UNIVERSIDADE DO VALE DO RIO DOS SINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

GUILHERME CLOSS FRAGA

O IMPACTO DA INTELIGÊNCIA ARTIFICIAL NO PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE*:
Um estudo comparativo entre desenvolvedores com e sem o uso de ferramentas de IA

São Leopoldo
2025

GUILHERME CLOSS FRAGA

O IMPACTO DA INTELIGÊNCIA ARTIFICIAL NO PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE*:
Um estudo comparativo entre desenvolvedores com e sem o uso de ferramentas de IA

Artigo apresentado como requisito parcial para obtenção do título de Bacharel em Ciência da Computação, pelo Curso de Ciência da Computação da Universidade do Vale do Rio dos Sinos (UNISINOS)

Orientador(a): Prof.ª Dra. Rosemary Francisco

São Leopoldo
2025

O IMPACTO DA INTELIGÊNCIA ARTIFICIAL NO PROCESSO DE DESENVOLVIMENTO DE *SOFTWARE*: Um estudo comparativo entre desenvolvedores com e sem o uso de ferramentas de IA

Guilherme Closs Fraga[1]
Rosemary Francisco[2]

Resumo: Lorem ipsum.

Palavras-chave: lorem ipsum.

Abstract: Lorem ipsum.

Keywords: lorem ipsum.

1 INTRODUÇÃO

Lorem ipsum.

1.1 Proposta de Solução

Lorem ipsum.

1.2 Objetivos

Lorem ipsum.

1.2.1 Objetivo Geral

Lorem ipsum.

1.2.2 Objetivos Específicos

Lorem ipsum.

a) lorem ipsum;

b) lorem ipsum;

c) lorem ipsum;

[1]Graduando em Ciência da Computação pela Unisinos. Email: guifraga8@edu.unisinos.br
[2]Doutora em Administração. Email: rosemaryf@unisinos.br

d) lorem ipsum;

e) lorem ipsum;

f) lorem ipsum.

2 FUNDAMENTAÇÃO TEÓRICA

Com base na temática que o presente trabalho visa abordar, se faz necessário discorrer sobre dois tópicos fundamentais para a construção e consolidação do mesmo, sendo eles: Desenvolvimento de *Software* e IA Generativa aplicada ao desenvolvimento de *software*. Nas seções seguintes, os conceitos são explorados e detalhados a fim de estabelecer os fundamentos teóricos necessários para compor este trabalho.

2.1 Desenvolvimento de *Software*

Um *software* refere-se a todos os componentes não físicos de um computador, redes de computadores ou dispositivos móveis. Em linhas gerais, refere-se aos programas e aplicações, como o próprio sistema operacional, que fazem aquele aparelho (ou ferramenta) operar de acordo com o que o usuário necessita. Isto é, *software* se trata de uma coleção de instruções, dados ou programas utilizados para operar computadores e executar determinadas tarefas (COUTINHO; BEZERRA, 2021).

Em outras palavras, pode-se dizer que é um termo genérico que se refere à aplicações, *scripts* e programas executáveis em um determinado dispositivo. Assim, o diferencia de *hardware*, que descreve os aspectos físicos de um aparelho. Sendo assim, podemos estabelecer o *software* como um componente variável de um computador, enquanto o *hardware* é constante (SAKURAI; ZUCHI, 2018)

De acordo com Maynard (2015), temos basicamente dois tipos de *software*, sendo eles: *software* de aplicativo e *software* de sistema. Um aplicativo seria aquele que atende uma necessidade específica ou executa determinadas tarefas. E, segundo Schwab (2019), *software* de sistema seria a parte essencial para que o *hardware* de um computador opere, servindo como a plataforma para a execução de aplicativos. Ademais, podemos mencionar outros tipos de *software*, como os de programação, que são ferramentas necessárias para desenvolvedores, *middlewares*, responsáveis por atuarem entre o *software* do sistema e aplicativos e até mesmo *drivers*, que são responsáveis por operarem dispositivos e periféricos de computadores.

Por fim, a sincronia entre diferentes tipos de *software* e *hardware*, é peça fundamental para o funcionamento eficiente de sistemas computacionais. Enquanto um aplicativo permite que seus usuários realizem determinadas tarefas, um sistema assegura que o *hardware* opere corretamente, dando suporte para diferentes aplicações. Com a constante crescente de complexidade em sistemas modernos, a presença de um *middleware* eficaz se torna indispensável para assegurar que diferentes *softwares* funcionem em harmonia. Da mesma forma que, *drivers* são

indispensáveis para comunicação entre o sistema e periféricos, o que amplia as funcionalidades de um computador, resultando na melhoria da experiência do usuário.

Sendo assim, os componentes técnicos de um computador são controlados por *softwares*. Isto igualmente o difere do *hardware* de um computador, rede ou dispositivo móvel, visto que ele não é fisicamente tangível. Cândido (2022) diz que o sistema operacional nada mais é do que o *software* fundamental de um computador ou dispositivo móvel. No que diz respeito aos termos de interface para o usuário, os três sistemas operacionais mais conhecidos, sendo eles *Windows*, *macOS* e *Linux*, diferem. Outros programas e aplicações podem servir como complemento para as funções básicas de um sistema operacional. Eles podem ser instalados por meio de um *driver* ou após *download* e instalação. Estes programas incluem, por exemplo, o próprio *MediaPlayer* ou *Microsoft Office* da *Microsoft* (NETO, 2019).

Além dos sistemas operacionais, existe uma ampla gama de *softwares* utilitários voltados para funções específicas que visam aprimorar tanto o desempenho quanto a segurança do computador. Entre eles, destacam-se os programas antivírus, as ferramentas de *backup* e os aplicativos de manutenção de disco, que atuam na proteção das informações e na preservação da eficiência do sistema. Somado a isso, o avanço tecnológico possibilitou o surgimento dos *softwares* de virtualização, que permitem a execução de diferentes sistemas operacionais em uma única máquina física, ampliando a flexibilidade e otimizando a utilização dos recursos de *hardware*.

Em diversos dispositivos, o *software* está diretamente integrado ao hardware, como ocorre em painéis eletrônicos de automóveis ou em máquinas de lavar. Exemplos comuns são os sistemas de navegação e os leitores de *Blu-Ray*, que funcionam sob controle de *software*, mas não permitem substituição por alternativas diferentes. Segundo Coutinho e Bezerra (2021), essas combinações fixas recebem a denominação de "embutidos". Já na década de 1950, *software* e *hardware* eram concebidos como uma única unidade, em que o *software* era tratado apenas como parte do *hardware*, conhecido então como código de programa. Foi o estatístico John W. Tukey quem utilizou pela primeira vez o termo *"software"* em 1958 (MAYNARD, 2015). Conforme apontam Sakurai e Zuchi (2018), somente cerca de vinte anos depois o governo dos Estados Unidos determinou que a IBM passasse a contabilizar seus componentes de *hardware* e *software* separadamente, decisão que abriu espaço para o fortalecimento de empresas especializadas no desenvolvimento de *software*.

O avanço tecnológico tornou possível a dissociação entre *software* e *hardware*, proporcionando maior flexibilidade e impulsionando a inovação no desenvolvimento de sistemas computacionais. Essa separação permitiu que os programas fossem atualizados sem a necessidade de alterações no *hardware*, acelerando os ciclos de inovação. A especialização na produção de *software* também deu origem a uma indústria sólida, formada por empresas voltadas à criação de soluções personalizadas para diferentes setores e demandas. Esse movimento não só ampliou as formas de utilização dos dispositivos, como também favoreceu o surgimento de novos modelos de negócio baseados em *software*, os quais desempenham papel central na atual economia digital.

Schwab (2019) destaca que o trabalho do programador consiste, de forma essencial, em traduzir requisitos e algoritmos para uma linguagem de programação que o computador consiga interpretar e executar. A partir desse processo, é possível desenvolver diferentes tipos de *software*, como aplicativos, jogos e sistemas de controle para robôs. Um exemplo histórico ligado à ideia de programação foi a invenção de um *tear* "programável" pelo francês Joseph-Marie Jacquard, que representou um marco na Revolução Industrial. O equipamento não possuía processador, sendo controlado por meio de cartões perfurados. Conforme explica Maynard (2015), esses cartões não utilizavam valores binários como 0 e 1, mas funcionavam através da presença ou ausência de perfurações: um furo indicava uma ação de movimento, enquanto a ausência significava outra. A criação de Jacquard é considerada uma das precursoras da automação moderna.

Maynard (2015) aponta que, em 1843, Augusta Ada King-Noel, mais conhecida como Ada Lovelace, desempenhou um papel marcante ao colaborar com Charles Babbage no projeto do *Analytical Engine*, uma calculadora mecânica de uso geral que acabou não sendo concluída. Durante esse processo, Ada elaborou diversas anotações e descreveu um método para calcular os números de Bernoulli por meio da máquina, o que, segundo Coutinho e Bezerra (2021), é considerado o primeiro programa de computador da história. Posteriormente, em 1941, Konrad Ernst Otto Zuse desenvolveu o Z3, reconhecido como o primeiro sistema de computação totalmente automatizado e programável do mundo.

O Z3 foi construído a partir dos conceitos idealizados por Babbage e tinha como diferencial a capacidade de realizar operações, como a adição, em menos de um quarto de segundo. O propósito central de sua criação estava ligado à busca por maior eficiência nos cálculos. Nos primeiros anos, os *softwares* eram desenvolvidos para máquinas específicas e eram comercializados em conjunto com o *hardware*. Já na década de 1980, passaram a ser distribuídos em mídias físicas, como disquetes, e mais tarde em CDs e DVDs. Atualmente, grande parte dos *softwares* é obtida por meio da *Internet*, seja diretamente nos sites dos fabricantes ou através de plataformas de distribuição de aplicativos.

Os modelos de ciclo de vida de *software* descrevem as etapas que um sistema percorre desde sua concepção até sua desativação, organizando a sequência de atividades necessárias para o desenvolvimento. O objetivo central desses modelos é garantir qualidade, reduzir custos e otimizar o tempo de entrega. Entre as fases mais comuns estão:

- Análise de Requisitos: onde são levantadas e documentadas as necessidades do sistema;
- Design: define a arquitetura, fluxos de trabalho e tecnologias a serem utilizadas;
- Implementação: fase em que o código é desenvolvido com base no planejamento estabelecido;
- Testes: que envolvem diferentes tipos de verificação para assegurar a confiabilidade do *software*;
- Manutenção: responsável por corrigir falhas e implementar melhorias após a entrega do

sistema.

De acordo com Cândido (2022), observa-se nos últimos anos uma tendência de desprofissionalização no aprendizado de programação, já que muitas pessoas têm adquirido essas habilidades de forma autodidata, recorrendo a tutoriais *online* em vez de cursos formais. Além disso, o *networking* global, aliado às diversas plataformas digitais, tem favorecido a formação de grandes comunidades que reúnem tanto entusiastas quanto programadores experientes, tornando cada vez menos nítida a distinção entre esses grupos.

Ao mesmo tempo, percebe-se um avanço significativo nos padrões e na qualidade do desenvolvimento de *software*. Os sistemas estão cada vez mais sofisticados e complexos, o que aumenta a demanda por soluções digitais. Esse contexto reflete uma sociedade em constante transformação, onde as necessidades mudam rapidamente, gerando uma procura crescente por Desenvolvedores *Full Stack* (um profissional de tecnologia que possui a capacidade de trabalhar em todas as camadas de um projeto de *software*) em detrimento de profissionais especializados apenas em uma área.

A complexidade crescente dos sistemas requer profissionais capazes de atuar em todas as camadas do desenvolvimento, abrangendo desde o *Frontend* (interface do usuário) até o *Backend* (servidor, banco de dados e lógica de negócios de uma aplicação). Nesse cenário, os Desenvolvedores *Full Stack* ganham destaque por sua flexibilidade e capacidade de adaptação às mudanças do mercado e às demandas dos usuários. Sua habilidade em compreender e integrar diferentes componentes de um sistema favorece um desenvolvimento mais unificado e eficiente, além de melhorar a comunicação e colaboração dentro das equipes. Assim, em um ambiente em que a agilidade e a adaptabilidade são fatores essenciais, esses profissionais se consolidam como recursos estratégicos para empresas que buscam manter a competitividade.

O desenvolvimento de *software* pode ser entendido como um conjunto de atividades voltadas ao projeto, criação, implementação e manutenção de programas que orientam o funcionamento de computadores, sendo independente do *hardware* e responsável por torná-los programáveis (SCHWAB, 2019). Segundo Maynard (2015), esses programas podem ser classificados em três categorias principais:

- *Softwares* de sistema: englobam sistemas operacionais, utilitários e ferramentas de gerenciamento;
- *Softwares* de programação: oferecem recursos como compiladores e depuradores para os desenvolvedores;
- *Softwares* aplicativos: voltados ao usuário final, como ferramentas de produtividade, aplicativos de segurança e plataformas digitais, incluindo serviços online e aplicativos móveis.

Outro tipo de *software* é aquele voltado para sistemas embarcados, usados no controle de máquinas e dispositivos que nem sempre são reconhecidos como computadores, como redes telefônicas, veículos automotores e robôs industriais. Esses equipamentos, juntamente com seus

softwares, podem ser conectados dentro do ecossistema da Internet das Coisas (*IoT*). Nesse contexto, programadores, engenheiros e desenvolvedores de *software* exercem funções essenciais no desenvolvimento desses sistemas, com responsabilidades que frequentemente se sobrepõem e cuja dinâmica colaborativa pode variar dependendo do departamento ou da comunidade em que atuam.

As soluções de engenharia de *software* baseiam-se em métodos científicos e precisam ser eficazes na prática, de forma semelhante a estruturas como pontes ou elevadores. À medida que os produtos se tornam mais inteligentes, incorporando microprocessadores, sensores e *software*, a responsabilidade dos engenheiros de *software* aumenta. Atualmente, a competitividade de um produto no mercado não depende apenas do *software* que ele utiliza, mas da integração eficiente entre todos os seus componentes.

Conforme aponta Cândido (2022), o desenvolvimento de *software* deve ser integrado ao desenvolvimento de produtos mecânicos e elétricos, o que faz com que o papel dos desenvolvedores de *software* seja menos delimitado em comparação ao dos engenheiros. Esses profissionais podem focar em áreas específicas do projeto, como a codificação, enquanto o ciclo de vida do *software* envolve a transformação de requisitos em funcionalidades, o trabalho em equipes multifuncionais, a gestão de processos e equipes, além de atividades de teste e manutenção.

De acordo com Schwab (2019), a atuação dos desenvolvedores de *software* vai além das equipes de programação, envolvendo também cientistas e fabricantes de dispositivos e *hardware*, que contribuem para a criação de códigos, mesmo não sendo os principais responsáveis pelo desenvolvimento. Essa participação não se limita às indústrias tradicionais de tecnologia da informação, abrangendo também empresas que não se dedicam exclusivamente à produção de *software* ou semicondutores.

A ampliação do papel dos desenvolvedores de *software* evidencia a crescente integração da tecnologia da informação com diversas áreas, como ciência, medicina, engenharia e artes. Atualmente, o desenvolvimento de *software* é fundamental para múltiplos setores, desde a indústria automotiva até o entretenimento, impulsionando a inovação e a transformação digital. Na indústria automotiva, esses profissionais são essenciais para o desenvolvimento de sistemas de condução autônoma e tecnologias avançadas de segurança. Já no cinema, os efeitos visuais e a animação digital dependem fortemente do *software* para criar mundos e personagens realistas. Dessa forma, os desenvolvedores de *software* exercem um papel cada vez mais diversificado e influente, moldando o futuro em diferentes áreas do conhecimento e da atividade humana.

2.1.1 Modelos de Desenvolvimento de *Software*

O desenvolvimento de *software* pode ser conduzido por diferentes abordagens, conhecidas como modelos de processo de *software*. Esses modelos têm como objetivo estruturar e organizar as atividades necessárias para transformar requisitos em um produto final de qualidade. Entre os modelos mais conhecidos estão o Modelo Cascata, o Modelo em V, o Modelo Espiral e os

Modelos Ágeis.

O Modelo Cascata (ou *Waterfall*) foi um dos primeiros modelos formais de processo de software. Proposto por Winston Royce em 1970, consiste em uma sequência linear de fases: levantamento de requisitos, análise, projeto, implementação, testes, implantação e manutenção. Cada fase deve ser concluída antes do início da próxima, funcionando como uma "cascata" de atividades (SOMMERVILLE, 2011). Entre suas vantagens podemos citar:

- Estrutura simples e de fácil entendimento;
- Definição clara das fases e entregas;
- Útil em projetos com requisitos bem definidos e estáveis.

Por outro lado, há também desvantagens que valem ser destacadas deste modelo, sendo elas:

- Pouca flexibilidade diante de mudanças nos requisitos;
- Risco de identificação tardia de falhas, geralmente apenas na fase de testes;
- Dificuldade em lidar com projetos complexos ou inovadores.

O Modelo em V é uma extensão do Modelo Cascata, mas com foco maior na validação e verificação em cada etapa do desenvolvimento. A representação em formato de "V" mostra que para cada fase de desenvolvimento existe uma fase correspondente de teste: requisitos são validados com testes de aceitação, projeto de sistema é validado com testes de integração, e assim sucessivamente (PRESSMAN, 2016). Entre suas principais vantagens, temos:

- Ênfase na qualidade e no teste desde o início do processo;
- Adequado para sistemas críticos, como os da área médica ou aeroespacial.

Já como desvantagens, temos:

- Semelhante ao Cascata, é rígido e pouco adaptável a mudanças;
- Elevado custo caso ocorra necessidade de alterações durante o desenvolvimento.

O Modelo Espiral, proposto por Barry Boehm em 1986, combina elementos do Modelo Cascata com técnicas de prototipação e gerenciamento de riscos. O desenvolvimento é representado como uma espiral, onde cada ciclo envolve: definição de objetivos, análise de riscos, desenvolvimento e validação, e planejamento da próxima iteração (BOEHM, 1986). Das vantagens deste modelo, ressaltam-se:

- Forte ênfase na análise e mitigação de riscos;
- Flexibilidade para incorporar mudanças ao longo do processo;
- Permite versões incrementais do sistema.

E, como desvantagens temos:

- Complexidade de gerenciamento maior do que em modelos lineares;
- Custo elevado, especialmente em projetos pequenos;

- Exige maior envolvimento de profissionais experientes em análise de riscos.

Os Modelos Ágeis surgiram como uma alternativa aos modelos tradicionais, sendo formalizados em 2001 com o Manifesto Ágil. Esses modelos priorizam colaboração com o cliente, entregas rápidas e contínuas, adaptação às mudanças e interações entre pessoas mais do que processos rígidos (BECK et al., 2001).

Entre os métodos ágeis mais utilizados, principalmente nos dias atuais, estão o *Scrum*, o *Kanban* e o *Extreme Programming* (conhecido também como *XP*). Todos eles seguem o princípio de dividir o projeto em pequenos ciclos (iterações), resultando em entregas frequentes e incrementais do produto. Como as principais vantagens, podemos citar:

- Alta flexibilidade para mudanças de requisitos;
- Entregas rápidas que geram valor contínuo ao cliente;
- Melhora na comunicação e na colaboração entre equipe e *stakeholders*.

E, das suas principais desvantagens:

- Pode gerar falta de documentação formal, dependendo da equipe;
- Requer alto envolvimento do cliente no processo;
- Menos indicado para projetos de missão crítica que demandam forte rastreabilidade.

Sendo assim, evidencia-se que cada modelo de desenvolvimento possui suas características, vantagens e limitações. A escolha do modelo mais adequado depende do contexto do projeto, da complexidade do sistema, do nível de risco, do grau de inovação e da estabilidade dos requisitos. Enquanto modelos tradicionais como Cascata e V são mais estruturados, os modelos Ágeis e o Espiral oferecem maior flexibilidade e adaptabilidade, características cada vez mais valorizadas no mercado atual. Na Tabela 1, temos um comparativo resumido dos principais pontos dos modelos citados.

Tabela 1 – Comparativo entre Modelos de Desenvolvimento de *Software*

Modelo	**Resumo**	**Vantagem**	**Desvantagem**	**Indicado para**
Cascata	Linear e sequencial	Simples e claro	Pouca flexibilidade	Projetos pequenos e estáveis
V	Cascata com foco em testes	Alta qualidade	Rígido a mudanças	Sistemas críticos
Espiral	Iterativo com análise de riscos	Flexível e seguro	Complexo e custo alto	Projetos grandes e arriscados
Ágeis	Iterativo e adaptável	Entregas rápidas	Pouca documentação	Projetos dinâmicos e mutáveis

Fonte: Elaborado pelo autor

2.1.2 Etapas do Processo de *Software*

Segundo Osterweil (2011), o desenvolvimento de um programa ou *software* voltado a atender às demandas de um cliente ou alcançar um propósito específico representa apenas um dos diversos resultados derivados de uma sequência de processos e atividades. Nesse cenário, Silva (2020) ressalta que os principais objetivos desse processo envolvem ampliar a compreensão sobre o tema tratado, criar meios eficientes de comunicação entre os envolvidos no projeto, assegurar a manutenção do sistema e coletar o retorno dos usuários. Dessa forma, destacam-se como metas centrais:

- Dispor de um guia que indique as tarefas a serem realizadas e as necessidades do cliente;
- Estimar os custos de desenvolvimento;
- Garantir padrões de qualidade;
- Projetar o tempo necessário para a execução.

Dessa forma, o processo de criação de *software* corresponde ao conjunto de métodos aplicados para elaborar programas de computador, independentemente de sua finalidade. Os modelos de desenvolvimento podem apresentar grande nível de detalhamento e complexidade, em razão da variedade de aplicações existentes, que vão desde jogos até editores de texto e ferramentas de *design*. Essas práticas envolvem não apenas os elementos técnicos necessários para o desenvolvimento, mas também a organização das equipes e a condução das diferentes fases do projeto.

Segundo Goes et al. (2022), mesmo que os aplicativos apresentem diferentes níveis de complexidade, a adoção de um processo de desenvolvimento é benéfica, pois aumenta as chances de êxito. Atualmente, coexistem métodos ainda em uso e outros já considerados obsoletos. Ao longo do projeto, partes da programação são concluídas progressivamente, com iterações intermediárias que incluem testes e validações constantes. Assim, grande parte das atividades de implementação e testes já se encontra avançada antes mesmo do término do processo, exigindo atenção contínua à gestão em todas as fases.

Esse modelo difere do processo em cascata, no qual as etapas seguem rigidamente em sequência. Em vez disso, cada iteração pode resultar em entregas parciais, permitindo que os clientes acompanhem o progresso e avaliem se os objetivos estão sendo alcançados. Isso favorece ajustes mais ágeis em cronograma e orçamento, além de facilitar a definição dos próximos passos.

Para Osterweil (2011), um modelo de processo de *software* consiste em adaptar estruturas já existentes a fim de simplificar a complexidade do desenvolvimento. Nesse contexto, o trabalho é dividido em períodos com escopo bem definido, sustentados por práticas de gerenciamento de projeto e controle de qualidade. Assim, o desenvolvimento pode seguir modelos aplicados de forma única, como o cascata, ou de modo iterativo, como o espiral, frequentemente com ajustes específicos conforme a necessidade.

O processo de desenvolvimento de *software* ocorre de forma iterativa, permitindo o aprimoramento contínuo dos elementos do programa. De modo geral, envolve dois grandes conjuntos de atividades: a implementação (*design* e programação) e a análise dos processos de negócio (modelos de processos e de dados). Segundo Medeiros (2019), a agilidade no desenvolvimento busca dar mais autonomia criativa aos desenvolvedores, reduzindo inicialmente a carga administrativa. Nesse cenário, surgem também abordagens alternativas, como a aplicação universal e a fábrica de *software*, que adaptam soluções pré-existentes às necessidades específicas de cada projeto.

De acordo com Silva (2020), os modelos de processo podem ser organizados em três tipos. O primeiro acompanha o desenvolvimento desde a concepção até a utilização em tempo real, incluindo as modificações necessárias após sua implantação. O segundo, conforme Osterweil (2011), estende-se ao ciclo de vida completo do *software*, abrangendo tanto os requisitos técnicos quanto a padronização e certificação das etapas. Já o terceiro se relaciona à visão do programador, valorizando métodos mais procedurais que podem facilitar ou dificultar a fluidez do processo, a depender da estratégia adotada.

Para reduzir riscos de atraso e de custos acima do previsto, é fundamental contar com uma arquitetura de *software* bem definida antes do início do desenvolvimento. Essa arquitetura orienta o planejamento e a organização do sistema, delimitando o ambiente de execução, os componentes a serem integrados e as restrições estruturais ou funcionais que precisam ser consideradas.

Segundo Chaves et al. (2020), mesmo em um cenário com arquitetura de *software* bem definida, a fase de desenvolvimento pode trazer desafios significativos e problemas imprevistos para a equipe. Como a escolha da metodologia depende do objetivo do produto, não existe um método universalmente superior. Projetos simples e com prazos curtos podem se beneficiar da abordagem em cascata, que reduz custos e tempo de entrega, garantindo maior previsibilidade.

Por outro lado, Wiegers e Beatty (2013) ressaltam que projetos mais complexos demandam métodos iterativos, já que o modelo cascata pode atrasar a identificação de falhas ou requisitos mal definidos, resultando em clientes insatisfeitos ou até em falhas graves. A abordagem iterativa permite ajustes ao longo do desenvolvimento, oferecendo maior flexibilidade e capacidade de adaptação a mudanças, conforme também destacado por Osterweil (2011).

Na prática, empresas de *software* adotam diferentes abordagens de acordo com suas necessidades e exigências de mercado. Em alguns casos, como no setor de defesa dos Estados Unidos, a aplicação está vinculada a certificações específicas, como a ISO 12207, que regula o ciclo de vida do *software*. Além disso, ao longo do tempo, pesquisadores têm buscado métodos repetitivos e previsíveis que aumentem a produtividade e a qualidade, seja por meio da padronização das atividades ou pela adoção de práticas de gestão de projetos.

Silva (2020) destaca que a falta de uma gestão de projetos eficiente pode levar a atrasos e aumento de custos no desenvolvimento de *software*, além de comprometer a entrega em termos de funcionalidade e prazo. Por isso, o gerenciamento eficaz é indispensável. Nesse contexto, a

definição de requisitos tem papel central: muitas vezes os clientes possuem apenas uma ideia geral do produto, sem clareza sobre seus objetivos, o que exige uma análise detalhada do escopo e a elaboração da especificação funcional.

O processo se inicia com a fase de planejamento, em que programadores, analistas de mercado, especialistas em *marketing* e clientes colaboram para definir as funcionalidades e os critérios de qualidade, além de avaliar riscos. Na sequência, ocorre a implementação, momento em que os desenvolvedores produzem o código-fonte e realizam testes de caixa preta e caixa branca para identificar falhas o quanto antes.

Outro aspecto essencial é a escolha da arquitetura de *software*, geralmente definida ainda no planejamento. Ela estabelece a estrutura do sistema, as interações entre componentes e as diretrizes de evolução, influenciando diretamente fatores como:

- Escalabilidade: capacidade do sistema crescer e lidar com maior carga de usuários;
- Manutenção: facilidade de incluir novas funcionalidades e corrigir falhas;
- Desempenho: eficiência em tempo de resposta e uso de recursos;
- Segurança: proteção de dados e resistência a ataques;
- Reusabilidade: aproveitamento de componentes em futuros projetos, economizando tempo e esforço.

Entre os diferentes modelos de arquitetura de *software*, destaca-se a Arquitetura Hexagonal (ou de Portas e Adaptadores), proposta por Alistair Cockburn. Seu princípio central é manter a lógica do sistema independente de elementos externos, como banco de dados, *frameworks* e interfaces. Essa separação favorece manutenção, testes e flexibilidade. Seus principais componentes são:

- Domínio central: reúne a lógica de negócios essencial, independente de tecnologias externas;
- Portas: definem a comunicação entre o domínio e o exterior;
  - Portas de entrada: permitem que adaptadores de entrada (ex.: serviços, controladores) acessem a lógica de negócios;
  - Portas de saída: usadas pela lógica de negócios para interagir com sistemas externos (ex.: repositórios, *gateways*);
- Adaptadores: implementações que conectam portas a sistemas externos;
  - Adaptadores de entrada: ligam o mundo externo ao domínio (ex.: *APIs REST*, interfaces gráficas);
  - Adaptadores de saída: conectam o domínio a serviços externos (ex.: bancos de dados, mensageria).

Essa arquitetura se destaca por facilitar testes isolados da lógica de negócios, sem dependência de fatores externos.

Além disso, as *APIs* (Interfaces de Programação de Aplicações) desempenham papel essencial na integração de sistemas, pois definem protocolos e formatos de comunicação padronizados. Aplicações modernas dependem fortemente delas para garantir interoperabilidade e escalabilidade. Um modelo bastante difundido é o *REST* (*Representational State Transfer*), um estilo arquitetural para serviços web que utiliza os métodos *HTTP*. Seus princípios incluem:

- Recursos: cada elemento é tratado como um recurso identificado por um *URI*;
- Métodos *HTTP*: uso de *GET*, *POST*, *PUT* e *DELETE* para operações *CRUD*;
- Representações: recursos descritos em formatos como *JSON* ou *XML*;
- *Stateless*: cada requisição contém todas as informações necessárias, sem depender de estado prévio;
- Cacheabilidade: respostas podem ser cacheadas para maior eficiência.

2.1.3 Ferramentas e Tecnologias

Ferramentas de desenvolvimento de *software* são peças fundamentais no ciclo de construção de um determinado projeto, oferencedo suporte às equipes envolvidas em todas as etapas do processo, desde a concepção até a implementação ou manutenção de sistemas. Tais ferramentas possuem uma vasta variedade de recursos, destinadas a agilizar o processo de desenvolvimento de uma aplicação.

Elas vão desde Ambientes de Desenvolvimento Integrado (*Integrated Development Environment* ou *IDEs*) até sistemas de controle de versão, testes e ferramentas que permitem análise de código. Portanto, explorar categorias e funcionalidades de diferentes ferramentas para o desenvolvimento de *software*, torna-se uma etapa crucial para entender qual ferramenta utilizar e como utilizar, de acordo com as tecnologias definidas para um determinado projeto. É a partir dessa decisão que se torna possível beneficiar uma aplicação, impulsionando sua eficiência e até mesmo, a qualidade do do desenvolvimento.

Para o projeto deste trabalho, a escolha das ferramentas/tecnologias *Node.js*, *Express.js*, *Postman*, *React* e *PostgreSQL* foi baseada em suas funcionalidades robustas, porém simples de serem implementadas, o que permite o desenvolvimento de aplicações bem estruturadas com um baixo nível de complexidade.

2.1.3.1 *Node.js* e *Express.js*

Node.js é um ambiente de execução *JavaScript* orientado a eventos, criado em 2009, que permite rodar aplicações do lado do servidor. Ele utiliza o motor V8 do *Google Chrome* para interpretar e executar o código, oferecendo alta performance e escalabilidade. Uma de suas principais vantagens é possibilitar que desenvolvedores utilizem *JavaScript* tanto no *Frontend* quanto no *Backend*, unificando a linguagem em toda a aplicação.

Entre suas características mais marcantes está o modelo não bloqueante e assíncrono, o que o torna especialmente eficiente em aplicações que exigem grande quantidade de operações de entrada e saída, como sistemas de tempo real, *APIs* e serviços de *streaming*. Esse modelo permite que o servidor lide com múltiplas conexões simultaneamente sem comprometer o desempenho.

Já o *Express.js* é um framework minimalista desenvolvido sobre o *Node.js*, com foco em simplificar a criação de aplicações *web* e *APIs*. Ele fornece uma estrutura mais organizada para lidar com rotas, requisições *HTTP*, respostas e *middlewares*, tornando o desenvolvimento mais ágil e menos complexo.

O *Express.js* adota uma arquitetura baseada em *middlewares*, que são funções executadas em sequência para processar requisições e respostas. Essa abordagem possibilita a personalização de fluxos e facilita a inclusão de funcionalidades como autenticação, validação de dados e tratamento de erros. Assim, os desenvolvedores conseguem estruturar aplicações de maneira modular e escalável.

Outro ponto importante é a vasta comunidade em torno de ambos. A quantidade de pacotes disponíveis no *npm* (*Node Package Manager*) permite ampliar rapidamente as funcionalidades das aplicações, agregando bibliotecas de autenticação, banco de dados, segurança, entre outras. Isso reduz o tempo de desenvolvimento e favorece a padronização.

Em conjunto, *Node.js* e *Express.js* se consolidaram como uma das principais escolhas para o desenvolvimento de aplicações modernas baseadas em arquitetura cliente-servidor. Sua combinação oferece desempenho, flexibilidade e simplicidade, sendo amplamente utilizada em projetos de diferentes portes, desde protótipos até sistemas de larga escala.

2.1.3.2 *Postman*

Ferramenta muito utilizada por desenvolvedores de *software* para testes, documentação e elaboração de *APIs*. É uma aplicação *desktop* que fornece uma interface simples e direta para realizar diversas tarefas relacionadas à construção de uma API.

Através do *Postman*, desenvolvedores podem enviar requisições *HTTP* para *APIs*, o que facilita a depuração e testes de um serviço *web* pronto ou em construção. Ele fornece suporte aos métodos mais conhecidos, como *GET*, *POST*, *PUT* e *DELETE*, permitindo que o profissional simule diferentes tipos de interação com a *API*.

Ademais, é uma ferramenta muito versátil para a documentação de *APIs*. Desenvolvedores conseguem facilmente criar coleções de requisições, adicionar descrições, salvar exemplos de testes e compartilhar de maneira prática essas informações com colegas de equipe ou com a comunidade de profissionais da área em geral.

Outra funcionalidade do *Postman* é a possibilidade de automatizar tarefas repetitivas através de *scripts*. Os profissionais podem elaborar diferentes *scripts* utilizando *JavaScript* propriamente para pré-processar requisições, extrair informações de respostas e realizar validações

automaticamente, tornando etapas de testes mais práticas, eficientes e confiáveis.

Por fim, a ferramenta *Postman* se torna essencial para desenvolvedores profissionais que trabalham com a elaboração, construção e/ou manutenção de *APIs*, pois oferece uma maneira simples e eficiente para testar, documentar e colaborar em serviços *web*. Com sua interface amistosa, recursos para automação e praticidade na colaboração, tornam esta ferramenta uma escolha comum entre equipes de desenvolvedores em diversos lugares.

2.1.3.3 *React*

Uma biblioteca *JavaScript* de código aberto (*open source*), criada pelo *Facebook* em 2013, com o objetivo de facilitar o desenvolvimento de interfaces de usuário interativas e dinâmicas. Desde seu lançamento, o *React* se consolidou como uma das ferramentas mais populares no ecossistema de desenvolvimento *web*, sendo amplamente adotado por empresas e desenvolvedores em todo o mundo.

Sua principal característica é a utilização do conceito de componentes reutilizáveis, que permitem estruturar aplicações em pequenas partes independentes. Essa abordagem torna o desenvolvimento mais modular, organizado e de fácil manutenção, além de favorecer a reutilização de código em diferentes partes da aplicação.

Outro diferencial importante do *React* é o uso do *Virtual DOM* (*Document Object Model virtual*). Esse mecanismo cria uma representação virtual da interface, atualizando apenas os elementos que realmente sofrem alterações. Dessa forma, o *React* reduz a quantidade de manipulações diretas no *DOM* real, resultando em um desempenho superior e em uma experiência de usuário mais fluida.

O *React* também adota uma abordagem declarativa, onde o desenvolvedor descreve o estado desejado da interface e a biblioteca se encarrega de atualizá-la de acordo com as mudanças nos dados. Esse paradigma facilita a compreensão do fluxo da aplicação e reduz a ocorrência de erros comuns em manipulações manuais da interface.

Além disso, o *React* possui um ecossistema rico e em constante expansão. Ferramentas complementares, como o *React Router* para gerenciamento de rotas e o *Redux* para controle de estados globais, ampliam suas capacidades e tornam a biblioteca adequada para projetos de diferentes portes, desde aplicações simples até sistemas de grande escala.

Por fim, a ampla comunidade ativa, a documentação detalhada e a adoção por grandes empresas tornam o *React* uma das escolhas mais seguras e eficientes para o desenvolvimento *Frontend* moderno. Sua flexibilidade, aliada à performance e ao suporte contínuo da comunidade, o consolidam como uma tecnologia essencial para a construção de interfaces ricas e responsivas.

2.1.3.4 *PostgreSQL*

Trata-se de um sistema de gerenciamento de banco de dados relacional (SGBDR) de código aberto, reconhecido pela sua robustez, escalabilidade e aderência aos padrões SQL. Ele foi desenvolvido inicialmente no final da década de 1980 e, desde então, evoluiu para se tornar uma das soluções mais utilizadas tanto no meio acadêmico quanto no mercado. Sua natureza *open source* permite que a comunidade contribua constantemente com melhorias, correções e novas funcionalidades.

Um dos diferenciais do *PostgreSQL* é o seu forte suporte à conformidade com os padrões SQL, aliado a extensões que oferecem funcionalidades avançadas, como suporte a dados *JSON*, *XML* e até mesmo dados geoespaciais por meio da extensão *PostGIS*. Essa flexibilidade possibilita que o sistema seja utilizado em diferentes cenários, desde aplicações tradicionais até soluções modernas que exigem manipulação de dados semiestruturados.

Além disso, o *PostgreSQL* é conhecido pela sua confiabilidade e integridade dos dados. Ele oferece recursos como controle de concorrência multiversão (*MVCC*), que garante transações consistentes mesmo em ambientes com múltiplos acessos simultâneos. Outros mecanismos importantes incluem suporte a transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e a implementação robusta de *backup* e recuperação, essenciais para ambientes críticos.

Outro ponto relevante é a extensibilidade do *PostgreSQL*. O sistema permite a criação de novos tipos de dados, operadores, funções e linguagens de programação, adaptando-se às necessidades específicas de cada aplicação. Essa característica o torna não apenas um banco de dados relacional, mas também uma plataforma versátil capaz de se moldar a diferentes domínios de aplicação.

Ele também se destaca em termos de desempenho e escalabilidade. É capaz de lidar com grandes volumes de dados e múltiplas conexões simultâneas sem perda significativa de eficiência. Além disso, conta com recursos como particionamento de tabelas, paralelização de consultas e otimização de índices, que contribuem para um processamento mais rápido e eficiente.

Por fim, a ampla adoção do *PostgreSQL* no mercado deve-se também à sua compatibilidade com diferentes sistemas operacionais, como *Linux*, *Windows* e *macOS*, bem como à sua integração com diversas linguagens de programação e *frameworks*. Essa versatilidade, somada à segurança, desempenho e constante evolução, o consolida como uma das principais opções para o desenvolvimento de sistemas modernos que demandam um gerenciamento eficiente e confiável de dados.

2.1.4 Tendências Atuais

É notável como o desenvolvimento de *software* tem evoluído de maneira acelerada, impulsionado pela necessidade de maior qualidade, rapidez e flexibilidade na entrega de soluções. Entre as principais tendências observadas atualmente estão a integração e entrega contínua (*CI/CD*), a cultura *DevOps*, a automação de testes, o uso da computação em nuvem e a incorporação de técnicas de Inteligência Artificial no processo de desenvolvimento.

A integração contínua (*Continuous Integration* ou *CI*) e a entrega contínua (*Continuous Delivery/Deployment* ou *CD*) tornaram-se práticas fundamentais para aumentar a agilidade no desenvolvimento. A integração contínua consiste em integrar o código desenvolvido por diferentes membros da equipe em um repositório central de forma frequente, garantindo que problemas sejam identificados rapidamente. Já a entrega contínua busca automatizar o processo de disponibilização do *software* em ambientes de homologação e produção, reduzindo o tempo entre a implementação e a entrega final ao cliente. Essa abordagem melhora a qualidade e reduz riscos, uma vez que as alterações são entregues em pequenos lotes, facilitando a detecção de falhas (FOWLER; FOORD, 2006).

Nesse mesmo contexto, surge a cultura *DevOps*, que busca aproximar as áreas de desenvolvimento (*Dev*) e operações (*Ops*), promovendo colaboração, automação e monitoramento contínuo em todo o ciclo de vida do *software*. O *DevOps* propõe quebrar silos organizacionais, incentivando a responsabilidade compartilhada pelo produto final. Isso resulta em maior velocidade de entrega, confiabilidade das aplicações e capacidade de resposta a mudanças e incidentes. Empresas que adotam *DevOps* relatam ganhos significativos em termos de escalabilidade e inovação (KIM et al., 2016).

Outro aspecto essencial das práticas modernas é a automação de testes. Com a crescente complexidade dos sistemas, os testes manuais isolados tornam-se insuficientes para garantir qualidade. A automação permite executar conjuntos de testes de maneira repetitiva, rápida e confiável, cobrindo desde testes unitários até testes de integração e regressão. Além de reduzir o tempo e custo da verificação, os testes automatizados aumentam a confiança no processo de entrega contínua, já que cada alteração pode ser validada em questão de minutos (FEWSTER; GRAHAM, 1999).

O avanço da computação em nuvem também representa um marco para o desenvolvimento de *software* contemporâneo. Plataformas como *Amazon Web Services* (*AWS*), *Microsoft Azure* e *Google Cloud* oferecem infraestrutura escalável, serviços gerenciados e ferramentas de integração que permitem às empresas reduzir custos e aumentar a eficiência. A nuvem facilita a adoção de arquiteturas modernas, como microsserviços e contêineres, além de apoiar práticas como *DevOps* e *CI/CD*. Com isso, organizações conseguem implantar soluções globalmente com maior rapidez e flexibilidade (ARMSTRONG; WILDER, 2019).

Por fim, observa-se o crescente uso da Inteligência Artificial (IA) como apoio ao desenvolvimento de *software*. Ferramentas baseadas em IA vêm sendo aplicadas para sugerir trechos

de código, detectar vulnerabilidades de segurança, prever falhas e até auxiliar no *design* de arquiteturas. Embora ainda seja um campo em evolução, a integração da IA no ciclo de vida do desenvolvimento aponta para ganhos de produtividade e qualidade. Contudo, devido à sua relevância e complexidade, esse tema será abordado em maior profundidade na seção seguinte deste trabalho.

2.2 IA Generativa aplicada ao desenvolvimento de *software*

Lorem ipsum.

3 TRABALHOS RELACIONADOS

Para fins de melhor localizar os trabalhos com temáticas relacionadas para o presente estudo, foi utilizado a plataforma de busca especializada em literatura acadêmica e científica Google Scholar. Os procedimentos de pesquisa e seleção consistiram em utilizar palavras chaves em português como “inteligência artificial generativa”, “desenvolvimento de software”, “produtividade”, “suporte”, “otimização” e “performance”, variando também os termos em inglês. Por fim, para selecionar os trabalhos a seguir, foi considerado o contexto do estudo de cada autor, conceitos abordados, tecnologias mencionadas e/ou utilizadas e problemáticas ressaltadas, que melhor correlacionassem com o presente estudo.

3.1 Trabalhos Acadêmicos

Dado o contexto do aprimoramento e evolução de Inteligências Artificiais Generativas nos anos recentes, tal como suas aplicabilidades para otimização de tarefas voltadas para a área de desenvolvimento de software, diversos estudos têm sido realizados com o intuito de explorar ainda mais os conceitos técnicos e práticos. A seguir, são apresentados estudos e pesquisas relacionados ao tema, estabelecendo um paralelo com a temática abordada para este trabalho.

Gomes (2023) apresenta uma análise comparativa de diferentes Inteligências Artificiais Generativas (IAs Generativas) aplicadas no desenvolvimento de software, mas particularmente focando em ferramentas auxiliares como *GitHub Copilot*, *Codeium*, *Tabnine* e *CodeGeeX*. Para fins práticos, seu trabalho focou em experimentos utilizando *GitHub Copilot* e *Codeium* para mensurar a eficiência das ferramentas.

Como resultado, o Gomes (2023) aponta que ambas as tecnologias se mostram extremamente úteis no processo de otimização do desenvolvimento, produtividade e qualidade de códigos, com a escolha entre elas dependendo das necessidades particulares do projeto e das preferências do profissional. Entretanto, o destaque ficou para a ferramenta *GitHub Copilot*, que apresentou uma leve vantagem, com uma capacidade superior de compreensão para construção de uma lógica de desenvolvimento mais coesa.

Em Silva et al. (2025), é realizado o teste prático de viabilidade de desenvolvimento de uma aplicação *web* para gerenciamento de sócios torcedores de um clube de futebol, utilizando IA Generativa como apoio na construção da aplicação. Ainda que, neste estudo os autores desenvolveram uma solução simples, utilizando tecnologias como *JavaScript*, *PHP*, *HTML5* e entre outras, o trabalho demonstrou que utilizar de Inteligência Artificial otimiza o processo de desenvolvimento, desde a geração de código, identificação de falhas e sugestões de melhores práticas para a construção de soluções.

De acordo com os autores e os resultados obtidos (SILVA et al., 2025), a adoção de IA Generativa no processo de desenvolvimento contribui tanto para acelerar a produção, quanto para a qualidade e otimização da aplicação. Para o estudo em questão, os autores contaram com o suporte de modelos de IA Generativa como *GPT-3.5* e *Claude 1*.

Em contrapartida, no trabalho de Ferreira (2023), o autor aborda sobre a influência das IAs Generativas na área de desenvolvimento de software, com base em um estudo qualitativo, combinando cenários exploratórios e descritivos. Aqui, o autor menciona o uso de ferramentas como o *ChatGPT*, *Bard*, *GitHub Copilot* para tratar sobre a temática de modelos baseados no conceito de Processamento de Linguagem Natural (PLN).

Ao longo de seu estudo, Ferreira (2023) aborda questões como o impacto da Inteligência Artificial na programação, trazendo dados e referências, para dissertar sobre tópicos como os desafios e benefícios do uso de ferramentas de IA. Sobre os desafios, o autor menciona a questão da dependência do uso de IA, limitando assim o desenvolvimento de habilidades próprias por parte do programador e a substituição do desenvolvedor por IA, assunto fortemente debatido desde a evolução das IAs Generativas nos anos recentes.

Santos (2024) investigou a influência das ferramentas de IA Generativas, citando *ChatGPT* e *GitHub Copilot*, com foco no processo de ensino e aprendizagem das instituições de ensino superior na área de desenvolvimento de *software*. O objetivo deste estudo consistiu em identificar oportunidades e desafios no uso destas tecnologias. Para isso, o autor realizou um comparativo com grupos de alunos iniciantes, estagiários no setor de tecnologia, alocando-os em um projeto simulado.

Para tanto o autor (SANTOS, 2024) seguiu um delineamento *crossover* 2x2 AB/BA, isto é, alternando os grupos entre o uso e não uso de IA. Após a coleta de dados, foi mensurado o conhecimento técnico dos grupos, tal como qualidade dos artefatos gerados, avaliação do produto desenvolvido e provas simuladas para testar o conhecimento adquirido. Aqui, o autor ressalta que os grupos que fizeram o uso de IA obtiveram melhores avaliações nos artefatos desenvolvidos, desde a documentação ao desenvolvimento de *software* propriamente dito. Entretanto, a nível de conhecimento, não refletiu em avaliações melhores nas provas simuladas (notas de avaliações, propriamente).

Costa (2024) explorou como o uso de IA Generativa pode otimizar e aprimorar o processo de desenvolvimento de *software* no contexto empresarial. O autor conduziu uma extensa revisão bibliográfica com o intuito de identificar as ferramentas e metodologias mais atuais e eficazes

no mercado. A partir de então, selecionou e implementou *frameworks* como *Spring Boot* para a construção de uma aplicação escalável em *Java*, utilizando o *ChatGPT* como ferramenta de suporte durante o processo.

Ademais, o autor (COSTA, 2024) fez o uso de ferramentas de teste e documentação de *APIs*, tais como *Postman* e até mesmo *JUnit*, como *framework* para validação de testes unitários. Em seu trabalho, o autor utilizou a IA Generativa através de *prompts* para criação de todos os requisitos, códigos e validações do seu projeto. Como resultado, ele evidenciou uma melhoria na eficiência tanto do desenvolvimento quanto da qualidade da aplicação entregue. Entretanto, ressaltou a importância de que, o conhecimento prévio para uso de IAs Generativas é fundamental, para que se torne possível o uso da mesma como ferramenta de apoio no processo de desenvolvimento de *software*.

Por fim, com base nos estudos analisados e considerando o crescimento acelerado das ferramentas de Inteligência Artificial Generativa na atualidade, evidencia-se que o profissional — independentemente da área de atuação — pode, e deve, tirar proveito dessas tecnologias para se manter em constante aprendizado e adaptado às novas realidades do mercado. Ou seja, não é necessário dominar todos os princípios técnicos por trás dessas ferramentas, mas sim saber utilizá-las estrategicamente, visando otimização, desempenho e melhores resultados.

Dessa forma, a proposta deste estudo é analisar, com base em dados e indicadores, o quanto o uso da Inteligência Artificial Generativa pode contribuir, junto aos profissionais da área de desenvolvimento de *software*, para a entrega de soluções mais ágeis. Para delimitar o foco principal do trabalho, o segmento analisado será o desenvolvimento de software, considerando o papel do desenvolvedor no cenário atual do mercado de trabalho.

Com base no levantamento de todas as informações dos trabalhos relacionados ao tema, é possível agrupar as principais características de cada um deles para fins de comparação. Dada as informações levantadas, separa-se as características em quatro parâmetros, sendo eles:

- **Tema principal:** define qual foi o principal tema abordado ao longo do estudo;
- **Metodologia aplicada:** define o modelo utilizado para o estudo em questão;
- **Tecnologias/Inteligências Artificiais utilizadas:** se aplicável, define as ferramentas e/ou tecnologias utilizadas para fins de estudo, tal como o modelo de Inteligência Artificial utilizado ou analisado;
- **Objetivo:** define o objetivo do modelo, ou seja, qual problemática o modelo buscou esclarecer ao seu decorrer.

A partir do resultado do levantamento de trabalhos relacionados, foi possível elaborar a Tabela 2, onde mostra o conteúdo dos critérios apontados em cada estudo, por ordem de apresentação.

Tabela 2 – Comparativo entre os trabalhos relacionados

Autores	**Tema**	**Metodologia**	**Tecnologias/IA**	**Objetivo**
Gomes (2023)	Comparação de IAs no desenvolvimento de *software*	Testes práticos comparativos	*GitHub Copilot*, *Codeium*, *Tabnine*, *CodeGeeX*	Avaliar eficiência de ferramentas de IA no desenvolvimento
Silva et al. (2025)	Aplicação de IA na criação de sistemas *web*	Estudo de caso com experimento	*JavaScript*, *PHP*, *HTML5*, *GPT-3.5*, *Claude 1*	Verificar ganho de produtividade com uso de IA
Ferreira (2023)	Impacto da IA na programação	Estudo qualitativo exploratório	*ChatGPT*, *Bard*, *GitHub Copilot*	Analisar benefícios e riscos do uso da IA
Santos (2024)	IA no ensino de tecnologia	Experimento com grupos (*crossover*)	*ChatGPT*, *GitHub Copilot*	Medir impacto da IA na aprendizagem e desempenho técnico
Costa (2024)	Uso de IA Generativa no desenvolvimento de *software* empresarial	Revisão bibliográfica com aplicação prática	*Spring Boot*, *Java*, *ChatGPT*, *Postman*, *JUnit*	Otimizar e melhorar eficiência e qualidade no desenvolvimento de aplicações

Fonte: Elaborado pelo autor

4 MATERIAIS E MÉTODOS

Lorem ipsum.

5 MODELO PROPOSTO

Lorem ipsum.

5.1 Visão Geral

Lorem ipsum.

5.2 Arquitetura e Tecnologias

Lorem ipsum.

5.3 Funcionalidades

Lorem ipsum.

5.4 Login e Cadastro

Lorem ipsum.

5.5 Menu e Tela Inicial

Lorem ipsum.

5.6 Navegação

Lorem ipsum.

6 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Lorem ipsum.

6.1 Coleta de Dados

Lorem ipsum.

6.2 Avaliação dos Resultados

Lorem ipsum.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Lorem ipsum.

Referências

ARMSTRONG, J.; WILDER, B. **The Cloud Adoption Playbook: Proven Strategies for Transforming Your Organization with the Cloud**. Hoboken: Wiley, 2019.

BECK, K. et al. **Manifesto for Agile Software Development**. 2001. Disponível em: <https://agilemanifesto.org/>. Acesso em: 29 ago. 2025.

BOEHM, B. W. A spiral model of software development and enhancement. **ACM SIGSOFT Software Engineering Notes**, ACM, v. 11, n. 4, p. 14–24, 1986.

CHAVES, L. C. et al. Construção de modelo para apoiar o processo de desenvolvimento de um sistema de apoio à decisão. **JISTEM - Journal of Information Systems and Technology Management**, São José, Tubarão, SC, v. 17, n. 1, 2020. ISSN 1807-1775. Disponível em: <https://dialnet.unirioja.es/servlet/articulo?codigo=8391447>. Acesso em: 28 ago. 2025.

COSTA, V. J. L. **ChatGPT: uma análise da ferramenta aplicada no processo de desenvolvimento de software**. Goiânia, GO: [s.n.], 2024. Pontifícia Universidade Católica de Goiás. Disponível em: <https://repositorio.pucgoias.edu.br/jspui/handle/123456789/7929>. Acesso em: 23 ago. 2025.

COUTINHO, E.; BEZERRA, C. Simulação de alocação de recursos em projetos de desenvolvimento de software utilizando teoria das filas. In: **Anais do III Workshop em Modelagem e Simulação de Sistemas Intensivos em Software**. Porto Alegre, RS: SBC, 2021. p. 30–39. ISSN 0000-0000. Disponível em: <https://sol.sbc.org.br/index.php/mssis/article/view/17257>. Acesso em: 24 ago. 2025.

CâNDIDO, A. K. R. **Uma revisão sistemática de estudos secundários sobre práticas ágeis de desenvolvimento de software**. Goiânia, GO: [s.n.], 2022. Pontifícia Universidade Católica de Goiás. Disponível em: <https://repositorio.pucgoias.edu.br/jspui/handle/123456789/4436>. Acesso em: 24 ago. 2025.

FERREIRA, I. S. **Benefícios e desafios do uso de IAs na programação**. Salgueiro, PE: [s.n.], 2023. TCC (Sistemas para Internet) – Instituto Federal de Educação, Ciência e Tecnologia do Sertão Pernambucano. Disponível em: <https://releia.ifsertao-pe.edu.br/jspui/handle/123456789/1184>. Acesso em: 21 jul. 2025.

FEWSTER, M.; GRAHAM, D. **Software Test Automation: Effective Use of Test Execution Tools**. Boston: Addison-Wesley, 1999.

FOWLER, M.; FOORD, M. **Continuous Integration: Improving Software Quality and Reducing Risk**. Boston: Addison-Wesley, 2006.

GOES, B. P. et al. Estudo de caso: metodologias ágeis scrum e asd aplicadas em empresas de desenvolvimento de software e projetos. **Tudo é Ciência: Congresso Brasileiro de Ciências e Saberes Multidisciplinares**, Volta Redonda, RJ, n. 1, p. 1–9, 2022. Disponível em: <https://conferencias.unifoa.edu.br/tc/article/view/13>. Acesso em: 28 ago. 2025.

GOMES, P. R. P. **Uma análise preliminar das IA generativa no suporte ao desenvolvimento de software**. Serra, ES: [s.n.], 2023. 61 p. Monografia (Bacharelado em Sistemas de Informação) – Instituto Federal do Espírito Santo. Disponível em: <https://repositorio.ifes.edu.br/handle/123456789/5914>. Acesso em: 21 jul. 2025.

KIM, G. et al. **The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations**. Portland: IT Revolution Press, 2016.

MAYNARD, A. D. Navigating the fourth industrial revolution. **Nature Nanotechnology**, Tempe, Arizona, EUA, v. 10, p. 1005–1006, 2015. Disponível em: <https://www.nature.com/articles/nnano.2015.286>. Acesso em: 24 ago. 2025.

MEDEIROS, A. YPEDUC: Uma adaptação de Metodologia Ágil para o Desenvolvimento de Software Educativo. **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)**, v. 30, n. 1, p. 379, 2019. ISSN 2316-6533. Disponível em: <http://milanesa.ime.usp.br/rbie/index.php/sbie/article/view/8742>. Acesso em: 28 ago. 2025.

NETO, J. G. **Metodologias ágeis em uma microempresa de desenvolvimento de softwares: um estudo de caso com o uso do Scrum**. Curitiba, PR: [s.n.], 2019. Universidade Tecnológica

Federal do Paraná. Disponível em: <http://repositorio.utfpr.edu.br/jspui/handle/1/19439>. Acesso em: 24 ago. 2025.

OSTERWEIL, L. J. **Software Processes are Software Too**. Berlin, Heidelberg, DE: Springer Berlin Heidelberg, 2011. 323–344 p. ISBN 978-3-642-19823-6. Disponível em: <https://doi.org/10.1007/978-3-642-19823-6_17>. Acesso em: 28 ago. 2025.

PRESSMAN, R. S. **Engenharia de Software: uma abordagem profissional**. 8. ed. Porto Alegre: AMGH, 2016.

SAKURAI, R.; ZUCHI, J. D. AS REVOLUÇÕES INDUSTRIAIS ATÉ A INDUSTRIA 4.0. **Revista Interface Tecnológica**, Taquaritinga, SP, v. 15, n. 2, p. 480–491, 2018. Disponível em: <https://revista.fatectq.edu.br/interfacetecnologica/article/view/386>. Acesso em: 24 ago. 2025.

SANTOS, G. P. dos. **Inteligência artificial generativa: o processo de ensino-aprendizagem no ensino superior de tecnologia**. Tese (Dissertação – Mestrado em Administração de Organizações) — Faculdade de Economia, Administração e Contabilidade de Ribeirão Preto, University of São Paulo, Ribeirão Preto, SP, 2024. Disponível em: <https://www.teses.usp.br/teses/disponiveis/96/96132/tde-23012025-124616/en.php>. Acesso em: 22 jul. 2025.

SCHWAB, K. **A Quarta Revolução Industrial**. [S.l.]: Edipro, 2019.

SILVA, I. H. da et al. DESENVOLVIMENTO DE APLICAÇÃO WEB COM INTELIGÊNCIA ARTIFICIAL GENERATIVA COMO AGENTE TUTOR SUPERVISOR NA CODIFICAÇÃO. **ARACÊ**, v. 7, n. 5, p. 22855–22877, 2025. Disponível em: <https://periodicos.newsciencepubl.com/arace/article/view/4942>. Acesso em: 21 jul. 2025.

SILVA, W. S. da. **Criação de um modelo de processo híbrido aplicado no desenvolvimento de um software baseado na integração das metodologias de desenvolvimento de software**. Santa Catarina, SC: [s.n.], 2020. Universidade do Extremo Sul Catarinense. Disponível em: <http://repositorio.unesc.net/handle/1/8849>. Acesso em: 28 ago. 2025.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Addison Wesley, 2011.

WIEGERS, K.; BEATTY, J. **Software Requirements**. 3. ed. [S.l.]: Microsoft Press, 2013.

APÊNDICE A – DIAGRAMA ER DO BANCO DE DADOS

Lorem ipsum.

APÊNDICE B – TELAS DA APLICAÇÃO

Lorem ipsum.

APÊNDICE C – TEXTO TEXTO

Lorem ipsum.

APÊNDICE D – CENÁRIOS DE TESTES

Lorem ipsum.

APÊNDICE E – TEXTO TEXTO

Lorem ipsum.

APÊNDICE F – TEXTO TEXTO

Lorem ipsum.

APÊNDICE G – RESPOSTAS DO QUESTIONÁRIO

Lorem ipsum.

APÊNDICE H – REQUISITOS FUNCIONAIS DA APLICAÇÃO

Lorem ipsum.

APÊNDICE I – TEXTO TEXTO

Lorem ipsum.

ANEXO A – NOME DO ANEXO

Lorem ipsum.