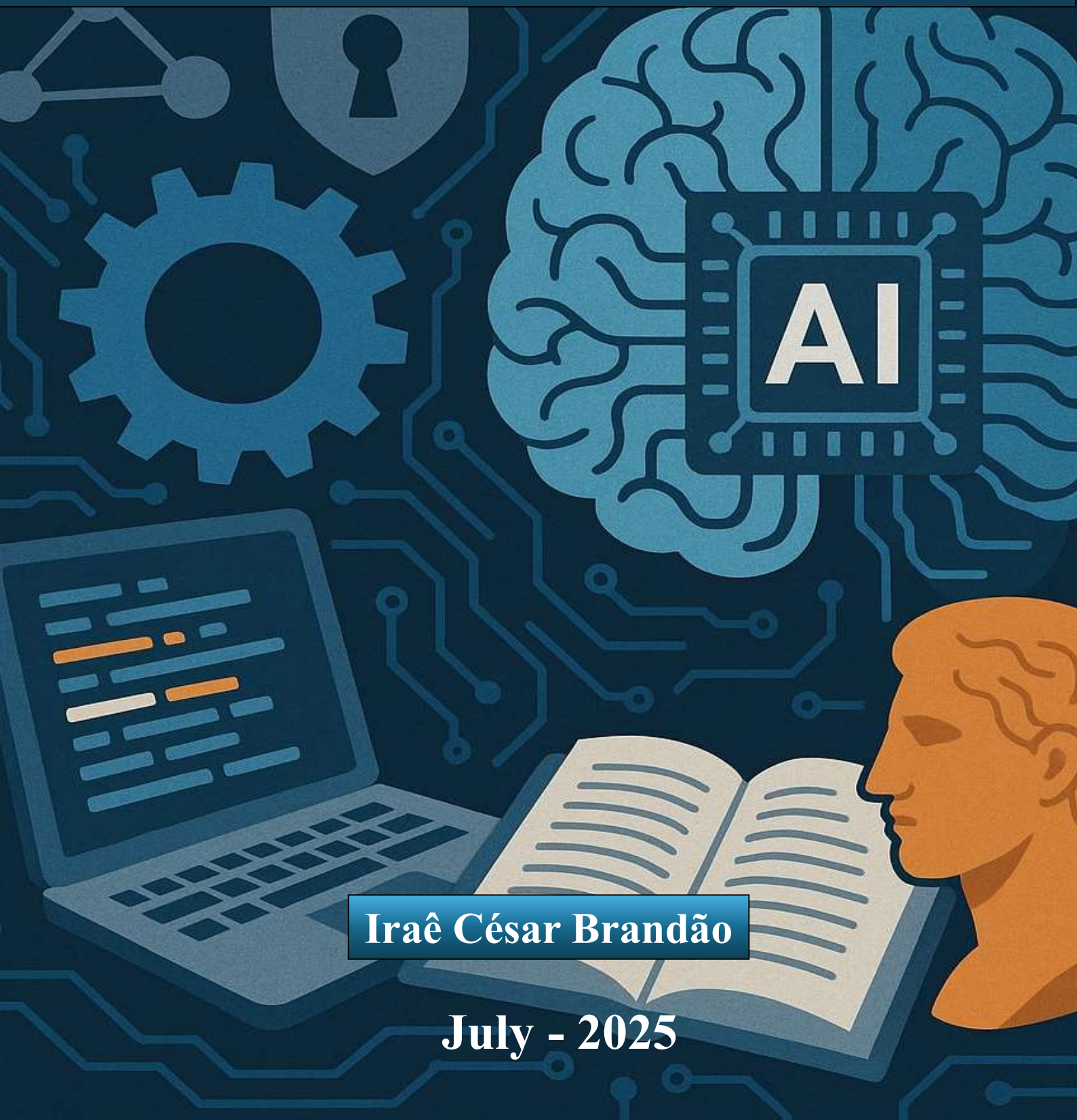
10.29327/7601326

DESAFIOS ATUAIS E A EVOLUÇÃO DA ENGENHARIA DE *SOFTWARE*:

Novas Exigências e a Dependência da Inteligência Artificial



Iraê César Brandão

July - 2025

Desafios Atuais e a Evolução da Engenharia de *Software*: Novas Exigências e a Dependência da Inteligência Artificial

Current Challenges and the Evolution of Software Engineering: New Demands and Dependence on Artificial Intelligence

Iraê César Brandão [i1]

[1]iraecbrandao@gmail.com.br

https://orcid.org/0000-0002-2079-0615

Abstract. *This essay examines the history, evolution, and emerging challenges faced by Software Engineering, from its genesis to the present day. Throughout the text, key historical milestones, influential authors, paradigm shifts, such as the Agile Manifesto and its recent impacts of Artificial Intelligence are discussed. The methodology employed was based on a literature review and critical analysis grounded in scientific articles, books, other specialized publications, and historical milestones. Furthermore, the essay addresses the growing superficiality of knowledge in the field, emphasizing the importance of recovering critical thinking and the urgent need to revisit the solid foundations of the discipline. The conclusion highlights that only through a rigorous approach, continuous study, and ethical responsibility will it be possible to build robust, efficient, and sustainable systems. The recovery of historical principles and the reintegration of a critical and reflective practice are essential for the advancement of Software Engineering. The solution does not lie in rejecting new technologies, but in using them with discernment and responsibility.*

Keywords: *Software Engineering; Artificial Intelligence; Critical Thinking Crisis; Agile Manifesto; Software crisis.*

Resumo. Este ensaio examina a história, a evolução e os desafios emergentes enfrentados pela Engenharia de *Software*, desde sua gênese até os dias atuais. Ao longo do texto, são discutidos marcos históricos relevantes, autores influentes, mudanças de paradigma, como o Manifesto Ágil e os impactos recentes da Inteligência Artificial. A metodologia utilizada se baseou em uma revisão de literatura e em uma análise crítica fundamentada em artigos científicos, livros, outras publicações especializadas e marcos históricos. Outrossim, o ensaio aborda a crescente superficialidade do conhecimento na área, enfatizando a importância de recuperar o pensamento crítico e a necessidade urgente de revisitar os fundamentos sólidos da disciplina. A conclusão ressalta que apenas por meio de uma abordagem rigorosa, estudo contínuo e responsabilidade ética será possível construir sistemas robustos, eficientes e sustentáveis. O resgate dos princípios históricos e a reintegração de uma prática crítica e reflexiva são essenciais para o avanço da Engenharia de *Software*. A solução não está em rejeitar as novas tecnologias, mas em utilizá-las com discernimento e responsabilidade.

Palavras-chave: Engenharia de *Software*; Inteligência Artificial; Crise de Pensamento Crítico; Manifesto Ágil; Crise do *Software*.

1. Introdução

A Engenharia de *Software*, uma disciplina relativamente jovem, emergiu formalmente como resposta à chamada *Crise do Software* nos anos 1960, quando se evidenciaram os desafios de se entregar sistemas de alta qualidade dentro dos prazos e orçamentos estipulados. A partir de então, a área tem se transformado de maneira substancial, passando por marcos históricos e revoluções tecnológicas, incluindo a popularização de novas linguagens de programação, o desenvolvimento de modelos de gestão e, mais recentemente, a introdução da Inteligência Artificial.

A evolução dessa disciplina é marcada por uma série de mudanças paradigmáticas que alteraram a forma como os sistemas são projetados, desenvolvidos e mantidos, com destaque para a adoção do Manifesto Ágil, que trouxe uma abordagem mais flexível e iterativa ao processo de desenvolvimento. No entanto, com as inovações que surgem constantemente, também aumentam os desafios, especialmente com o impacto da Inteligência Artificial e a crescente dependência de ferramentas automáticas que, muitas vezes, diluem o pensamento crítico e a compreensão profunda dos fundamentos.

Nos últimos anos, a Engenharia de *Software* tem enfrentado um paradoxo: enquanto novas tecnologias, como a IA e a automação, oferecem soluções rápidas e eficientes, elas também promovem uma superficialização do conhecimento entre os profissionais. Essa "cultura da velocidade", onde a produção de sistemas e *software* se torna cada vez mais acelerada, muitas vezes em detrimento da qualidade e da reflexão crítica, apresenta um desafio significativo para a sustentabilidade e ética da profissão.

A crise do pensamento crítico, exacerbada pela dependência excessiva de tecnologias automatizadas, tem gerado uma reflexão urgente sobre a necessidade de resgatar o entendimento profundo dos fundamentos da Engenharia de *Software* e a relevância de uma abordagem consciente e criteriosa na utilização de novas ferramentas e metodologias. Esse contexto abre um leque de questões a serem discutidas, como: De que maneira a crescente dependência de assistentes automatizados influencia a qualidade dos sistemas desenvolvidos e compromete o desenvolvimento do pensamento crítico e da autonomia profissional na Engenharia de *Software*?

O objetivo desta pesquisa é analisar a evolução da Engenharia de *Software*, seus marcos históricos e as implicações dos novos paradigmas emergentes, como a Inteligência Artificial e o Manifesto Ágil, na prática dos profissionais da área. Contudo, esta pesquisa está limitada a uma análise teórica e conceitual, com foco nos desenvolvimentos históricos e nas mudanças de paradigma, sem a exploração empírica ou de estudos de caso que possam ilustrar as consequências práticas dessas transformações no dia a dia dos engenheiros de *software*.

A investigação se concentra, portanto, na revisão da literatura relevante e nas discussões conceituais sobre os impactos dessas novas tecnologias e metodologias, buscando identificar os principais desafios que a área enfrenta e as possíveis soluções para preservação da qualidade e profundidade no desenvolvimento de sistemas.

2. Pressupostos Conceituais

2.1. Engenharia de *Software*

A Engenharia de *Software* surgiu como solução para a falta de métodos sistemáticos no desenvolvimento de programas. Atualmente, seus horizontes se expandem para a criação de

sistemas resilientes, seguros e éticos, integrando Inteligência Artificial, computação em nuvem[1], *blockchain*[2], e outras tecnologias emergentes.

Ela surgiu como uma resposta à *crise do software* na década 1960, quando os sistemas complexos começaram a ser desenvolvidos sem métodos sistemáticos, resultando em falhas, atrasos e orçamentos estourados. A disciplina foi estabelecida com o objetivo de aplicar princípios da engenharia tradicional à criação de *software*, promovendo a padronização, a documentação e a aplicação de práticas que garantissem a qualidade, confiabilidade e a entrega eficiente dos sistemas. Nesse contexto, a Engenharia de *Software* não é apenas um conjunto de técnicas para criar programas, mas uma abordagem para garantir que o *software* desenvolvido atenda às necessidades dos usuários e seja sustentável ao longo do tempo [The Chaos Manifest 2011].

Atualmente, ela se expandiu além de suas raízes iniciais, incorporando novas tecnologias e abordagens que exigem um equilíbrio entre inovação e rigor técnico. Entre os principais marcos que moldaram a sua evolução, podemos destacar a introdução de paradigmas como a programação estruturada nos anos 1970, a objetificação do código nos anos 1990 com a popularização de linguagens orientadas a objetos, e, mais recentemente, a adoção de metodologias ágeis, como o Manifesto Ágil em 2001 [Ambysoft 2022]. Estes marcos refletem mudanças significativas não só na forma como o *software* é desenvolvido, mas também na maneira como os profissionais da área abordam o processo de criação de sistemas.

2.2. Engenheiros de *Software* e Desenvolvedores

Os engenheiros de *software* são profissionais especializados na análise, projeto, implementação, teste e manutenção de sistemas de *software* complexos. Eles utilizam uma combinação de técnicas, metodologias e tecnologias para garantir que o mesmo seja construído de forma eficiente e confiável. Por outro lado, desenvolvedores, muitas vezes referidos como programadores, são os profissionais que escrevem o *código-fonte*[3], implementando as funcionalidades definidas pelos engenheiros de *software*. Embora as duas funções compartilhem algumas responsabilidades, os engenheiros tendem a ter um enfoque mais amplo e de longo prazo, considerando também os aspectos arquitetônicos e a sustentabilidade do sistema.

São profissionais essenciais no desenvolvimento de aplicações e sistemas digitais, trabalhando em diversas áreas como análise, projeto, desenvolvimento, testes e manutenção de *software.* Ambos desempenham papéis importantes, mas com focos e responsabilidades distintas, impulsionando a inovação tecnológica e transformando a maneira como interagimos com o mundo digital.

A imagem a seguir, Figura 1, foi elaborada para ilustrar a nossa atualidade e a sua evolução com o Manifesto Ágil e Inteligência Artificial, os desafios enfrentados.

---

[1] Computação em nuvem ou *cloud computing* – é a entrega de serviços de computação pela internet, como servidores, armazenamento, bancos de dados, rede, *software*, análise e inteligência, sob demanda, com base no consumo. Em vez de gerenciar a infraestrutura de TI local, as empresas podem acessar esses recursos como um serviço, pagando apenas pelo que usam.

[2] *Blockchain* (em português *cadeia de* blocos) – é uma tecnologia de registro distribuído e imutável que armazena dados em blocos interligados em uma cadeia. Esses blocos contêm informações, como transações financeiras ou dados digitais, e são validados por uma rede descentralizada de computadores. A principal característica do *blockchain* é sua segurança: uma vez registrada, a informação não pode ser alterada sem o conhecimento de todos os participantes da rede.

[3] Código-fonte – é o conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das linguagens de programação existentes, de maneira lógica. Existem linguagens que são compiladas e as que são interpretadas [LINFO 2004].

Figura 1: Engenheiro de *software* imerso em reflexão e estudo

2.3. Conceitos de Linguagens de Programação

As linguagens de programação constituem o principal meio de comunicação entre os desenvolvedores e os sistemas computacionais, permitindo a definição precisa de instruções que serão interpretadas ou executadas por máquinas. Elas podem ser classificadas em linguagens de baixo ou alto nível, conforme o grau de abstração em relação ao hardware. Linguagens de alto nível, como C, C++, *Java* e *Python,* são amplamente utilizadas no desenvolvimento de *software* por oferecerem maior legibilidade, portabilidade e recursos avançados [Cardelli 1996].

Além disso, as linguagens modernas frequentemente incorporam suporte a diversos paradigmas de programação (*e.g.,* como a programação orientada a objetos, a programação funcional e a programação concorrente), cada um adequado a diferentes contextos e demandas de projeto. A escolha apropriada da linguagem e do paradigma influencia diretamente na eficiência, na manutenibilidade e na escalabilidade das aplicações desenvolvidas.

2.4. Pandemia e seus efeitos na Engenharia de *Software*

A pandemia de COVID-19, que teve início em 2019, teve um impacto significativo na forma de como se desenvolver um *software*. Com a aceleração da transformação digital, muitas

empresas adotaram tecnologias emergentes, como computação em nuvem, inteligência artificial e automação, para continuar suas operações durante os períodos de isolamento. A mudança para o trabalho remoto (home office) também desafiou as metodologias tradicionais de desenvolvimento, forçando muitas equipes a adotarem práticas ágeis de maneira mais intensa. A pandemia ressaltou a necessidade de sistemas de *software* resilientes e escaláveis, capazes de suportar mudanças rápidas e imprevistas.

Diante desse cenário, a Engenharia de *Software*, que nasceu como uma solução para os desafios do desenvolvimento de *software*, continua a evoluir com o tempo. Seus horizontes estão cada vez mais ampliados, abrangendo tecnologias emergentes como a IA, computação em nuvem e *blockchain*[4]. A constante adaptação das metodologias de desenvolvimento, como o Ágil, permite que a área continue relevante, mesmo diante dos novos desafios. Contudo, os profissionais da Engenharia de *Software* devem sempre buscar o equilíbrio entre a adoção de novas tecnologias e a preservação dos princípios fundamentais da disciplina, garantindo que os sistemas criados sejam não apenas eficientes, mas também seguros, éticos e sustentáveis [Ambysoft 2022, Ambler & Lines 2012 e Manifesto Agile 2001].

3. Estudos Relacionados

3.1. História e Evolução da Engenharia de *Software*

A crise do *software* nos anos 1960 revelou a dificuldade em desenvolver sistemas complexos de forma confiável. Sommerville (2010 p. 5) destaca que "[...] a Engenharia de *Software* surgiu como uma resposta sistemática para produzir *software* de alta qualidade dentro de restrições de tempo e orçamento [...]". A conferência da *NATO*[5] em 1968, considerada o marco inicial da disciplina, enfatizou a necessidade de abordagens mais estruturadas [Pressman & Maxim 2016 p. 34].

3.2. Principais Marcos, *Softwares* e Linguagens

Segundo Sommerville (2010), a evolução das linguagens e modelos de desenvolvimento foram essenciais para consolidar a Engenharia de *Software* como ciência aplicada. Pressman & Maxim (2016) também destacam que as linguagens de alto nível como *FORTRAN*[6] e *COBOL*[7] foram fundamentais para a padronização dos processos de programação [ANSI X3.9 1978, Bisette 2024].

A evolução das linguagens de programação reflete mudanças paradigmáticas. *FORTRAN*, criado em 1957, é citado por Brooks (1975) como a "[...] primeira grande abstração

[4] *Blockchain* – é uma tecnologia de registro distribuído, uma espécie de livro-razão digital, onde as transações são agrupadas em blocos e conectados formando uma cadeia imutável. Este sistema descentralizado e transparente permite a validação e registro de informações por uma rede de computadores, sem a necessidade de um intermediário central [IBM 2025].

[5] *NATO* ou OTAN (Organização do Tratado do Atlântico Norte) – realiza diversas conferências ao longo do ano, que abordam temas variados, desde a segurança cibernética e espacial a questões de defesa e política nuclear. [IDN 2024].

[6] *Fortran* – acrônimo de *IBM Mathematical FORmula TRANslation System* – é uma linguagem de programação voltada para computação científica e técnica e que especialmente projetada pela execução eficiente em tempo de execução numa ampla variedade de processadores. Foi inicialmente desenvolvida na década de 1950. [ANSI X3.9 1978 e Bisette 2024].

[7] *COBOL* – *Common Business Oriented Language* – é uma linguagem de programação de alto nível, projetada para o processamento de dados em ambientes empresariais e financeiros. É conhecida por sua legibilidade e capacidade de lidar com grandes volumes de dados. Embora não seja a linguagem de programação mais moderna, ainda é amplamente utilizada em sistemas de mainframe, especialmente em grandes instituições financeiras e governos [Bisette 2024].

para programadores científicos [...]". *COBOL* 1959 focou na acessibilidade para negócios [Bisette 2024]. O surgimento da linguagem C na década de 1970 [Kernighan & Ritchie 1988] introduziu portabilidade e eficiência, enquanto C++ [Stroustrup 1985] acrescentou o paradigma de orientação a objetos.

Alguns dos principais marcos históricos são:

- A Crise do *Software* em 1968: Uma conferência internacional que destacou a crescente dificuldade de criar *software* de qualidade à medida que a complexidade dos sistemas aumentava, levando à formalização da Engenharia de *Software*.
- Desenvolvimento de Metodologias Estruturadas (década de 1970): A introdução da programação estruturada, que enfatizava o uso de fluxogramas e diagramas para mapear o comportamento do programa, ajudou a melhorar a organização do código.
- Programação Orientada a Objetos (década de 1980): A criação de linguagens como C++ e a filosofia de *design* orientada a objetos trouxeram uma nova maneira de modelar e organizar programas, facilitando a reutilização de código e a criação de sistemas mais escaláveis.
- O Manifesto Ágil (em 2001): Uma reação contra os processos de desenvolvimento rígidos e formais, promovendo uma abordagem mais flexível, com foco em entregas rápidas e interação constante com o cliente [Ambler & Lines 2012, Manifesto Agile 2001].

A famosa citação de Brooks (1975), "não existem balas de prata", reflete a ideia de que não há soluções rápidas ou fáceis para os desafios complexos, especialmente em áreas como o desenvolvimento de *software*. Em vez de buscar uma única resposta mágica, é fundamental adotar uma abordagem gradual e multifacetada para resolver problemas. Esse princípio pode ser aplicado a diversas áreas, como a educação, onde estratégias diversificadas são necessárias para atender às diferentes necessidades dos estudantes. A busca por uma solução simples, nesse contexto, é ilusória e desconsidera a complexidade do processo educacional.

A evolução dos paradigmas continuou com *Smalltalk*[8] e a Programação Orientada a Objetos – POO [Cardelli 1996], reforçada pela modelagem *UML*[9] na década de 1990 [Booch, Rumbaugh & Jacobson 1998]. Com o tempo, linguagens como *Java* e *Python* reforçaram a portabilidade e a produtividade, enquanto arquiteturas de *software*, no século XXI, a ascensão da Inteligência Artificial redefiniu o papel do engenheiro de *software*, exigindo novos conhecimentos em *Machine Learning*[10] e *Big Data*[11] [Bass, Clements & Kazman 2021].

3.2.1. Marcos históricos

Os marcos históricos da Engenharia de *Software* são eventos ou movimentos significativos que têm moldado e transformado os princípios e práticas dessa disciplina ao longo do tempo. Após a elaboração do Quadro 1, foram selecionados alguns desses marcos, cujo impacto foi

[8] *Smalltalk* – é uma linguagem de programação puramente orientada a objetos, considerada a primeira a implementar este paradigma de forma completa. Nele, tudo é tratado como um objeto, incluindo números, classes, métodos e blocos de código [Squeak 1996-2025].

[9] *UML* (*Unified Modeling Language*) – é uma linguagem de modelagem que utiliza diagramas para representar visualmente a arquitetura, o *design* e a implementação de sistemas de *software* [Booch, Rumbaugh & Jacobson 1998].

[10] *Machine learning* (sigla: *ML*) – é um ramo da inteligência artificial (IA) que se concentra na criação de sistemas que aprendem e melhoram com a experiência, sem serem explicitamente programados [Bass, Clements & Kazman 2021].

[11] *Big Data* – se refere a grandes volumes de dados complexos, muitas vezes não estruturados, que são difíceis de gerenciar e analisar com ferramentas tradicionais [Bass, Clements & Kazman 2021].

determinante para a formalização dos objetivos dentro de um conjunto fundamental de atividades: especificação, desenvolvimento, verificação e evolução de *softwares*:

Quadro 1: Principais Marcos da *Engenharia* de *Software*

Anos	Marcos históricos
1957	**Lançamento do *FORTRAN – First Formula Translator***
1959	**Lançamento do *COBOL – Common Business Oriented Language***
1970	**Desenvolvimento do Modelo em Cascata – *Winston Royce***
1980	**Popularização dos microcomputadores e surgimento do C++**
1990	**Modelagem orientada a objetos e criação da UML**
2001	**Publicação do Manifesto Ágil**
2020	**Popularização da Inteligência Artificial Generativa**

Fonte: Elaborado pelo autor.

Com o objetivo ilustrar de forma visual, elaboramos o infográfico conforme a Figura 2 a evolução da Engenharia de Software no período compreendido entre 1950 e 2025, que apresenta, de maneira cronológica e sintética, as principais eras, transformações e avanços tecnológicos que marcaram a área ao longo das décadas. A figura evidencia a transição desde os primeiros sistemas funcionais e centralizados até a era atual, caracterizada pelo uso intensivo de computação em nuvem, inteligência artificial, automação e sistemas distribuídos. Além disso, o infográfico destaca como a Engenharia de *Software* passou a depender cada vez mais dessas tecnologias emergentes, ampliando o papel do engenheiro de software, que hoje atua não apenas no desenvolvimento de código, mas também no projeto de soluções escaláveis, inteligentes e integradas a infraestruturas modernas.

Figura 2: Evolução da Engenharia de Software – 1950 a 2025

Fonte: Elaborado pelo autor.

3.3. O Manifesto Ágil e a Era Digital

Em 2001, o Manifesto Ágil foi criado por um grupo de 17 desenvolvedores, propôs uma mudança significativa na forma como o *software* deveria ser desenvolvido, enfatizando a colaboração, a flexibilidade e a entrega incremental de valor [Manifesto Agile 2001]. Ele estabeleceu quatro valores fundamentais: "[...] indivíduos e interações acima de processos e ferramentas, *software* funcionando acima de documentação extensiva, colaboração com o cliente acima de negociação de contratos e resposta a mudanças acima de seguir um plano [...]" [Beck *et al*. 2001], marcando uma ruptura com a rigidez dos modelos tradicionais como o *Waterfall*[12]. Esses princípios visavam proporcionar maior adaptabilidade durante o *ciclo de vida do desenvolvimento de software*[13], especialmente em ambientes dinâmicos e de mudanças rápidas.

As metodologias ágeis são conjuntos de práticas e técnicas que visam a gestão de projetos de forma mais flexível e adaptável às mudanças, com foco em entregas rápidas e iterativas. Dentre as metodologias que emergiram a partir desse manifesto, se destacam o *Kanban* e o *Scrum*[14], que focam na organização do trabalho por meio de sprints curtos e reuniões diárias de acompanhamento [Anderson 2010], e o *XP– Extreme Programming*[15], que enfatiza a programação em par e a prática de testes contínuos. Ambas são formas de implementar os princípios do Ágil, proporcionando flexibilidade e qualidade no desenvolvimento de *software* [Beck *et al*. 2001, Schwaber & Beedle 2001, Prada 2022]. Pressman & Maxim (2016 p. 83) ressaltam que o Ágil atendeu à necessidade da "entrega contínua de valor ao cliente em ciclos curtos", impulsionada pela popularização da Internet e da transformação digital.

O impacto do Manifesto Ágil foi transformador. De acordo com Highsmith (2002), um dos principais defensores das metodologias ágeis, essa abordagem proporcionou um foco maior no valor entregue ao cliente, permitindo um processo de desenvolvimento mais colaborativo e menos burocrático. Além disso, as metodologias ágeis propõem a construção de *software* de forma iterativa, com entregas frequentes e revisões contínuas, o que possibilita ajustar o curso do projeto conforme necessário, em vez de seguir um plano rígido. Portanto, o Manifesto Ágil não apenas transformou o processo de desenvolvimento de *software,* mas também ajudou a redefinir a relação entre desenvolvedores e clientes, priorizando a colaboração e a entrega contínua de valor ao invés da rigidez de abordagens anteriores [Ambysoft 2022, Ambler & Lines 2012].

O *Disciplined Agile Delivery* (DAD) foi introduzido por Ambler & Lines (2012), uma estrutura que busca harmonizar a flexibilidade e a adaptabilidade do desenvolvimento ágil com disciplinas necessárias para projetos corporativos complexos. DAD reconhece as realidades de portfólios de iniciativas interdependentes, oferecendo uma abordagem mais estruturada sem comprometer os princípios ágeis essenciais. Essa adaptação, conforme *ibidem*, é essencial para

---

[12] *Waterfall* – Metodologia Cascata – é uma abordagem sequencial e linear para gestão de projetos, onde cada fase deve ser completamente concluída antes de iniciar a próxima. É como um fluxo de água numa cachoeira, onde as etapas seguem um fluxo descendente, sendo que cada etapa depende da conclusão da anterior.

[13] O ciclo de vida de um *software* – se refere às etapas que um *software* passa, desde a sua concepção até o seu desuso, incluindo desenvolvimento, operação e manutenção. Este ciclo abrange desde a análise de requisitos e o *design*, passando pelo desenvolvimento, testes, implantação, manutenção e, finalmente, a descontinuação.

[14] *Kanban* e *Scrum* – são metodologias ágeis populares, mas com abordagens distintas para o gerenciamento de projetos. O *Kanban* é um sistema visual que foca na otimização do fluxo de trabalho e na limitação do trabalho em progresso (*WIP – Work In Progress*). Já o *Scrum* é um *framework* iterativo e incremental, com ciclos definidos (*sprints*) e papéis específicos, como *Scrum Master* e *Product Owner*.

[15] *XP* (sigla de *Extreme Programming*) – é uma metodologia ágil de desenvolvimento de *software* que visa entregar *software* de alta qualidade de forma rápida e adaptável, com foco na comunicação, feedback, e entregas frequentes de pequenas funcionalidades. Criado por Kent Beck na década de 1990, o XP se adapta a mudanças nos requisitos do projeto e prioriza a qualidade do produto final [Beck *et al.* 2001.

empresas que buscam vantagem competitiva através da agilidade, garantindo que as práticas ágeis sejam aplicadas de maneira disciplinada e eficaz em contextos diversificados. Essa integração entre flexibilidade e disciplina é relevante para o sucesso em um ambiente digital em constante transformação.

3.4. Evolução dos Serviços em Computação em Nuvem e sua Relação com a Engenharia de *Software* Moderna

A Computação em Nuvem[16] (*cloud computing*) se consolidou como um dos principais paradigmas tecnológicos da Engenharia de *Software* atualmente, ao possibilitar o acesso ubíquo, escalável e sob demanda a recursos computacionais compartilhados. Segundo Mell & Grance (2011), a computação em nuvem se caracteriza por um modelo que permite o provisionamento rápido de recursos como redes, servidores, armazenamento e aplicações, com mínimo esforço de gerenciamento por parte do usuário. Essa abordagem promoveu uma mudança significativa na forma como sistemas de software são projetados, desenvolvidos e mantidos..

A construção do Quadro 2, o qual apresenta de forma sistematizada a evolução dos serviços em computação em nuvem e seus principais marcos tecnológicos. Esse quadro tem como objetivo estabelecer uma correlação direta entre o avanço das plataformas *cloud* e as novas iniciativas da Engenharia de *Software*, evidenciando como a adoção de serviços em nuvem e a integração com IA passaram a ser elementos essenciais para o desenvolvimento de soluções modernas, escaláveis e inteligentes.

Quadro 2: Evolução da Computação em Nuvem (*Cloud Computing*)

Período	Fase da Nuvem	Principais Características	Marcos Relevantes
1960–1980	**Conceito Inicial**	**Compartilhamento de recursos computacionais**	***Mainframes* e *time-sharing***
1990–2000	***Pré-Cloud***	**Virtualização inicial e redes distribuídas**	**Surgimento da Internet comercial**
2000–2005	**Infraestrutura *Online***	**Data centers remotos e serviços hospedados**	***Web services* e hospedagem *web***
2006–2010	***Cloud* Pública**	**Infraestrutura sob demanda (IaaS)**	**Lançamento da AWS (2006)**
2010–2015	***Cloud* como Serviço**	**Plataformas e *softwares* como serviço (PaaS e SaaS)**	***Azure* e *Google Cloud***
2015–2020	***Cloud* Nativa**	**Microserviços, *containers* e *DevOps***	***Docker* e *Kubernetes***
2020–2025	***Cloud* Inteligente**	**Integração com IA, automação e *Edge Computing***	**IA generativa e computação híbrida**

Fonte: Elaborado pelo autor.

Com a adoção dos modelos de serviço Infraestrutura como Serviço (IaaS), Plataforma como Serviço (PaaS) e *Software* como Serviço (SaaS), a Engenharia de *Software* passou a dissociar o desenvolvimento de aplicações da gestão direta da infraestrutura física, favorecendo maior produtividade, reuso e padronização [Armbrust *et al*. 2010]. Essa transformação impactou diretamente o ciclo de vida do software, tornando-o mais dinâmico e alinhado às necessidades de escalabilidade e alta disponibilidade exigidas pelos sistemas modernos.

[16] Computação em Nuvem (*cloud computing*) - é a entrega de serviços de TI (*e.g.,* como servidores, armazenamento, bancos de dados, software e análises) pela internet, permitindo que empresas e usuários acessem esses recursos sob demanda, pagando apenas pelo uso, sem precisar comprar e manter infraestrutura física própria. Ela oferece flexibilidade, escalabilidade, economia de custos e inovação mais rápida, com acesso de qualquer lugar [Mell & Grance 2011].

Outrossim, a evolução da computação em nuvem impulsionou o surgimento de arquiteturas *cloud-native*, microserviços e práticas *DevOps*, as quais demandam ambientes altamente automatizados e distribuídos. De acordo com Sommerville (2010), tais abordagens ampliam a complexidade dos sistemas de software, exigindo do engenheiro de *software* competências que extrapolam a codificação tradicional, incluindo o projeto de arquiteturas resilientes, a integração contínua e o gerenciamento de ambientes distribuídos.

Nesse cenário, a integração da IA aos sistemas de *software* intensificou ainda mais a dependência da nuvem. Conforme destacado por Pressman & Maxim (2016), aplicações baseadas em aprendizado de máquina e IA generativa requerem grande capacidade computacional, armazenamento massivo de dados e infraestrutura escalável, elementos que são viabilizados predominantemente por plataformas de computação em nuvem. Assim, a nuvem se tornou o principal suporte para o desenvolvimento, treinamento, implantação e evolução de sistemas inteligentes.

3.5. Computação em Nuvem, Inteligência Artificial e a Dependência da Engenharia de Software

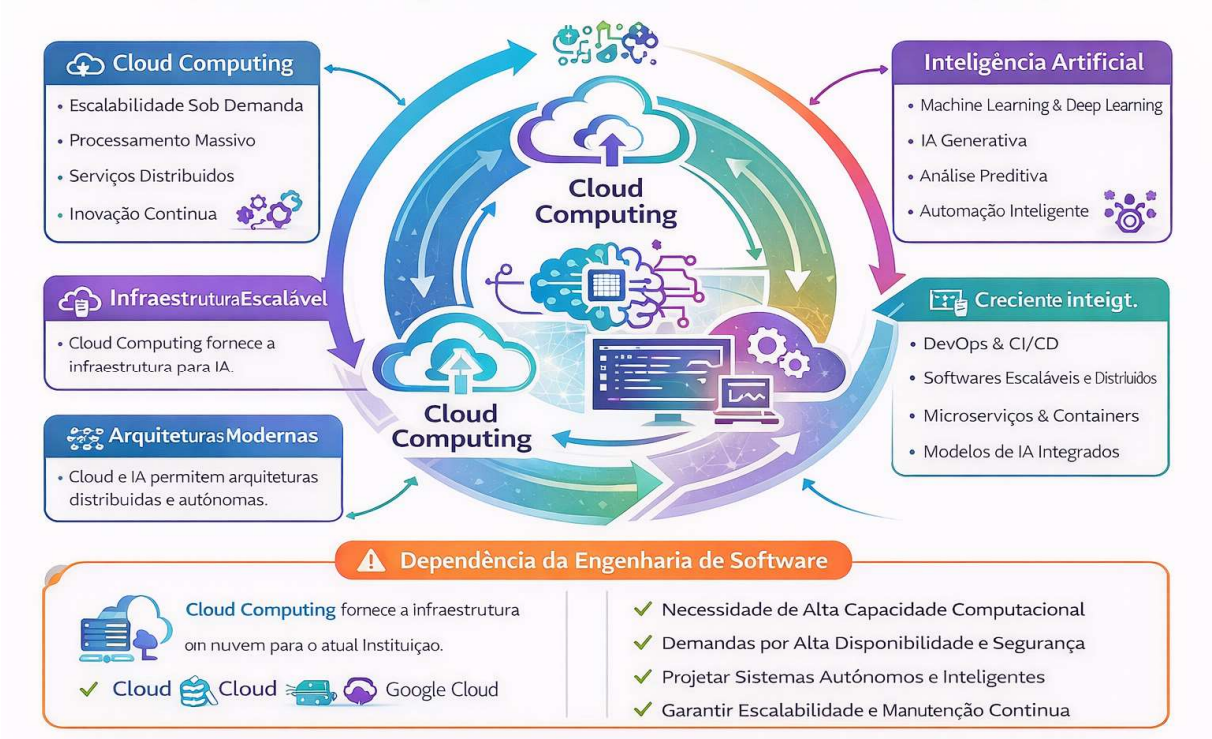
A Computação em Nuvem (*Cloud Computing*) e a Inteligência Artificial (IA) constituem, na atualidade, dois dos principais eixos tecnológicos que sustentam a evolução da Engenharia de *Software*. Conforme já abordado no tópico anterior, a consolidação desses paradigmas transformou profundamente a forma como sistemas são concebidos, desenvolvidos, implantados e mantidos, deslocando o foco do desenvolvimento puramente local para ambientes distribuídos, escaláveis e fortemente automatizados.

De acordo com Mell & Grance (2011), a computação em nuvem permite o acesso sob demanda a um conjunto compartilhado de recursos computacionais configuráveis, como redes, servidores, armazenamento e aplicações. Essa característica tornou a nuvem a infraestrutura predominante para sistemas modernos, viabilizando arquiteturas elásticas, alta disponibilidade e redução significativa de custos operacionais. Nesse contexto, a Engenharia de *Software* passou a depender diretamente dos serviços em nuvem para garantir desempenho, escalabilidade e continuidade operacional dos sistemas.

Paralelamente, o avanço da Inteligência Artificial intensificou essa dependência, uma vez que soluções baseadas em aprendizado de máquina, aprendizado profundo e IA generativa exigem grande capacidade computacional e processamento de dados em larga escala. Conforme destacado por Pressman & Maxim (2016), a incorporação de IA aos sistemas de software amplia a complexidade do processo de desenvolvimento, demandando ambientes capazes de suportar treinamento de modelos, inferência em tempo real e integração contínua, aspectos que são viabilizados majoritariamente por plataformas de computação em nuvem.

Dessa forma, elaboramos a Figura 3 com objetivo de ilustrar que a Engenharia de *Software* moderna se tornou intrinsecamente dependente da computação em nuvem e da IA, não apenas como ferramentas auxiliares, mas como componentes centrais para o desenvolvimento de sistemas modernos, escaláveis e inteligentes. Essa dependência tecnológica estabelece o cenário para novas discussões acerca do impacto dessas tecnologias na formação profissional, nos processos cognitivos e na tomada de decisão, temas que serão aprofundados no próximo tópico.

Figura 3: Dependência da Engenharia de Software em IA e Cloud Computing



Fonte: Elaborado pelo autor.

3.6. A Nova Geração e o Colapso de Pensamento Crítico

A pandemia de COVID-19 e o colapso da saúde pública em 2020 exigiram uma digitalização abrupta de serviços. A pandemia acelerou a digitalização dos serviços de saúde e as dificuldades enfrentadas pelo sistema de saúde devido ao aumento da demanda e à pressão inesperada [Keesara & Scnulman 2020]. De acordo com Kitchin (2014), "[...] a necessidade por soluções rápidas provocou uma explosão de ferramentas *Low-code*[17] e automação de *software* assistida por IA [...]". Tecnologias como *GitHub Copilot* e *GPT-3* passaram a gerar trechos inteiros de código, diminuindo o esforço técnico individual [Brown *et al.* 2020].

Conforme publicação da *World Health Organization* [WHO 2020], que fornece uma análise abrangente sobre as implicações da pandemia para os sistemas de saúde ao redor do mundo, se constatou que a crise sanitária expôs fragilidades estruturais e a limitada capacidade de resposta de muitos sistemas públicos. O documento destaca que a pandemia implicou a necessidade urgente de fortalecimento da atenção primária à saúde, da ampliação da infraestrutura hospitalar e da adoção de novas tecnologias para monitoramento e resposta a emergências sanitárias.

O desenvolvimento de *software* e a segurança ainda caminham em tensão constante, como discutido na palestra "*Desenvolvimento versus Segurança?*" no 26º GTS do CGI.br. Apesar dos avanços na área, a camada de aplicação segue sendo um dos principais alvos de incidentes. A dificuldade em integrar segurança desde o início do desenvolvimento revela problemas estruturais, de incentivos e comunicação entre as equipes. Superar essa divisão é essencial para fortalecer a tecnologia de forma segura e sustentável [Ferreira 2016].

[17] *Low-code* – é uma abordagem de desenvolvimento de *software* que reduz a necessidade de codificação manual, permitindo que desenvolvedores e até mesmo usuários com menos conhecimento técnico criem aplicações de forma mais rápida e eficientes [SAP 2025].

No entanto, segundo Susskind (2020), "a IA aprendeu predominantemente com dados medíocres disponíveis na internet, reproduzindo erros e práticas inadequadas". Assim, muitos desenvolvedores atualmente adotam o que pode ser chamado de *Vibe Coding*[18], uma prática de codificação superficial sem a internalização dos princípios fundamentais da Engenharia de *Software*.

3.7. *Vibe Coding* e a Criação de *Softwares*

O conceito de *Vibe Coding,* popularizado por Karpathy (2023), propõe uma nova abordagem para o desenvolvimento de *software*, em que o programador atua como um "diretor" e a IA "executora" das instruções em linguagem natural. Essa colaboração fluida entre IA e humanos permite criar códigos complexos em um tempo significativamente menor, otimizando o *time-to-market*[19] e proporcionando ganhos de produtividade no processo de desenvolvimento.

Embora torne a criação mais rápida e acessível, o desenvolvedor ainda precisa ter conhecimentos de programação para interpretar, validar e corrigir os códigos gerados. Não basta apenas copiar e colar: é essencial entender o que foi produzido para garantir qualidade, segurança e aderência às boas práticas [Karpathy 2023].

O desenvolvimento de *software*, embora promissor, está sujeito a riscos significativos quando códigos são elaborados de forma inadequada ou contêm *bugs*. Conforme aponta Sommerville (2010), erros de codificação não apenas impactam a funcionalidade dos sistemas, mas também elevam os custos e os riscos de falhas críticas. Desenvolvedores sem o conhecimento técnico necessário (*know-how*) para interpretar e corrigir essas falhas podem agravar ainda mais o problema, comprometendo a confiabilidade do produto final.

Segundo Osmani (2025), a IA vem transformando o desenvolvimento de *software*, levando os programadores a atuarem não apenas como escritores de código, mas como colaboradores em um fluxo de trabalho orientado por intenções. Essa nova prática, chamada de *Vibe Coding,* redefine a programação ao integrar assistentes de IA como *GitHub Copilot*[20] e *OpenAI Codex*[21], que não apenas automatizam tarefas rotineiras, mas também influenciam decisões de arquitetura e *design* de sistemas. Para Osmani, o futuro da programação está cada vez mais focado na colaboração criativa entre humanos e máquinas, otimizando tempo e ampliando as possibilidades de inovação.

Apesar de seu potencial revolucionário, o *Vibe Coding* apresenta riscos que não podem ser negligenciados. Estudos, como os de Pearce *et al.* (2022), alertam que códigos gerados por IA podem conter vulnerabilidades ocultas, não seguir boas práticas de segurança ou apresentar inconsistências difíceis de detectar. Em projetos complexos, a ausência de controle detalhado pode tornar a depuração mais desafiadora, exigindo atenção redobrada dos desenvolvedores.

Conforme Bacci (2025), os riscos e limitações do *Vibe Coding* são:

---

[18] *Vibe coding* – é uma nova abordagem para desenvolver *software* onde a IA gera código baseado em descrições em linguagem natural. Em vez de escrever código manualmente, o desenvolvedor interage com a IA fornecendo instruções sobre o que o *software* deve fazer, e a IA gera o código automaticamente [Karpathy 2023].

[19] *Time-to-market* (sigla *TTM* ou *tempo para o mercado* em português) – se refere ao período que decorre desde a concepção de um produto ou serviço até o seu lançamento e disponibilização para os consumidores.

[20] *GitHub Copilot* – é um assistente de programação com inteligência artificial (IA) desenvolvido pelo *GitHub* em colaboração com a *OpenAI*. Ele funciona como um par de programação, oferecendo sugestões de código e preenchimento automático dentro de ambientes de desenvolvimento integrado (IDEs), como *Visual Studio Code*, *Visual Studio* e outros. O *Copilot* analisa o código em tempo real, prevê o que o desenvolvedor está escrevendo e oferece sugestões contextuais para acelerar o processo de codificação [https://docs.github.com/pt/copilot].

[21] *OpenAI Codex* – é um modelo de inteligência artificial desenvolvido pela *OpenAI* que traduz linguagem natural para código de programação. Basicamente, ele funciona como um assistente de codificação que ajuda desenvolvedores a escrever, entender e manipular código [Código *OpenAI* - https://openai.com/codex/].

> "[...] O código gerado pode conter falhas, vulnerabilidades ou não seguir boas práticas; Em projetos complexos, é preciso dividir bem as etapas para garantir consistência; Nem sempre é fácil depurar erros, especialmente se você não entende o que foi gerado; Há risco de aceitar sugestões da IA sem entender seus impactos. Por isso, é essencial manter uma postura crítica e revisar tudo o que for implementado. A recomendação é sempre documentar as interações e validar os resultados com testes"

Assim, manter uma postura crítica diante das sugestões da IA é fundamental para garantir a qualidade e a segurança dos sistemas desenvolvidos. Segundo Amershi *et al*. (2019), é imprescindível que todo o código gerado seja revisado, documentado e validado através de testes rigorosos antes da implementação. A responsabilidade final permanece com o humano, que deve entender o impacto de cada linha criada para assegurar a confiabilidade do software.

3.8. Sustentabilidade e Responsabilidade Social na Engenharia de Software

O conceito de "Engenharia de *Software*" foi formalmente introduzido em 1968, durante a conferência "*NATO Software Engineering Conference*", promovida pela Organização do Tratado do Atlântico Norte – OTAN, em *Garmisch*, na Alemanha. Segundo os autores Pressman & Maxim (2016 p. 7), a expressão foi cunhada por Fritz Bauer, que definiu Engenharia de *Software* como "o estabelecimento e uso de princípios sólidos de engenharia para obter s*oftware* confiável e eficiente, que opere em máquinas reais, de forma econômica", caracterizada pela crescente incapacidade de entregar sistemas dentro do prazo, do orçamento e da qualidade esperados, resultando em projetos com alta taxa de falhas, custos elevados e insatisfação dos usuários.

Conforme destaca Booch (2019, p. 8), embora Bauer tenha desempenhado um papel significativo na consolidação e difusão do termo engenharia de software no contexto acadêmico e industrial internacional, a autoria pioneira da expressão é atribuída a Margaret Hamilton, cientista da computação e engenheira de sistemas norte-americana. Ela é amplamente reconhecida por ter cunhado e utilizado o termo *software engineering* entre os anos de 1963 e 1964, período em que liderava a equipe responsável pelo desenvolvimento do software de orientação e navegação a bordo do programa espacial *Apollo,* da *National Aeronautics and Space Administration* (NASA). A adoção do termo teve como objetivo conferir legitimidade científica e técnica à área, sustentando a argumentação de que o desenvolvimento de *software* deveria ser tratado com o mesmo rigor metodológico, responsabilidade e criticidade atribuídos às demais engenharias.

Conforme Sommerville (2010), a crise do *software* refletia a necessidade urgente de novas abordagens para o desenvolvimento de sistemas complexos, uma vez que os métodos tradicionais se mostravam insuficientes diante da crescente complexidade dos projetos. Assim, a Engenharia de *Software* emergiu como uma disciplina para estruturar práticas de desenvolvimento, utilizando metodologias, ferramentas e processos que assegurassem a previsibilidade e a qualidade do produto final. Desde então, a área evoluiu incorporando abordagens como processos iterativos, metodologias ágeis e práticas de garantia de qualidade, se consolidando como um campo essencial para o avanço da tecnologia da informação.

Anos atrás, a discussão sobre se o desenvolvimento de *software* era mais arte do que engenharia tomou força, principalmente pelo fato de a criatividade dos programadores desempenhar um papel essencial no processo. Esse mito, no entanto, foi desconstruído com base em uma análise aprofundada de sistemas convencionais, evidenciando que a Engenharia de *Software* é fundamental para a criação de soluções robustas e eficientes. Como Kalinowski *et al.* (2023) afirmam, embora a criatividade seja um aspecto valioso, ela deve ser equilibrada com rigor técnico, especialmente quando se trata de sistemas de *software* que precisam atender a requisitos críticos, e que:

> "[...] envolve modelos construídos por profissionais de outras áreas que não a computação em si [e] ao o comparar *software* com arte, precisamos pensar nas consequências de um *software* sofrer defeitos que resultem em falhas quando submetido à operação [e que] dessa forma, *softwares* que apresentam falhas e operações podem literalmente sistematizar o caos [e ainda] isso não seja claro para a sociedade como um todo, as consequências de falhas podem ser ainda mais drásticas no caso de sistemas de *softwares* inteligentes [...]" [Kalinowski *et al.* 2023 p. 11-12].

Conforme Bruno Gianella em seu artigo publicado na Medium, onde trata a transição da engenharia de *software* para engenheiro de IA, onde afirma que a transição:

> "[...] é facilitada por um conjunto de habilidades base que são altamente transferíveis, incluindo programação, solução de problemas e compreensão de algoritmos complexos. No entanto, para navegar com sucesso no campo da IA, é crucial desenvolver uma compreensão profunda de aprendizado de máquina, estatísticas e processamento de linguagem natural. É igualmente importante adquirir experiência prática com *frameworks*[22] de IA [e] entender como manipular grandes conjuntos de dados" [Gianella 2024].

Segundo alerta da própria *OpenAI* em sua interface: "o ChatGPT pode cometer erros. Considere verificar informações importantes" (*OpenAI* 2023 - tradução nossa), se tratando de um alerta que reafirma a limitação inerente dessas ferramentas ao lidar com dados, contextos complexos e validação científica.

Marcus & Davis (2020).reforça que a IA, embora sofisticada na geração de texto, não compreende contexto nem garante precisão científica, o que exige revisão humana rigorosa. Autores como Kreps & Kriner (2023) destacam que a IA pode reproduzir vieses sociais e epistemológicos presentes em seus dados de treinamento.

Bender *et al*. (2021) chamam atenção para os riscos de confiar em modelos sem transparência sobre suas fontes e processos de inferência. Portanto, mais do que uma questão técnica, a revisão crítica é um compromisso ético do pesquisador com a qualidade, a veracidade e a integridade de sua produção [Bender *et al*. 2021].

Com a crescente demanda por sistemas digitais, surge a necessidade de práticas sustentáveis no desenvolvimento de *software*. A Engenharia de *Software* Sustentável propõe a criação de soluções que minimizem impactos ambientais, como o consumo excessivo de energia e a geração de resíduos eletrônicos . Além disso, a responsabilidade social no setor passa a incluir preocupações com acessibilidade, privacidade de dados e inclusão digital [Penzenstadler *et al*. 2012], reforçando o papel ético desse profissional na construção de um futuro mais justo e equilibrado.

4. Metodologia Aplicada

Para a consecução dos objetivos deste ensaio, foi realizada uma abordagem baseada exclusivamente em pesquisa bibliográfica, fundamentada em artigos científicos, livros e publicações relevantes sobre a Engenharia de *Software* e sua evolução histórica. A coleta de dados utilizou bases reconhecidas como Google Acadêmico, *Scopus* e *Web of Science*, com descritores como "engenharia de *software*", "história da engenharia de *software*", "evolução do desenvolvimento de sistemas" e "práticas contemporâneas em desenvolvimento de *software*", "*Cloud computing*" e "IA. O recorte temporal priorizou publicações dos últimos 30 anos, sem desconsiderar obras clássicas que marcaram o surgimento e a consolidação da área.

[22] *Framework* em desenvolvimento de *software* é um conjunto de componentes de código reutilizáveis e estruturas que fornecem uma base para a construção de aplicações. Pense nele como uma estrutura pré-definida que agiliza o desenvolvimento, permitindo que os desenvolvedores se concentrem em funcionalidades específicas em vez de reinventar a roda [TechTarget 2024].

Durante a revisão, se constatou que a Engenharia de *Software* surgiu formalmente no final da década de 1960, como resposta à chamada "crise do *software*", e desde então vem se transformando continuamente para atender às crescentes demandas de qualidade, segurança e escalabilidade. As metodologias tradicionais deram lugar, em muitos contextos, a abordagens ágeis e adaptativas, refletindo a busca constante por eficiência no desenvolvimento. A pesquisa destacou que, apesar dos avanços teóricos e práticos, muitos dos desafios iniciais como a gestão de projetos, a garantia de qualidade e a integração de segurança. permanecem como temas centrais atualmente.

A seleção das referências ocorreu com base na análise dos resumos e da relevância temática dos materiais encontrados. Todo o estudo foi estruturado a partir da leitura crítica dos documentos selecionados, visando compreender não apenas os marcos históricos, mas também as tendências e os debates atuais que moldam a prática da Engenharia de *Software*. Dessa forma, a metodologia adotada buscou assegurar uma visão ampla e fundamentada sobre a evolução dessa área estratégica para o desenvolvimento tecnológico.

5. Discussão

A Engenharia de *Software* vive hoje uma crise de pensamento crítico. Como alerta Brooks (1975 p. 179), "não existem balas de prata" no desenvolvimento de *software*, e a complexidade inerente não pode ser eliminada com ferramentas automáticas. A cultura ágil e a popularização de assistentes de codificação promoveram rapidez, mas, ao mesmo tempo, esvaziaram a compreensão profunda dos fundamentos. Hoje, a necessidade de resgatar esse espírito de estudo sério e experimentação controlada é mais urgente do que nunca [Manifesto Agile 2001].

Bass, Clements & Kazman (2021) ressaltam que "a negligência em arquiteturas sólidas gera sistemas frágeis e insustentáveis". Na prática atual, muitos novos desenvolvedores dependem da IA sem questionar a validade do que é produzido, fenômeno esse agravado pela falta de estudo teórico consistente [Sommerville 2010 p. 14].

Para reverter esse quadro, se torna necessário revisitar a formação sólida em paradigmas de programação, teoria da computação, algoritmos e estruturas de dados. Como reforça Knuth (1997), "a programação deve ser encarada como uma arte que exige rigor, criatividade e profundo domínio técnico".

A reflexão de Kalinowski *et al.* (2023) sobre a “comparação entre *software* e arte” destaca a importância de se considerar as consequências de falhas em sistemas, especialmente os inteligentes, que podem afetar diretamente a sociedade. Embora a criatividade seja essencial, a Engenharia de *Software* deve garantir que a inovação seja acompanhada por rigor técnico e responsabilidade. A dependência crescente de IA sem compreensão profunda dos fundamentos pode aumentar significativamente os riscos operacionais e éticos.

A cultura da velocidade, impulsionada pela era ágil [Manifesto Agile 2001] e pela IA generativa, privilegiou soluções rápidas em detrimento da qualidade e da reflexão. A falta de um estudo teórico consistente e a dependência das soluções automáticas, como apontado por Sommerville (2010), cria um paradoxo: ao buscar eficiência e rapidez, estamos, muitas vezes, comprometendo a qualidade e a robustez do que está sendo produzido, seja um código de *software* ou um aprendizado educacional.

A afirmação de Gianella (2024), destaca a importância das habilidades fundamentais como programação e solução de problemas na transição para a engenharia de IA enfatizando também a necessidade de dominar aprendizado de máquina, estatísticas e processamento de

linguagem natural, além de experiência prática com *frameworks* e manipulação de grandes conjuntos de dados. Esses elementos são essenciais para uma adaptação eficaz ao campo da IA.

Sobre a Engenharia de *Software* moderna, se deve incorporar considerações críticas sobre o uso de inteligência artificial, especialmente em sistemas interativos como o *ChatGPT*. Conforme alertado pela *OpenAI* em sua interface, "o *ChatGPT* pode cometer erros. Considere verificar informações importantes" [*OpenAI* 2023 - tradução nossa], destacando a necessidade de validação contínua em ambientes onde contextos complexos e validação científica são essenciais. Ainda na mesma aba, existe *PrevenIA - Consultor Técnico* [OpenAI 2025], com a seguinte mensagem: “Esse GPT tem por objetivo ser um consultor técnico, dando apoio sobre as normas regulamentadoras do trabalho. O ChatGPT pode cometer erros. Considere verificar informações importantes”.

. Marcus & Davis (2020) reforçam que apesar da sofisticação na geração de texto, a IA ainda carece de compreensão contextual e garantia de precisão científica, exigindo revisão humana rigorosa.

Além disso, autores como Kreps & Kriner (2023) enfatizam que a IA pode perpetuar vieses sociais e epistemológicos presentes nos dados de treinamento, enquanto Bender *et al.* (2021) alertam para os riscos associados à falta de transparência nos modelos quanto às suas fontes e processos de inferência. Assim, a crítica responsável não é apenas uma questão técnica, mas um compromisso ético fundamental do pesquisador com a qualidade, veracidade e integridade de sua produção.

A superficialidade se manifesta também na adoção acrítica de linguagens e *frameworks*, escolhidos mais por modismo do que por adequação técnica [Fowler 2010]. Essa nova leva de desenvolvedores, educados sob a lógica da instantaneidade digital, prefere "soluções mágicas" à construção consciente de conhecimento. Não se trata apenas de avanço tecnológico, mas de uma mudança de mentalidade: da construção consciente para a dependência automática. O resultado é uma geração de desenvolvedores que muitas vezes reproduzem erros e vulnerabilidades, sem entender os fundamentos.

6. Considerações Finais

Retomar a excelência na Engenharia de *Software* exige revisitar os fundamentos históricos, reaprender as bases conceituais e práticas, e cultivar pensamento crítico. A solução não é negar a tecnologia emergente, mas usá-la com discernimento. Somente com critério, estudo e responsabilidade poderemos construir sistemas robustos, éticos e sustentáveis. Este retorno aos princípios fundamentais é relevante para que os profissionais da área possam enfrentar de maneira consciente a complexidade crescente dos sistemas e a velocidade das mudanças tecnológicas. Para isso, é imperativo que a educação e a formação continuada dos desenvolvedores envolvam não apenas o domínio das ferramentas mais modernas, mas também a compreensão profunda das teorias e práticas que formam a sua base.

A questão inicial deste ensaio foi respondida ao destacar que a dependência de assistentes automatizados pode comprometer a qualidade do *software* e enfraquecer a formação crítica dos profissionais da área. A utilização excessiva dessas ferramentas pode levar a soluções superficiais e reduzir a compreensão profunda dos fundamentos da Engenharia de *Software*. Além disso, a falta de reflexão e questionamento técnico impede que os desenvolvedores antecipem problemas complexos e tomem decisões bem fundamentadas. A resposta a essa questão aponta para a necessidade de equilibrar o uso dessas tecnologias com uma base sólida de conhecimento.

Entretanto, esta pesquisa está limitada a uma análise teórica e conceitual, com foco nos desenvolvimentos históricos e nas mudanças de paradigma, sem a exploração empírica ou de estudos de caso que poderiam ilustrar as consequências práticas dessas transformações no dia a dia dos engenheiros de software. A ausência de uma abordagem empírica impede uma compreensão mais completa de como as mudanças nas práticas de desenvolvimento e a introdução de novas tecnologias impactam a dinâmica do trabalho e a qualidade dos sistemas desenvolvidos.

Para um estudo mais aprofundado, seria valioso investigar como cada evolução na Engenharia de *Software*, desde os primeiros modelos de desenvolvimento até as abordagens ágeis e o uso de IA generativa, influenciou as práticas de desenvolvimento, os problemas encontrados ao longo do tempo, e as soluções adotadas. Estudar casos concretos e suas implicações permitirá uma análise mais clara dos benefícios e limitações de cada paradigma. A evolução da área é marcada por tensões entre inovação e tradição, eficiência e qualidade, rapidez e reflexão. Compreender essas tensões, suas consequências práticas e como foram abordadas ao longo do tempo é essencial para aprimorar o ensino e a prática, promovendo uma formação mais sólida e uma maior capacidade de adaptação às novas demandas do mercado.

Portanto, um estudo empírico sobre as consequências práticas das transformações tecnológicas na Engenharia de *Software*, focando nas dificuldades, erros e sucessos dos engenheiros ao longo do tempo, poderia fornecer *insights* cruciais para uma formação mais consciente e uma abordagem mais equilibrada no uso de novas ferramentas e metodologias. Isso contribuiria para a construção de sistemas mais robustos, sustentáveis e éticos, alinhados com a evolução da tecnologia e as exigências do mundo real.

Referências

Ambler, S. W. (2001). "***The Threat of the New"***. [*online*], Idioma Ingles, *London: Dr. Dobb's - Informa PLC*, [n.p.] Disponível em: <https://www.drdobbs.com/the-threat-of-the-new/184414798>. Acesso em 25 abr. 2025.

Ambler S. W.; & Lines M. (2012). "***Disciplined Agile Delivery:*** *A Practitioner's Guide to Agile Software Delivery in the Enterprise (IBM Press)"***.** Idioma Ingles. *1st Edition, Kindle Edition*, [s.l.]: *IBM Press*. ISBN: 9780132810104.

Armbrust, M. *et al.* (2010). ***A view of cloud computing.*** *Communications of the ACM*, v. 53, n. 4, p. 50–58 (9 p.). DOI: 10.1145/1721654.1721672. Disponível em: <https://www.researchgate.net/publication/220422375_A_View_of_Cloud_Computing>. Acesso em: 25 mai. 2025.

Ambysoft (2022). "***Examining the Agile Manifesto: Think Outside the Agile Box"***. [*online*]. Idioma Ingles. [s.l.]: *Ambysoft Inc*., *Copyright* 2002-2022, [n.p.]. Disponível em: <https://www.ambysoft.com/essays/agileManifesto.html>. Acesso em 20 abr. 2025.

Amershi, S. *et al*. (2019), "***Software Engineering for Machine Learning: A Case Study***," *IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Montreal, QC, Canada*, pp. 291-300. DOI: 10.1109/ICSE-SEIP.2019.00042. Disponível em: Disponível em: <https://ieeexplore.ieee.org/document/8804457<. Acesso em: 18 abr. 2025.

Anderson, D. J. (2010). "***Kanban: Successful Evolutionary Change for Your Technology Business"***. Idioma ingles. *Blue Book ed., Blue Hole Press,* 278 p. ISBN: 9780984521401.

ANSI X3.9 (1978). "***American National Standard Programming Language FORTRAN"***. *Archive Pdf* [*online*] Idioma Ingles. ANSI ® X3.9-1978 - *Revision of* ANSI X3.9-1966,

[s.l.]: *American National Standards Institute*, *Inc*. Disponível em: <https://web.archive.org/web/20171002221014/http://www.eah-jena.de/~kleine/history/languages/ansi-x3dot9-1978-Fortran77.pdf>. Acesso em: 25 abr. 2025.

Bacci, J. (2025). "***Vibe Coding"***: o que é e como surgiu essa tendência. [*online*]. São Paulo: Distrito Tecnologia e Serviços S.A., [n.p.]. Disponível em: <https://distrito.me/blog/vibe-coding-o-que-e-exemplos-beneficios-e-como-usar/>. Acesso em: 16 abr. 2025.

Bass, L.; Clements, P.; & Kazman, R. (2021). "***Software Architecture in Practice***". [*online*] *Pdf.* Idioma ingles*. The SEI Series in Software Engineering*, 3. ed. *Boston: Addison-Wesley*, 560 p. Disponível em: <https://ptgmedia.pearsoncmg.com/images/9780321815736/samplepages/0321815734.pdf >. Acesso em: 15 abr. 2025.

Beck, K; (2000). "***Extreme Programming Explained:*** *Embrace Change"*. 2 ed. *Addison-Wesley, Massachusetts: Pearson Education*, 187 p. ISBN: 0321278658. Disponível em: <https://ptgmedia.pearsoncmg.com/images/9780321278654/samplepages/9780321278654.pdf>. Acesso em: 15 abr. 2025.

Bender, E. M.; Gebru, T.; Mcmillan-Major A.; & Shmitchell, S. (2021). "***On the Dangers of Stochastic Parrots:*** *Can Language Models Be Too Big?". Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*. [*online*]. [s.l.]: *ACM Digital Library,* pp. 610-623. Disponível em: <https://doi.org/10.1145/3442188.3445922>. Acesso em: 03 mai. 2025.

Bender *et al*. (2001). "***Manifesto for Agile Software Development***". [*online*]. [s.l.]: Ward Cunningham, [n.p.]. Disponível em: <https://agilemanifesto.org/>. Acesso em: 27 abr. 2025.

Bisette, V. (2024). "***Cobol & Fortran:*** *The Legacy Systems Programming Guide 2024"*. [s.l.]: *Independently Published*, 508 p. ISBN: 9798328079587.

Booch, G. (2019). ***Computer Society***. [*online*] *Pdf. Published by the IEEE Computer Society*, 7 p. Disponível em: < https://ieeecs-media.computer.org/media/marketing/cedge_newsletter/ce9boo.pdf>. Acesso em: 20 abr. 2025.

Booch, G.; Rumbaugh, J.; & Jacobson, I; (1998). "***The Unified Modeling Language User Guide"***. [*online*] *Pdf*, Idioma ingles. 1. ed., ISBN: 0201571684, *Reading: Addison-Wesley*, 512 p. Disponível em: <https://patologia.com.mx/informatica/uug.pdf>. Acesso em: 19 abr. 2025.

Brooks, F. P. (1975). "***The Mythical Man-Month:*** *Essays on Software Engineering*". [online] *Pdf*. 1. ed. *Boston: Addison-Wesley*, 322 p. Disponível em: <https://www.cesarkallas.net/arquivos/livros/informatica/Addison.Wesley.The.Mythical.Man-Month.Essays.on.Software.Engineering.20th.Anniversary.Edition.pdf>. Acesso em 20 abr. 2025.

Cardelli, L. (1996). "***Bad engineering properties of object-orient languages"***. [*online*]. Doi.org/10.1145/242224.242415. *CA, USA: ACM Computing Surveys* (*CSUR*). V. 28, Issue 4es, Pages 150 p. Disponível em: <https://dl.acm.org/doi/fullHtml/10.1145/242224.242415>. Acesso em: 08 jul. 2025.

Ferreira, V. O. (2018). "***Desenvolvimento vs Segurança?"***. [*online*]. *SlideShare***.** 26ª edição do GTS - Grupo de Trab. em Seg. de Redes do Comitê Gestor da Internet no Brasil .

Brasília: CGI.br, 40 p. Disponível em: <https://medium.com/securityin/palestra-gts-26-desenvolvimento-versus-seguran%C3%A7a-8d799effe15a>. Acesso em: 20 abr. 2025.

Gianella, B. (2024). "***Migrando de Engenheiro de Software para Engenheiro de IA***". [online]. [s.l.]: Medium, [n.p.]. Disponível em: <https://medium.com/@bruno.giannellam/migrando-de-engenheiro-de-software-para-engenheiro-de-ia-274af6000077>. Acesso em: 28 abr. 2025.

IBM (2025). "***O que é blockchain?"*** [*online*]. *Amonk, NY*: IBM, [n.p.]. Disponível em: <https://www.ibm.com/br-pt/topics/blockchain>. Acesso em 09 jul. 2025.

IDN (2024). "***International Conference "NATO and Euro-Atlantic Security in a Geopolitically Competitive World"".*** [*online*]. Porto: Inst. da Defesa Nacional, [n.p.]. Disponível em: <https://www.idn.gov.pt/pt/noticias/Paginas/Conferencia-Internacional-NATO-and-Euro-Atlantic-Security-in-a-Geopolitically-Competitive-World.aspx>. Acesso em: 23 abr. 2025.

Kalinowski, M. *et al*. (2023). "***Engenharia de Software para Ciência de Dados***": Um guia de boas práticas com ênfase na construção de sistemas de *Machine Learning* em *Python*. [*online*] *Book*. São Paulo: Casa do Código, ISBN: 9788555193347. Disponível em :<https://www.casadocodigo.com.br/products/livro-escd>. Acesso em: 25 abr. 2025.

Karpathy, A. (2023). "***Andrej Karpathy Vibe Coding***". [*online*] [s.l.]: Klover, Fonte adaptada de entrevistas e publicações de Andrej Karpathy, [n.p.]. Disponível em: <https://www.klover.ai/andrej-karpathy-vibe-coding/>. Acesso em: 04 jun. 2025.

Kernighan, B. W.; & Ritchie, D. M. (1988). "***The C Programming Language***". [*online*]. 2. ed. *Englewood Cliffs: Prentice Hall,* [n.p.]. ISBN: 0131103709. Disponível em: <https://www.cimat.mx/ciencia_para_jovenes/bachillerato/libros/%5BKernighan-Ritchie%5DThe_C_Programming_Language.pdf\>. Acesso em: 27 abr. 2025.

Kitchin, R; (2014). "***The Data Revolution: Big Data, Open Data, Data Science, and Data Analytics"***. 1. ed. *London: Sage Publications Ltd,* 240 p. ISBN: 9781446287477.

Knuth, D. E. (1997). **"*The Art of Computer Programming, Volume 1: Fundamental Algorithms"*.** 3. ed. Série: *The Art of Computer Programming Reading: Addison-Wesley Professional,* 672 p. ISBN: 9780201896831.

Kreps, S.; & Kriner, D. L. (2023). "***How AI Threatens Democracy***". *Journal of Democracy, Project MUSE*. vol. 34 no. 4, 2023, p. 122-131. Disponível em: <https://dx.doi.org/10.1353/jod.2023.a907693>. Acesso em: 03 mai 2025.

LINFO (2004). "***Source Code Definition***". [*online*]. [s.l.]: *Linfo.org*, [n.p.]. Disponível em: <https://www.linfo.org/source_code.html>. Acesso em: 09 jul. 2025].

Mell, P.; Grance, T.(2011). ***The NIST definition of cloud computing***. [*online*], [PDF], *NIST Special Publication* 800-145**.** *Gaithersburg: National Institute of Standards and Technology*, 7 p. Disponível em:<https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf>. Acesso em: 26 abr. 2025.

Manifesto Agile (2001). **"*Manifesto for Agile Software Development*"**. [*online*]. *Ward Cunninghan: Agile Manifesto*, Org . [n.p.]. Disponível em: <https://agilemanifesto.org/>. Acesso em: 27 abr. 2025.

Marcus, G., & Davis, E. (2020). "***Rebooting AI: Building Artificial Intelligence We Can Trust*".** ISBN: 9781524748258. New York: Pantheon Books, 288 p.

Osmani, A. (2025). “***Vibe Coding: O The Future of Programming”****: Leverage Your Experience in the Age of AI*. Idioma ingles. *O'Reilly Media, Inc.* ISBN: 9798341634756.

OpenAI (2023). “***ChatGPT”***. [s.l.]: *Chatgpt*. Disponível em: <https://chatgpt.com/>. Acesso em: 05 jul. 2025.

OpenAI (2025). “***PrevenIA - Consultor Técnico”.*** [s.l.]: *Chatgpt*. Disponível em: <https://chatgpt.com/g/g-92drIpC5Q-prevenia-consultor-tecnico>. Acesso em: 09 jul. 2025.

Pearce, H. *et al.* (2022). ***Asleep at the Keyboard? Assessing the Security of GitHub Copilot’s Code Contributions***. *In: IEEE Symposium on Security and Privacy*. Disponível em: <https://doi.org/10.48550/arXiv.2108.09293>. Acesso em: 15 abr. 2025.

Penzenstadler, B. *et al.* (2012). "***Sustainability in Software Engineering: A Systematic Literature Review***". *Journal of Systems and Software. DOI:10.1049/ic.2012.0004.* Disponível em: <https://www.researchgate.net/publication/255949743_Sustainability_in_Software_Engineering_A_Systematic_Literature_Review>. Acesso em: 27 abr. 2025.

Pressman, R. S.; & Maxim, B. R. (2016). "***Engenharia de Software:*** *uma abordagem profissional*". Traduzido por: Reginaldo Arakaki, Julio Arakaki e Renato Manzan de Andrade. 8. ed. Porto Alegre: AMGH Editora, 968 p. ISBN: 9788580555332.

SAP (2025). “***O que é o desenvolvimento de aplicativos low-code/no-code?”*** [*online*]. São Paulo: SAP SE, [n.p.]. Disponível em: <https://www.sap.com/brazil/products/technology-platform/build/what-is-low-code-no-code.html>. Acesso em: 09 jul. 2025.

Schwaber, K.; & Beedle, M. (2001). “***Agile Software Development with Scrum”***. *Prentice-Hall, Upper Saddle River, 1st Edition, Person, 176 p.* ISBN: 9780130676344.

Sommerville, I. (2010). "***Software Engineering***". 9. ed. *Londes: Pearson*, 792 p. ISBN: 9780137035151.

Squeak (2025). “***Welcome to Squeak/Smalltalk”***. [*online*] [s.l.]: *Squeak* Org, 1996-2025, [n.p.]. Disponível em: <https://squeak.org/>. Acesso em: 09 jul. 2025.

Stroustrup, B. (1985). “***The C++ Programming Language”****.* 1. ed. *Reading: Addison-Wesley*.

Susskind, R. (2020). “***The Future of the Professions: How Technology Will Transform the Work of Human Experts”****. Oxford: Oxford University Press*.

TechTarget (2024). "***What is a framework?***" [*online*]. *Newton, MA: TechTarget*, [n.p.]. Disponível em: <https://www.techtarget.com/whatis/definition/framework>. Acesso em 05 jun. 2025.

The Chaos Manifesto (2011). “***Chaos Manifesto:*** *The Laws of CHAOS and the CHAOS 100 Best PM Practices*”. [*online*] [*Pdf*]. Idioma Ingles. *Boston: The standish Group*, 52 p. Disponível em: <https://web.archive.org/web/20150501022820/http://www.versionone.com/assets/img/files/ChaosManifest_2011.pd>. Acesso em: 09 jul. 2025.

WHO (2020). *“**COVID-19 Strategy Update**”*. [*online*]. *Pdf*. [s.l.]: *World Health Organization*, [n.p.]. Disponível em: <https://www.who.int/publications/m/item/covid-19-strategy-update>. Acesso em: 27 abr. 2025.

Sobre Autor

[i] **Iraê César Brandão**

O autor se dedica a explorar as contradições do cotidiano, a potência do pensamento reflexivo e a beleza da transformação pessoal. Sua trajetória profissional transita pelas áreas de tecnologia, comunicação, educação e escrita, sempre guiado por uma inquietação intelectual que o leva a buscar novas leituras, formações e experiências práticas. Em constante processo de reciclagem e ampliação de seus conhecimentos, mantém uma rotina de estudos exaustivos, fundamentada na curiosidade e no desejo de compreender melhor os desafios da modernidade e suas evoluções.

Seus objetivos envolvem promover o diálogo entre profissionais de TI, professores, gestores, colaboradores e clientes, incentivando parcerias inovadoras e o intercâmbio de ideias. Aberto à criação de novas soluções, produtos e serviços, atua com responsabilidade ética e legal, sempre comprometido com o desenvolvimento humano, tecnológico e com a construção de uma cidadania mais consciente e colaborativa.

Formações Acadêmicas: Graduado em Gestão de Tecnologia da Informação pela UNICSUL, com MBA Executivo em Segurança Cibernética (FI) e MBA Executivo em Gestão Estratégica de *Marketing*, Planejamento e Inteligência Competitiva (FI).

Pós-graduado e especialista nas seguintes áreas:

- Filosofia (FAAL);
- Sociologia (FAAL);
- Uso Educacional da Internet (UFLA);
- Docência do Ensino Superior e Neuropsicologia (Faculeste);
- Docência em Administração (Faculeste);
- Docência para Educação Profissional e Tecnológica (Faculeste);
- Ciências da Natureza, suas Tecnologias e Mundo do Trabalho (UFPI);
- Linguagens, suas Tecnologias e Mundo do Trabalho (UFPI);
- Matemática e suas Tecnologias e Mundo do Trabalho (UFPI);

Aperfeiçoamentos e Formação Complementar: Com um olhar atento às transformações educacionais e tecnológicas, o autor investe continuamente em aperfeiçoamentos alinhados às diretrizes da Base Nacional Comum Curricular (BNCC), incluindo formações nas áreas de Mundo do Trabalho, Matemática e suas Tecnologias, Ciências da Natureza e suas Tecnologias, e Linguagens e suas Tecnologias, por meio de plataformas como AVAMEC e SEB.

Sua trajetória inclui uma sólida formação extracurricular em Desenvolvimento de Sistemas, Programação Web e diversas linguagens de programação (*JavaScript, HTML, CSS, Python, C#,* entre outras), bem como especializações em *Azure*, *Power BI*, *Marketing* Digital, Qualidade e Testes de *Software* (Q&A), Tecnologias Digitais da Informação e Comunicação (TDICs), Educação Midiática, Perícia Forense Computacional, Empreendedorismo, Ciências Contábeis, Lei Geral de Proteção de Dados (LGPD), *Cloud Computing*, Inteligência Artificial e *Machine Learning*, entre outras áreas voltadas à Tecnologia da Informação.

Área de Atuação: Atua como empresário na área de Tecnologia e Segurança da Informação há mais de 25 anos. Na Educação, integra a Rede Estadual de Ensino, lecionando Tecnologia da Informação em cursos técnicos e nas disciplinas do Novo Ensino Médio, com foco nas competências da BNCC. Também exerce atividades como tutor universitário, ampliando seu alcance na formação de novos profissionais.

https://orcid.org/0000-0002-2079-0615

https://iraecbrandao.com.br

https://www.linkedin.com/in/irae-cesar-brandao-a2112b69/

http://lattes.cnpq.br/3757125329283407

https://www.researchgate.net/profile/Irae-Brandao-2