen tem nas medidas de produtividade dos programadores; na qualidade estrutural do código-fonte; na segurança e confiabilidade do software. A revisão integrativa da literatura e síntese empírica dos estudos realizados de 2022-2026 no país mostraram que a IAGen reduz substancialmente o tempo da atividade de codificação primária (-49%), além de validar até 86% das decisões tomadas durante a atividade de priorização de requisito. Por outro lado, nota-se que a IAGen gera um paradoxo. Ela preserva a coesão local e a hierarquia de herança, mas tende a incrementar o acoplamento estrutural (CBO/RFC), complexidade ciclomática (WMC), podendo ainda gerar *code smells* em 49,5% das tarefas. Além disso, observa-se que a taxa média de tempo gasto na refatoração é de 38,1 minutos por tarefa. Em razão disso, para que ferramentas como GitHub Copilot, ChatGPT e Gemini sejam utilizadas de forma sustentável, elas exigem alto nível de governança técnica, sendo necessário que tenham suas atividades auditadas por humanos de forma híbrida. Também, elas devem estar integradas à análise de segurança estática.

Palavras-chave: IA generativa; engenharia de software; produtividade; qualidade de código; segurança.

ABSTRACT

The rise of Generative Artificial Intelligence (GenAI) and the widespread adoption of Large Language Models (LLMs) have revolutionized the field of software engineering, bringing virtual

programming assistants to every stage of the systems development life cycle. In this context, this study explores the effects that GAI has on programmers' productivity metrics; on the structural quality of source code; and on software security and reliability. An integrative literature review and empirical synthesis of studies conducted in Brazil from 2022 to 2026 showed that IAGen substantially reduces the time spent on primary coding (-49%) and validates up to 86% of the decisions made during the requirement prioritization process. On the other hand, it is noted that IAGen creates a paradox. It preserves local cohesion and the inheritance hierarchy but tends to increase structural coupling (CBO/RFC) and cyclomatic complexity (WMC), and may also generate code smells in 49.5% of tasks. Furthermore, the average time spent on refactoring is 38.1 minutes per task. Consequently, for tools such as GitHub Copilot, ChatGPT, and Gemini to be used sustainably, they require a high level of technical governance, necessitating that their activities be audited by humans in a hybrid manner. Additionally, they must be integrated with static security analysis.
Keywords: generative AI; software engineering; productivity; code quality; security.

1. INTRODUÇÃO

A transformação digital acelerada da última década culminou no surgimento e na rápida disseminação de ferramentas baseadas em Inteligência Artificial Generativa (IAGen) e Grandes Modelos de Linguagem (LLMs, do inglês *Large Language Models*) (Vaswani *et al.*, 2017). Plataformas e assistentes virtuais de codificação como GitHub Copilot, OpenAI ChatGPT, Google Gemini, Anthropic Claude e Cursor redefiniram as práticas contemporâneas de desenvolvimento de software (GitHub; OpenAI, 2022; Google DeepMind, 2023).

Capacitados pelo processamento de bilhões de linhas de código *open source*, esses modelos são capazes de interpretar *prompts* em linguagem natural, traduzir requisitos funcionais complexos em implementações executáveis, sugerir refatorações e autocompletar estruturas sintáticas inteiras em tempo real.

O entusiasmo inicial do mercado com o aumento acelerado na velocidade de entrega vem seguido de um debate acadêmico e técnico essencial. A principal dúvida desta pesquisa é entender se o uso de IA generativa em sistemas de software realmente (e por muito tempo) vai aumentar a eficiência do ciclo de vida de um software. Um processo mais aberto de desenvolvimento também pode trazer alguma imperfeição que acabe resultando na criação de fragilidades de segurança nas aplicações? E, adicionalmente, como se dará o gerenciamento das dívidas técnicas?

A justificativa deste estudo fundamenta-se na urgente necessidade de respaldar a tomada de decisão de gestores de tecnologia, arquitetos de sistemas e educadores com evidências empíricas quantitativas e qualitativas. Embora muitos relatórios de mercado indiquem o investimento bilionário em infraestrutura de IA, a comunidade científica ainda carece de *frameworks* (ou modelos) capazes de mensurar o impacto na velocidade de escrita do código, a degradação arquitetural a longo prazo dessa arquitetura e os vetores de vulnerabilidade introduzidos pelos copilotos inteligentes.

O objetivo geral deste artigo é analisar os impactos da Inteligência Artificial Generativa na Engenharia de Software, investigando seus efeitos sobre a produtividade das equipes, a qualidade estrutural do código-fonte gerado e a segurança da informação no desenvolvimento de sistemas.

A fim de atingir o objetivo geral deste estudo, foram estabelecidos os seguintes objetivos específicos: i) Levantar e medir as diferenças de produto (desempenho produtivo) e tempo (de desenvolvimento) proporcionados pela utilização de assistentes generativos; ii) Analisar a qualidade do código gerado, utilizando métricas objetivas de orientação a objetos como: coesão, acoplamento, complexidade e *code smells*; iii) Identificar possíveis riscos de segurança, vulnerabilidades (OWASP) e alucinações técnicas inerentes à automação de código. Além disso, propõe boas práticas de governança.

Este artigo está estruturado em sete seções principais. A seguir, na Seção 2 são apresentados os fundamentos teóricos dos principais conceitos abordados neste trabalho. A Seção 3 sintetiza os trabalhos relacionados em uma análise comparativa. Na Seção 4, descreve-se a metodologia utilizada. A Seção 5 discute os resultados e discussão organizados pelos três eixos específicos. A Seção 6 apresenta a análise das ameaças à validade do trabalho e a Seção 7 conclui o trabalho com as considerações finais e referências.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. Engenharia de Software e o Ciclo de Vida do Desenvolvimento

A Engenharia de Software (ES) procura estabelecer uma nova disciplina de engenharia que busca aplicar métodos sistemáticos, mensuráveis e disciplinados para o processo de desenvolvimento, operação e manutenção de software (Pressman; Maxim, 2021; Sommerville, 2019). Entre esses processos, destaca-se o ciclo de vida do software, que compreende atividades como elicitação de

requisitos, análise, projeto arquitetural, implementação, testes e manutenção. A inserção da IA generativa ocorre de maneira transversal a essas fases: na fase inicial, apoia a especificação e priorização de requisitos; na fase de projeto, auxilia no suporte à tomada de decisão de padrões de projeto (Colombo *et al.*, 2025) e na verificação de diagramas UML (Doi *et al.*, 2026); e na implementação, atua como um *pair programmer* virtual (Ferreira, 2025).

2.2. Inteligência Artificial Generativa: LLMs, Arquitetura *Transformer* e Geração de Código

O modelo tecnológico subjacente aos assistentes modernos é baseado na arquitetura *Transformer* (Vaswani *et al.*, 2017) que utiliza o mecanismo de autoatenção (*self-attention*). Este modelo é caracterizado pelas capacidades de capturar dependências de longo alcance em sequências de texto e que superaram as limitações das redes neurais recorrentes tradicionais. Os LLMs são treinados utilizando a técnica de aprendizado autossupervisionado em grandes volumes de dados de código público e aprendem distribuições de probabilidade estatística de sequências de *tokens*. Assim, a geração de código configura-se como um processo probabilístico de previsão do próximo *token*, condicionado pelo contexto fornecido pelo desenvolvedor por meio de *prompts* (Nguyen; Nadi, 2022; White *et al.*, 2024).

2.3. Ferramentas de Assistência à Programação

As ferramentas existentes vão desde extensões que se integram diretamente nos ambientes de desenvolvedores (IDEs), como o GitHub Copilot, Amazon Q Developer e Cursor até em interfaces de chat conversacionais genéricas como o ChatGPT e Claude. Ao

contrário das extensões de IDEs, que analisam continuamente o contexto dos arquivos abertos e a estrutura do projeto para dar sugestões de código à medida que o código é desenvolvido, as ferramentas conversacionais dependem de *prompts* elaborados dos usuários. Estudos demonstram que a eficácia dessas ferramentas está diretamente ligada à capacidade da IA em interpretar o contexto do sistema e à habilidade de engenharia de *prompt* do desenvolvedor (Deshmukh *et al.*, 2025; White *et al.*, 2024).

2.4. Produtividade no Desenvolvimento de Software

No contexto inicial de desenvolvimento de software, a produtividade era medida de maneiras simples, como contagem de linhas de código (LOC). Na atualidade, no campo da engenharia de software, existem várias abordagens multidimensionais, como o *framework* SPACE (*Satisfaction, Performance, Activity, Communication, Efficiency*) e métricas como DORA (*Deployment Frequency, Lead Time for Changes, Change Failure Rate, Time to Restore Service*) (Forsgren; Humble; Kim, 2018) que medem a produtividade do software. Em relação à IAGen, é possível considerar dois pontos principais: o tempo que demora para ser completada uma tarefa de codificação, e diminuição do esforço cognitivo que deve ser realizado em tarefas repetitivas ou de protótipos (Yamada; Mendes, 2026; Assis; Silva, 2025).

2.5. Qualidade de Código e Dívida Técnica

A qualidade de código compreende atributos internos de legibilidade, modularidade, coesão, acoplamento e facilidade de manutenção (IEEE, 1990; Sommerville, 2019). As métricas do conjunto CK (Chidamber e Kemerer) propõem medidas para avaliar

a estrutura interna das classes na programação orientada a objetos. As métricas são: LCOM (Falta de Coesão nos Métodos), CBO (Acoplamento entre Objetos de Classe), WMC (Métodos Ponderados por Classe) e RFC (Resposta para uma Classe). A presença de sintomas de má estruturação designa-se como code smells (Martin, 2019). Quando código de qualidade questionável é incorporado ao repositório, acumula-se a chamada dívida técnica, que demanda esforço futuro de refatoração para evitar a degradação da sustentabilidade do sistema (Ferreira, 2025; Siddiq *et al.*, 2022).

2.6. Segurança no Desenvolvimento e Riscos de Código Automatizado

A construção de software seguro se dá através da adoção de boas práticas de codificação (*secure coding standards*) e mitigando as vulnerabilidades descritas na OWASP (*Open Web Application Security Project*). Como os LLMs são treinados em repositórios públicos que frequentemente contêm falhas sintáticas, práticas obsoletas ou vulnerabilidades conhecidas, o código gerado automaticamente corre o risco de reproduzir brechas de segurança, tais como injeção de SQL, tratamento inadequado de dados sensíveis e falhas de controle de acesso (Siddiq *et al.*, 2022; Clark *et al.*, 2024).

3. TRABALHOS RELACIONADOS

Para contextualizar o desenvolvimento da área de conhecimento entre IAGen e Engenharia de Software, foi feita uma síntese dos principais trabalhos empíricos publicados nos últimos 5 anos. O quadro a seguir (Quadro 1) compara esses trabalhos indicando os

principais autores, ano, metodologia utilizada, ferramenta avaliada e principais descobrimentos.

Quadro 1. Síntese dos estudos empíricos sobre IA Generativa na Engenharia de Software (2022–2026)

Autor / Ano	Método	Ferramenta	Principais Achados
Ferreira (2025)	Experimento quantitativo em 23 repositórios Java (115 *issues*)	GitHub Copilot (GPT-5 mini)	Preserva coesão (LCOM*) e herança (DIT), mas aumenta acoplamento (CBO/RFC) e WMC. 49,5% das tarefas geraram *code smells*, exigindo 38,1 min de refatoração.
Yamada e Mendes (2026)	Estudo de caso qualiquantitativo (questionário + experimento prático)	IAGen genérica (ChatGPT / Copilot)	Redução de 1h15m para 38m no tempo médio (ganho de 49%), mantendo 100% das funcionalidades planejadas.
Assis e Silva (2025)	Experimento controlado (*bootcamp*) com grupos controle e experimental	Protótipo Gemini-2.5-Flash	Automação da tradução de requisitos em código; redução estatisticamente relevante do esforço cognitivo e do tempo de codificação.
Colombo *et al.* (2025)	Estudo experimental com projetistas de software orientados a objetos	LLMs diversos (ChatGPT, Claude)	LLMs auxiliam projetistas inexperientes na identificação e aplicação correta de padrões GoF em etapas iniciais.

Queiroz e Lima (2025)	Experiência educacional com 16 estudantes em elicitação de requisitos	ChatGPT-3.5	IA corroborou 86,02% das decisões de priorização para MVPs; alunos relataram desconfiança pontual.
Doi *et al.* (2026)	Estudo de caso com estudantes de Licenciatura em Computação	IAGen (ChatGPT)	Identificação eficaz de falhas lógicas em diagramas de classe UML; limitações em elementos gráficos; auditoria híbrida estimulada.
Nabarro *et al.* (2026)	Estudo prático com 22 estudantes (PBL + Metodologias Ágeis)	IAGen (ChatGPT)	Alto desempenho em elicitação de requisitos e modelagem funcional; dificuldades na modelagem estrutural.
Siddiq *et al.* (2022)	Estudo empírico em 16.400 *snippets* e *datasets* de código	GitHub Copilot / GPT-Code-Clippy	Modelos generativos propagam *code smells* e funções potencialmente inseguras presentes nas bases de treinamento.

Fonte: *Elaborado pelos autores (2026), com base nos dados compilados das fontes pesquisadas.*

Tendo em vista essa lacuna observada na literatura, esta pesquisa tem como objetivo preencher tal deficiência. A maioria dos trabalhos publicados trata separadamente de melhorias de velocidade imediatas ou de aspectos puramente pedagógicos. Faltam, entretanto, trabalhos que considerem a relação entre a melhora na velocidade de codificação, degradação de arquiteturas e custo escondido de manutenção em sistemas reais. Neste sentido, a

presente pesquisa procura justamente sintetizar estas três dimensões num estudo de integração das correlações entre produtividade, qualidade e segurança, baseando-se em recentes evidências.

4. METODOLOGIA

Trata-se de um estudo de abordagem mista (quali-quantitativa) de caráter aplicado e de objetivos exploratório-descritivo, tendo sido realizado um estudo de revisão integrativa da literatura e análise de síntese empírica de dados (Creswell, 2014; Bogdan; Biklen, 1994).

A coleta de dados bibliográficos foi executada nas seguintes bases de dados acadêmicas e bibliotecas digitais: IEEE Xplore, ACM Digital Library, Scopus, Google Scholar, SciELO e na biblioteca digital da Sociedade Brasileira de Computação (SBC OpenLib - SOL). A *string* usada foi a seguinte: ("generative AI" OR "LLM" OR "ChatGPT" OR "GitHub Copilot") AND ("software engineering" OR "engenharia de software”) AND (“productivity” OR "code quality" OR "code smells" OR “security”).

Os critérios de inclusão estabelecidos foram artigos completos publicados entre 2022 e 2026, estudos em português ou inglês, trabalhos com avaliação empírica ou relatos rigorosos de experiência sobre o uso de IAGen na Engenharia de Software. Os critérios de exclusão descartaram artigos duplicados, editoriais curtos sem validação empírica e trabalhos focados exclusivamente em robótica ou hardware sem relação direta com código-fonte.

Por fim, os estudos selecionados foram organizados em três categorias: produtividade e métricas de tempo; qualidade estrutural e arquitetura do código; e segurança, confiabilidade e governança. A

comparação considerou as ferramentas utilizadas, os métodos empregados, os principais resultados e as limitações de cada pesquisa, permitindo identificar os benefícios e os desafios associados à utilização da IAGen na Engenharia de Software.

5. RESULTADOS E DISCUSSÃO

5.1. Impactos na Produtividade - Ganhos de Tempo e Fluxo de Trabalho

Os resultados compilados evidenciam que a Inteligência Artificial Generativa proporciona ganhos expressivos de velocidade nas fases iniciais e repetitivas da codificação. Yamada e Mendes (2026) realizaram um estudo experimental na Universidade do Estado do Pará, onde vários desenvolvedores desenvolveram a mesma aplicação, porém, alguns com auxílio de IAGen, e outros sem esta ajuda. O resultado mostrou que, sem a IA, os desenvolvedores levaram em média 1h15min para construir a aplicação; enquanto com o auxílio da IA, eles levaram apenas 38min para fazer o mesmo trabalho. Ou seja, a utilização da IA salvou cerca de 49,3% do tempo, e não houve perda nas funcionalidades implementadas.

Assim como encontramos em Assis e Silva (2025), que verificaram a redução significativa do esforço cognitivo dos programadores utilizando um protótipo que emprega o modelo Gemini-2.5-Flash para automatizar a tradução de requisitos em código. De forma semelhante, Queiroz e Lima (2025) analisaram a utilização do ChatGPT durante a engenharia de requisitos, constatando que esse modelo conseguiu corroboração de 86,02% das decisões de priorização de requisitos tomadas pelos alunos para a elaboração de

Produtos Mínimos Viáveis (MVPs). Isso contribuiu para a redução do tempo necessário para a equipe chegar a um consenso.

Porém, esses dados mostram que a melhoria na produtividade não é linear. O tempo economizado na primeira etapa de geração de código é parcialmente consumido na etapa posterior de revisão humana, depuração e adaptação do código gerado ao contexto específico do projeto. Este fenômeno, documentado por Ferreira (2025) e Nguyen e Nadi (2022), sugere que o benefício real da IA generativa reside menos na velocidade bruta e mais na redução do esforço cognitivo em tarefas repetitivas e de menor valor estratégico, liberando o desenvolvedor para atividades de maior complexidade criativa e arquitetural.

5.2. Qualidade do Código Gerado: Análise de Métricas Estruturais e *Code Smells*

A avaliação da qualidade interna revelou um cenário intrincado e contraditório. Estudo quantitativo feito por Ferreira (2025), em 23 repositórios de código *open-source* Java, que analisou 115 *issues* e 281 classes editadas pelo GitHub Copilot, fornece dados importantes sobre a arquitetura do código gerado automaticamente.

Em relação à coesão (métrica LCOM*), a atuação da IA manteve-se neutra em 55,7% dos casos, piorou a coesão em 27,9% e melhorou em apenas 16,4%. Na herança (DIT), a IA demonstrou estabilidade, respeitando as superclasses existentes. Todavia, as métricas de CBO, RFC e WMC apresentaram uma clara tendência de degradação. Isso significa que, além do Copilot estar introduzindo acoplamento excessivo do tipo dependência externa, a arquitetura de

acoplamento está se tornando muito complicada, formando um "dominó" que dificulta futuras alterações.

Além disso, Ferreira (2025) encontrou que 49,5% das implementações feitas pela IA trouxeram novos *code smells*. Em particular, 52,3% destes *code smells* são considerados de severidade Média e 27,1% Alta pelo SonarQube. A média estimada do tempo gasto para corrigir estes débitos técnicos foi de 38,1 minutos por tarefa com erro. Doi *et al.* (2026) e Nabarro *et al.* (2026) também verificaram que os LLMs são eficazes em detectar erros lógicos no texto, mas tímidos na consistência estrutural nos diagramas de classe UML. Isto sugere que os LLMs possuem alguma dificuldade em entender relações visuais complexas de dependência.

5.3. Riscos e Desafios: Segurança, Alucinações Técnicas e Governança

A automação desassistida introduz riscos severos à segurança e à confiabilidade do software. O principal risco refere-se à reprodução involuntária de vulnerabilidades conhecidas como as listadas no OWASP Top 10 e práticas de codificação inseguras ou obsoletas presentes nos dados de treinamento dos modelos (Siddiq *et al.*, 2022). Como os LLMs foram treinados em repositórios públicos sem curadoria sistemática de segurança, padrões vulneráveis como injeção de SQL, armazenamento inseguro de credenciais e validação inadequada de entradas persistem nas sugestões geradas automaticamente.

O fenômeno das alucinações técnicas em que o modelo gera funções plausíveis porém sintaticamente incorretas ou referenciando bibliotecas inexistentes exige supervisão constante.

Outro risco crítico é o viés de super confiança ou dependência tecnológica, em que desenvolvedores aceitam sugestões da IA sem a devida auditoria de código, acumulando vulnerabilidades ocultas e riscos de violação de licenciamento de software (uso inadvertido de código protegido por *copyleft*) e da LGPD (vazamento de dados sensíveis incluídos em prompts enviados a APIs externas) (Clark *et al.*, 2024; Russo, 2024).

5.4. Síntese Integrada: O Paradoxo da Produtividade na Era da IA

Ao juntar todos os pontos encontramos o que chamamos de paradoxo da produtividade na era da IA: ganha-se em termos de velocidade na fase de implantação (podendo reduzir até 49% do tempo original), mas perde-se com a conta de cheques futura no custo de refatoração (a média de débito técnico por tarefa com erro foi de 38,1 minutos) e na dificuldade cada vez maior de manter sistemas de código fortemente acoplado. A IA generativa também se mostrou muito eficiente como assistente tático na implantação pontual, pois ajuda bastante a agilizar e diminuir as barreiras cognitivas de tarefas individuais para desenvolvedores menos experientes. Porém, não consegue ver a árvore da floresta, o que é necessário para se assegurar a sustentabilidade e evolução controlada de sistemas de médio e grande porte.

5.5. Boas Práticas e Recomendações para a Adoção Responsável

As boas práticas abaixo foram formuladas a partir das descobertas apresentadas nos estudos revisados, visando minimizar os riscos e otimizar os ganhos em relação ao uso da IA generativa:

1. Auditoria Humana Híbrida

   Todos os códigos sugeridos pela IA devem passar

obrigatoriamente pelo processo de code *review* entre os programadores. Cuidado especial deve ser tomado com módulos que envolvam segurança, autenticações e acesso a dados.

2. Integração de Análise Estática
   Também é importante integrar as ferramentas de análise estática como SonarQube, ESLint ou Checkstyle no *pipeline* de CI/CD, garantindo que *code smells* não sejam mesclados na *branch* principal.

3. Testes Automatizados Rigorosos
   É necessário ampliar a cobertura de testes unitários e de integração que não utilizaram a IA. Esses testes devem verificar se a funcionalidade do código gerado pela IA funciona corretamente.

4. Políticas de Governança Claras
   Além disso, algumas políticas devem ser definidas na organização. É importante garantir que dados sensíveis presentes nos *prompts* sejam protegidos. Além disso, é necessário garantir que licenças *open source* estejam em conformidade. Também é vital garantir que o time receba educação contínua sobre as limitações e riscos das ferramentas de IA.

6. AMEAÇAS À VALIDADE E LIMITAÇÕES

Este estudo possui algumas limitações que devem ser consideradas na interpretação dos resultados. Os estudos empíricos realizados focaram na linguagem Java e no agente GitHub Copilot (GPT-5 mini), o que pode afetar a validade externa do estudo. Portanto, a

aplicação de resultados em outras linguagens de programação dinâmica (Python, JavaScript) ou outros LLMs (Claude Sonnet, DeepSeek R1) requer cuidado, pois eles podem ter diferentes comportamentos de arquitetura.

O fato de estarmos em um período de rápido desenvolvimento de ferramentas de IA traz consigo um viés temporal. Modelos estão sendo lançados constantemente. Isso significa que algumas das limitações apresentadas nesses estudos podem ser corrigidas em versões futuras. Ou então piorar. Isso reforça ainda mais a necessidade de pesquisas longitudinais repetidas que acompanhem o progresso das ferramentas de IA e como elas impactam a qualidade dos sistemas de softwares que usam IA generativa para manter.

7. CONSIDERAÇÕES FINAIS

O artigo investigou os impactos da Inteligência Artificial Generativa na Engenharia de Software. Foram investigados três aspectos: (i) produtividade; (ii) qualidade do código e (iii) segurança durante o desenvolvimento de sistemas. Em relação à problemática de pesquisa colocada, pode-se concluir que, sim, a IAGen traz melhorias reais e significativas no ritmo de desenvolvimento, diminuindo em quase 50% o tempo gasto na fase de codificação e contribuindo para a tomada de decisão nos processos de análise de requisito e projeto. Contudo, esses ganhos imediatos criam novos riscos estruturais quando a ferramenta é adotada de forma acrítica e sem governança técnica adequada.

Os resultados demonstraram que o código gerado por LLMs tende a elevar o acoplamento entre classes e introduzir *code smells* em

cerca de metade das tarefas, gerando uma taxa expressiva de débito técnico que exige refatoração humana contínua. As principais contribuições do presente trabalho são apresentar uma visão empírica sintetizada que chama a atenção para a necessidade de governança técnica na utilização de assistentes de IA generativa em projetos reais de Engenharia de Software. Assim, o trabalho contribui com o estado da arte da literatura nacional em relação a esta nova temática.

Os possíveis trabalhos futuros seriam realizar experimentos controlados a longo prazo que avaliassem os diferentes LLMs (GitHub Copilot, Claude Sonnet, DeepSeek R1, etc.) em projetos corporativos grandes. Tais experimentos deveriam medir não apenas métricas de velocidade, como também indicadores de saúde da arquitetura ao longo do tempo. Além disso, seria interessante investigar os *frameworks* de engenharia de *prompt* focados em manter a arquitetura do software limpa, minimizar automaticamente débitos técnicos e verificar ativamente se o software segue padrões de segurança durante todo o ciclo de vida do desenvolvimento.

REFERÊNCIAS BIBLIOGRÁFICAS

ASSIS, Filipe Borges Franco de; SILVA, Lucas Portela Costa da. **O uso da IA generativa no aumento da produtividade no desenvolvimento de software.** 2025. Trabalho de Conclusão de Curso (Bacharelado em Sistemas de Informação) – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais (IFMG), Ouro Branco, 2025.

BOGDAN, Robert C.; BIKLEN, Sari Knopp. **Investigação qualitativa em educação:** uma introdução à teoria e aos métodos. Porto: Porto Editora, 1994.

CLARK, Andrew et al. A quantitative analysis of quality and consistency in AI-generated code. In: INTERNATIONAL CONFERENCE ON SOFTWARE AND SYSTEM ENGINEERING (ICoSSE), 7., 2024. **Proceedings** [...]. IEEE, 2024. p. 37-41.

COLOMBO, Vitor Monteiro et al. Avaliação do uso de LLMs para suporte à tomada de decisão na adoção de padrões de projeto em software orientado a objetos. In: SIMPÓSIO BRASILEIRO DE COMPONENTES, ARQUITETURAS E REUTILIZAÇÃO DE SOFTWARE (SBCARS), 19., 2025, Recife. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2025. p. 101-111. DOI: https://doi.org/10.5753/sbcars.2025.14602

CRESWELL, John W. **Investigação qualitativa e projeto de pesquisa: escolhendo entre cinco abordagens.** 3. ed. Porto Alegre: Penso, 2014.

DESHMUKH, Rushali et al. Optimizing human-AI interaction: innovations in prompt engineering. In: INTERNATIONAL CONFERENCE ON INTELLIGENT DATA COMMUNICATION TECHNOLOGIES AND INTERNET OF THINGS (IDCIoT), 3., 2025. **Proceedings** [...]. IEEE, 2025. p. 1240-1246.

DOI, Yudi Vinicios Ribeiro et al. O uso de inteligência artificial generativa no ensino de diagramas de classes: um estudo de caso em contexto educacional. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO (SBIE), 37., 2026, Goiânia. **Anais** [...].

Porto Alegre: Sociedade Brasileira de Computação, 2026. p. 1458-1473. DOI: https://doi.org/10.5753/sbie.2026.27844.

FERREIRA, Vitor Lany Freitas. **Impacto do código gerado por inteligência artificial na manutenibilidade do software.** 2025. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Software) – Pontifícia Universidade Católica de Minas Gerais, Belo Horizonte, 2025.

FORSGREN, Nicole; HUMBLE, Jez; KIM, Gene. **Accelerate:** the science of lean software and DevOps. Portland: IT Revolution Press, 2018.

GITHUB; OPENAI. **GitHub Copilot:** your AI pair programmer. 2022. Disponível em: https://github.com/features/copilot. Acesso em: 15 fev. 2026.

GOOGLE DEEPMIND. **Gemini:** a family of highly capable multimodal models. 2023. Disponível em: https://deepmind.com/gemini. Acesso em: 10 jan. 2026.

IEEE. **IEEE standard glossary of software engineering terminology.** IEEE Std 610.12-1990. New York: IEEE, 1990.

MARTIN, Robert C. **Código limpo:** habilidades práticas do Agile software. Rio de Janeiro: Alta Books, 2019.

NABARRO, Cristina Becker M. et al. Uma abordagem baseada em inteligência artificial generativa, PBL e metodologias ágeis para o ensino de modelagem de sistemas. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO (SBIE), 37., 2026, Goiânia. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2026. p. 945-957. DOI: https://doi.org/10.5753/sbie.2026.27415

NGUYEN, Nhan; NADI, Sarah. An empirical evaluation of GitHub Copilot's code suggestions. In: IEEE/ACM INTERNATIONAL CONFERENCE ON MINING SOFTWARE REPOSITORIES (MSR), 19., 2022. **Proceedings** [...]. IEEE/ACM, 2022. p. 1-5. DOI: https://doi.org/10.1145/3524842.3528470

PRESSMAN, Roger S.; MAXIM, Bruce R. **Engenharia de software:** uma abordagem profissional. 9. ed. Porto Alegre: McGraw-Hill, 2021.

QUEIROZ, Francisca Karolina; LIMA, Márcia Sampaio. Uso do ChatGPT na priorização de requisitos: uma experiência educacional em engenharia de software. In: SIMPÓSIO BRASILEIRO DE EDUCAÇÃO EM COMPUTAÇÃO (EDUCOMP), 5., 2025, Juiz de Fora. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2025. p. 491-501. DOI: https://doi.org/10.5753/educomp.2025.5370.

RUSSO, Daniel. Navigating the complexity of generative AI adoption in software engineering. **ACM Transactions on Software Engineering and Methodology** (TOSEM), v. 33, n. 5, p. 1-28, 2024. DOI: https://doi.org/10.1145/3652154

SIDDIQ, Mohammed Latif et al. An empirical study of code smells in transformer-based code generation techniques. In: IEEE INTERNATIONAL WORKING CONFERENCE ON SOURCE CODE ANALYSIS AND MANIPULATION (SCAM), 22., 2022. **Proceedings** [...]. IEEE, 2022. p. 71-82.

SOMMERVILLE, Ian. **Engenharia de software.** 10. ed. São Paulo: Pearson, 2019.

VASWANI, Ashish et al. Attention is all you need. In: ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS (NeurIPS), 30., 2017.

Proceedings [...]. NIPS, 2017. p. 5998-6008. DOI: https://doi.org/10.48550/arXiv.1706.03762

WHITE, Jules et al. ChatGPT prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In: **GENERATIVE AI FOR EFFECTIVE SOFTWARE DEVELOPMENT.** Cham: Springer, 2024. p. 71-108. DOI: https://doi.org/10.48550/arXiv.2303.07839

YAMADA, Armando Isamu de Almeida; MENDES, Gleisson Amaral. Impactos da inteligência artificial generativa no desenvolvimento de software: percepção de profissionais e avaliação experimental. **Revista Ibero-Americana de Humanidades, Ciências e Educação,** v. 12, n. 8, p. 1-23, 2026.

---

[1] Discente do Curso de Bacharel em Engenharia de Software da Universidade de Vassouras, Campus Maricá. Av. Roberto Silveira, 1954, Flamengo, Maricá, RJ, CEP 24903-815. ORCID: https://orcid.org/0009-0007-8356-6948. E-mail: acesse o artigo original para visualizar o e-mail

[2] Discente do Curso de Bacharel em Engenharia de Software da Universidade de Vassouras, Campus Maricá. Av. Roberto Silveira, 1954, Flamengo, Maricá, RJ, CEP 24903-815. ORCID: https://orcid.org/0009-0007-8473-3955. E-mail: acesse o artigo original para visualizar o e-mail

[3] Discente do Curso de Bacharel em Engenharia de Software da Universidade de Vassouras, Campus Maricá. Av. Roberto Silveira, 1954, Flamengo, Maricá, RJ, CEP 24903-815. ORCID:

https://orcid.org/0009-0004-9146-2012. E-mail: acesse o artigo original para visualizar o e-mail

[4] Discente do Curso de Bacharel em Engenharia de Software da Universidade de Vassouras, Campus Maricá. Av. Roberto Silveira, 1954, Flamengo, Maricá, RJ, CEP 24903-815. ORCID: https://orcid.org/0009-0004-2650-8571. E-mail: acesse o artigo original para visualizar o e-mail

[5] Doutora em Bioquímica pela Universidade Federal do Rio de Janeiro. Docente do Curso de Bacharel em Engenharia de Software da Universidade de Vassouras, Campus Maricá. Av. Roberto Silveira, 1954, Flamengo, Maricá, RJ, CEP 24903-815. ORCID: https://orcid.org/0000-0003-3825-7242. E-mail: acesse o artigo original para visualizar o e-mail

[6] Mestre em Gestão do Trabalho pela Universidade Santa Úrsula. Docente do Curso de Bacharel em Engenharia de Software da Universidade de Vassouras, Campus Maricá. Av. Roberto Silveira, 1954, Flamengo, Maricá, RJ, CEP 24903-815. ORCID: https://orcid.org/0000-0002-9377-5993. E-mail: acesse o artigo original para visualizar o e-mail