REVISTA TÓPICOS
ISSN 2965-6672 | Qualis A2

ALÉM DO VIBE CODING: UM MODELO DE ENGENHARIA DE SOFTWARE ASSISTIDA POR INTELIGÊNCIA ARTIFICIAL BASEADO EM ARQUITETURA, CONTEXTO, VERIFICAÇÃO E GOVERNANÇA

BEYOND VIBE CODING: A MODEL OF AI-ASSISTED SOFTWARE ENGINEERING BASED ON ARCHITECTURE, CONTEXT, VERIFICATION, AND GOVERNANCE

Engenharias • 06/11/2026

REGISTRO DOI: 10.70773/revistatopicos/793690700

Raul Greco Junior[1]

Vanina Roncaglio[2]

Thyago Bohrer Borges[3]

Giuseppe Silva Favato[4]

RESUMO

A disseminação de modelos de linguagem capazes de gerar e modificar código deslocou parte do desenvolvimento de software da escrita manual para a interação em linguagem natural, prática popularizada como vibe coding. Este artigo examina as consequências de engenharia dessa transição e propõe um modelo conceitual e operacional de Engenharia de Software Assistida por Inteligência Artificial (ESAIA), estruturado nos eixos arquitetura, contexto, verificação e governança. O estudo, teórico-conceitual combina revisão integrativa de evidências de 2022 a 2026, com proposições apoiadas em periódicos revisados por pares e documentos normativos; os demais servem apenas como referência de coerência. A base revisada converge em três pontos: atributos não funcionais resistem ao controle por solicitação em linguagem natural e aceleram a dívida técnica; instrumentos que medem mais de uma propriedade revelam queda consistente de desempenho e falhas elementares persistentes; e a trajetória normativa recente trata segurança e qualidade como propriedades do processo, verificadas por mecanismos declarados. O ESAIA reúne sete camadas (intenção, especificação, arquitetura, contexto, execução, verificação e governança), seis métricas observáveis e um protocolo experimental de quatro condições que isola o efeito do processo, mantidos constantes modelo, tarefa e orçamento. A hipótese central é que a qualidade da mudança depende menos da fluência do modelo e mais da densidade de mecanismos verificáveis que o circundam, em intensidade crescente com a criticidade e a complexidade da tarefa. O problema está menos em delegar a implementação à inteligência artificial do que em fazê-lo sem mecanismos que a tornem verificável e rastreável.

Palavras-chave: engenharia de software; inteligência artificial; vibe coding; agentes de código; qualidade de software; governança.

ABSTRACT

The spread of large language models capable of generating and modifying code has shifted part of software development from manual writing to natural-language interaction, a practice popularized as vibe coding. This article examines the engineering consequences of that shift and proposes a conceptual and operational model of AI-Assisted Software Engineering (ESAIA), organized along four axes: architecture, context, verification, and governance. The theoretical-conceptual study combines an integrative review of evidence from 2022 to 2026, with claims resting on peer-reviewed journal articles and normative documents; the remaining works serve only as coherence references. The reviewed base converges on three points: non-functional attributes resist control through natural-language prompting and accelerate technical debt; assessment instruments that measure more than one property reveal consistent performance degradation and persistent elementary faults; and recent normative work treats security and quality as process properties, verified through declared mechanisms. ESAIA brings together seven layers (intent, specification, architecture, context, execution, verification, and governance), six observable metrics, and a four-condition experimental protocol that isolates the effect of the engineering process while holding model, task, and budget constant. The central hypothesis is that change quality depends less on model fluency than on the density of verifiable mechanisms surrounding it, with an optimal intensity that grows with task criticality and complexity. The problem lies less in delegating implementation to artificial intelligence than in doing so without explicit mechanisms that make production verifiable and traceable.

Keywords: software engineering; artificial intelligence; vibe coding; coding agents; software quality; governance.

1. INTRODUÇÃO

A engenharia de software constituiu-se historicamente como a tentativa de converter uma atividade criativa e sujeita a falhas em um processo suficientemente disciplinado para produzir sistemas úteis, manuteníveis, seguros e economicamente sustentáveis. A introdução de modelos de linguagem de grande porte na programação não dissolve esse problema fundador; desloca-o. Escrever sintaxe deixa de ser o centro do trabalho, enquanto especificar, contextualizar, restringir, observar e validar passam a determinar a confiabilidade do resultado. O gargalo migra da produção para o julgamento, e é aí que reside a dificuldade: a capacidade de gerar código cresceu mais rápido que a capacidade humana de revisar cada decisão local que esse código materializa.

O termo vibe coding consolidou-se para descrever a programação mediada por conversação com inteligência artificial. Sarkar e Drosos (2025), em estudo empírico dedicado ao fenômeno — trabalho de anais aqui tomado como referência de coerência —, analisaram mais de oito horas de sessões e observaram ciclos iterativos de solicitação, avaliação rápida, teste e edição. A caracterização contraria duas simplificações opostas: não é verdade que o desenvolvedor deixe de exercer expertise, pois ela permanece, redistribuída para o gerenciamento de contexto, a inspeção e a decisão sobre quando confiar, corrigir ou interromper a automação; tampouco é verdade que a simples presença humana no ciclo garanta controle, já que o que se avalia em uma inspeção rápida é, em grande medida, o comportamento visível do artefato.

O problema investigado neste artigo decorre dessa redistribuição: a facilidade de obter código executável pode reduzir, na prática, o

rigor de especificação, de arquitetura, de verificação e de governança que a engenharia de software desenvolveu ao longo de décadas. A literatura disponível caracteriza bem o fenômeno em suas extremidades: descreve-se a prática conversacional (Sarkar; Drosos, 2025), mede-se o desempenho de modelos em benchmarks (Jimenez *et al.*, 2024; Nogueira; Vieira; Campos, 2026) e documentam-se rupturas na interação prolongada (Te'eni; Katz, 2026). No entanto, há pouco formalismo quanto ao objeto intermediário: o sistema de engenharia que circunda o modelo e decide se uma alteração pode ser incorporada ao produto. Faltam, em particular, uma descrição funcional desse sistema que não dependa de fornecedor, métricas que cubram simultaneamente resultado e custo e um desenho experimental capaz de estimar o efeito do processo com modelo e tarefa mantidos constantes.

A pergunta de pesquisa que organiza o trabalho é, portanto: como estruturar o desenvolvimento assistido por inteligência artificial de modo a preservar os ganhos de automação sem abandonar os princípios de engenharia de software? O objetivo geral é propor um modelo conceitual e operacional de Engenharia de Software Assistida por Inteligência Artificial, doravante ESAIA, apoiado em quatro eixos: i) arquitetura, ii) contexto, iii) verificação e iv) governança. Como objetivos específicos, o artigo sintetiza criticamente as evidências recentes sobre produtividade, segurança e qualidade não funcional; delimita o conceito de vibe coding em termos operacionais; integra recuperação de contexto, protocolos de ferramentas e agentes às práticas clássicas de engenharia; define métricas observáveis; e propõe um protocolo experimental replicável.

A hipótese central, desdobrada em proposições testáveis na Seção 6, é que a qualidade de uma mudança de software produzida com assistência de inteligência artificial depende menos da fluência do modelo do que da densidade de mecanismos verificáveis que o circundam, e que a intensidade ótima desses mecanismos cresce com a criticidade e a complexidade da tarefa. A formulação é deliberadamente restrita: ao indicar sob quais condições a qualidade da mudança varia, define o que observar e o que contaria contra ela, e por isso é mais fácil de submeter a teste do que as afirmações genéricas de que o vibe coding seria bom ou ruim, que nenhum resultado isolado permite refutar.

A contribuição pretendida é tripla. Oferece-se, primeiro, uma síntese crítica das evidências disponíveis, com atenção explícita à validade temporal dos resultados quantitativos. Propõe-se, em seguida, o framework ESAIA como descrição funcional, independente de fornecedor, das camadas que transformam intenção em mudança verificável. Apresenta-se, por fim, um instrumento de pesquisa: uma escala de maturidade, seis métricas e um protocolo de quatro condições que permitem tratar o processo de engenharia como variável experimental. O artigo descreve o método na Seção 2, delimita o objeto conceitual na Seção 3, sintetiza as evidências na Seção 4, apresenta o modelo na Seção 5, trata de maturidade, hipóteses e validação na Seção 6, discute implicações e limites na Seção 7 e conclui na Seção 8.

2. MÉTODO

Este trabalho caracteriza-se como pesquisa teórico-conceitual, conduzida por meio de revisão integrativa orientada por evidências e culminando na proposição de um framework. A escolha decorre

da natureza do problema: o objeto a caracterizar — o sistema de engenharia que circunda modelos generativos — ainda não dispõe de descrição consolidada, de modo que a contribuição possível nesta etapa é de síntese e formalização, não de medição primária. Não são apresentados resultados primários do autor: todos os valores atribuídos a desempenho, segurança, produtividade e prevalência de falhas provêm de estudos publicados e são identificados pela respectiva referência. A distinção é necessária para que os cenários de validação da Seção 6 não sejam confundidos com achados já observados.

A revisão cobriu publicações de 2022 a setembro de 2026 nos domínios de engenharia de software assistida por inteligência artificial, geração de código, agentes, produtividade, segurança de código gerado, qualidade não funcional, recuperação de contexto aplicada a repositórios, benchmarking e protocolos de exposição de ferramentas. Privilegiaram-se periódicos e conferências com revisão por pares, repositórios oficiais de anais, documentos normativos e fontes primárias institucionais, entre os quais Empirical Software Engineering, Journal of Systems and Software, os anais do ICLR e do ICML, o PPIG, o HHAI, as publicações do NIST, a norma ISO/IEC 25010, o relatório DORA e a especificação do Model Context Protocol.

Quatro critérios de rigor foram aplicados à extração. O primeiro é a hierarquia da evidência: as proposições do artigo sustentam-se em artigos publicados em periódicos com revisão por pares e em documentos normativos, enquanto trabalhos publicados em anais de congresso, pré-prints e relatórios setoriais são empregados apenas como referência de coerência, isto é, para corroborar, ilustrar ou delimitar o alcance daquilo que a base revisada já estabelece. O segundo é a fidelidade à fonte: para cada dado quantitativo

utilizaram-se os valores reportados no documento original, verificados individualmente contra registros bibliográficos. O terceiro é a contextualização temporal: resultados com forte dependência do estado da tecnologia são apresentados como medidas de um conjunto específico de ferramentas e participantes em um período determinado, acompanhados dos avisos metodológicos posteriores dos próprios autores. O quarto é a preferência por versões publicadas sobre pré-prints, com explicitação do status editorial quando a literatura ainda é emergente.

O desenho impõe limites que convém declarar desde já. Uma revisão integrativa não produz estimativas de efeito agregadas e não substitui metanálise; a seleção de fontes, embora orientada por critérios explícitos, não seguiu protocolo de revisão sistemática com dupla triagem independente; e o framework proposto é, neste artigo, uma construção teórica sustentada por evidências convergentes, não um artefato validado empiricamente. É essa limitação, aliás, que motiva a formulação do protocolo experimental apresentado adiante.

3. DO VIBE CODING À ENGENHARIA ASSISTIDA

A literatura clássica define engenharia de software pelos processos que cercam a produção de código: especificação, projeto, implementação, verificação, validação, evolução e gestão (Sommerville, 2016). A automação da escrita não suprime nenhuma dessas funções; altera quem ou o que as executa e, principalmente, torna urgente decidir quais permanecem sob responsabilidade humana. Para que a discussão seja empiricamente tratável, vibe coding é aqui definido como o regime de desenvolvimento em que a especificação é predominantemente conversacional, a verificação

é predominantemente observacional e a decisão de aceitação é tomada com base no comportamento aparente do artefato, e não em evidências sistemáticas de conformidade. A definição é intencionalmente gradativa: situa-se num contínuo cuja outra extremidade é a engenharia assistida, em que a conversação permanece como interface, mas a aceitação depende de critérios declarados e verificáveis.

Ao longo desse contínuo é útil distinguir três níveis, que diferem não pela tecnologia empregada, mas pela unidade de valor que cada um reconhece. Na geração de código, a unidade de valor é a resposta do modelo, e a evidência de sucesso é a plausibilidade sintática. Na assistência ao desenvolvimento, é a tarefa concluída com apoio da ferramenta, atestada por execução local e algum teste. Na engenharia assistida, é a mudança aceita por um sistema de evidências, atestada pela aprovação em critérios funcionais, não funcionais, arquiteturais e de governança. O Quadro 1 sistematiza a distinção, que constitui a base conceitual do modelo proposto na Seção 5: o deslocamento do foco da fluência do modelo para o processo que decide se o código pode ser incorporado ao produto.

Quadro 1. Três níveis de utilização de inteligência artificial no desenvolvimento de software

Nível	**Pergunta dominante**	**Evidência de sucesso**	**Risco típico**
Geração	O modelo produziu código?	Resposta sintaticamente plausível	Confundir plausibilidade com correção
Assistência	A tarefa foi concluída?	Execução e testes locais	Validar somente o comportamento visível

Engenharia assistida	A mudança é aceitável no sistema?	Critérios funcionais, não funcionais, arquiteturais e de governança	Maior custo de preparação, compensado por rastreabilidade

Fonte: Elaboração do autor (2026), com base em Sommerville (2016), Sarkar e Drosos (2025) e Sun *et al.* (2026).

A passagem de um nível a outro não é gratuita. Ela exige preparação, formalização e infraestrutura de verificação, isto é, custos que só se justificam quando o risco da mudança o requer. Essa condicionalidade será retomada como hipótese de moderação na Seção 6 e é o que impede que o modelo proposto seja lido como recomendação de rigor máximo e uniforme.

4. EVIDÊNCIAS EMPÍRICAS SOBRE O PROBLEMA

As evidências disponíveis não formam um corpo homogêneo, e sua leitura exige ordenar o que sustenta e o que apenas corrobora. Esta seção apresenta primeiro a base evidencial do artigo, composta por periódicos revisados por pares e documentos normativos, da qual derivam as proposições defendidas adiante, e em seguida os registros de anais, pré-prints e relatórios setoriais, que mostram achados independentes apontando na mesma direção.

A base evidencial começa pela qualidade não funcional. Sun *et al.* (2026) revisaram 109 trabalhos, conduziram dois workshops com profissionais e avaliaram empiricamente correções produzidas por três modelos de linguagem. Guiado pela ISO/IEC 25010, o estudo encontrou desalinhamento entre as ênfases acadêmicas, as prioridades da indústria e o comportamento efetivo dos modelos: enquanto a literatura concentra-se em segurança, eficiência de

desempenho e manutenibilidade, os profissionais destacam manutenibilidade e legibilidade e alertam para a aceleração da dívida técnica. Os autores observaram, ainda, instabilidade quando atributos não funcionais eram otimizados apenas por solicitação em linguagem natural. A implicação é imediata: requisitos não funcionais exigem critérios mensuráveis e mecanismos de aceitação externos ao modelo.

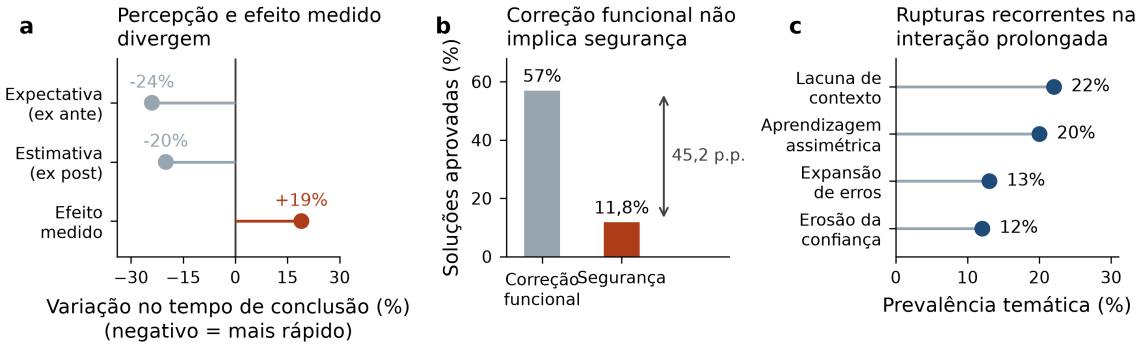
O segundo pilar da base diz respeito ao modo de avaliar. Nogueira, Vieira e Campos (2026) propõem o PROBE, benchmark de 1.651 problemas em cinco linguagens, avaliado sobre seis modelos e três estratégias de prompting, que combina correção funcional, proximidade de soluções válidas e qualidade de código. A proposta amplia o paradigma de avaliação consolidado pelo SWE-bench, que mede agentes pela resolução de issues reais verificada por testes (Jimenez *et al.*, 2024), e os autores observam degradação consistente de desempenho com o aumento da dificuldade, além da persistência de falhas elementares, como estouros de inteiros, acessos fora dos limites de vetores e recursos a algoritmos de força bruta. O resultado é central para o argumento deste artigo porque desqualifica a leitura unidimensional da capacidade dos modelos: quando o instrumento mede mais de uma propriedade, a competência aparente diminui, e a diferença entre produzir uma resposta e produzir uma solução aceitável torna-se mensurável.

O terceiro pilar é normativo. O Secure Software Development Framework do NIST organiza, na versão 1.1, práticas incorporáveis a qualquer ciclo de vida, com o propósito de reduzir vulnerabilidades, mitigar o impacto das não detectadas e atuar sobre suas causas (Souppaya; Scarfone; Dodson, 2022). Em 2024, o NIST publicou o perfil comunitário SP 800-218A, voltado ao desenvolvimento seguro

de sistemas de inteligência artificial generativa (NIST, 2024), e em dezembro de 2025 submeteu a consulta pública a revisão 1.2 (NIST, 2025). Somada à ISO/IEC 25010:2023, que define o modelo de qualidade de produto em nove características, essa trajetória fornece o critério de aceitação adotado neste artigo: segurança e qualidade são propriedades do processo, verificadas por mecanismos declarados.

Sobre essa base incidem as evidências corroborativas. A Figura 1 reúne três achados de desenhos distintos, sendo um experimento randomizado divulgado como pré-print, um benchmark de segurança publicado em anais e um estudo qualitativo longitudinal também de anais, que convergem com a base revisada em registros complementares.

Figura 1. Três dimensões de avaliação do desenvolvimento assistido por inteligência artificial



Fonte: Elaboração do autor (2026), com dados de Becker *et al.* (2025), em (a), n = 246 tarefas; Zhao *et al.* (2026), em (b), n = 186 tarefas; e Te'eni e Katz (2026), em (c), n = 163 episódios. Figura de corroboração: as três fontes são pré-print ou trabalhos de anais, utilizados como evidência convergente e não como fundamento das proposições. Em (c), os percentuais indicam prevalência temática e não constituem partição exaustiva.

No plano da produtividade, a principal lição é metodológica. Becker *et al.* (2025), em experimento randomizado conduzido pela METR, acompanharam 16 desenvolvedores experientes em 246 tarefas de

projetos maduros, nos quais possuíam em média cinco anos de experiência. Antes das tarefas, os participantes previram redução de 24% no tempo de conclusão quando pudessem usar ferramentas de inteligência artificial; após executá-las, estimaram redução de 20%. O efeito medido entre fevereiro e junho de 2025 foi de sinal oposto: aumento de 19% no tempo de conclusão. O valor não deve ser lido como medida atual e universal — em fevereiro de 2026 a própria METR informou que evidências posteriores sugeriam aceleração com ferramentas mais recentes, ainda que sob forte viés de seleção e intervalos de confiança amplos (METR, 2026). Como pré-print, o achado não fundamenta as proposições do artigo; corrobora o que a base normativa já pressupõe, ao mostrar que percepção e efeito medido podem divergir sistematicamente e que a primeira não deveria operar como critério de decisão organizacional.

No plano da segurança, a corroboração é mais direta. Zhao *et al.* (2026) propuseram o SUSVIBES, benchmark composto por 186 solicitações de funcionalidade provenientes de projetos de código aberto reais, selecionadas a partir de situações em que implementações humanas haviam introduzido vulnerabilidades, e avaliaram doze configurações agentivas com modelos de fronteira. Na configuração destacada pelos autores, 57% das soluções foram funcionalmente corretas e apenas 11,8% foram consideradas seguras — uma lacuna de 45,2 pontos percentuais entre duas propriedades que a validação superficial trata como equivalentes —, e estratégias preliminares de mitigação por enriquecimento da solicitação não bastaram para fechá-la. Trabalho de anais, o dado ilustra, em coerência com Sun *et al.* (2026) e com o referencial do NIST, que correção e segurança são propriedades distintas e exigem mecanismos de verificação distintos.

Há, por fim, a corroboração relativa à sustentação do contexto ao longo do tempo. Te'eni e Katz (2026) analisaram 163 episódios válidos de interação entre um desenvolvedor e um agente de código durante a construção e a depuração de um sistema ao longo de meses, e identificaram quatro dinâmicas recorrentes, com prevalências de 22% para determinantes de lacuna de contexto, 20% para aprendizagem assimétrica e não sincronizada, 13% para a espiral de expansão de erros e 12% para a espiral de erosão da confiança. Os mecanismos descritos — amnésia de sessão, decaimento de instruções, estreitamento arquitetural, violações repetidas de regras e correções que introduzem novos problemas — interessam menos por suas prevalências, derivadas de um único caso, do que por oferecerem categorias traduzíveis em controles de engenharia. No plano organizacional, o relatório DORA de 2025 sintetiza a inteligência artificial como amplificador das condições existentes: organizações com sistemas de engenharia sólidos convertem automação em desempenho, enquanto organizações frágeis amplificam instabilidade, retrabalho e gargalos (DeBellis *et al.*, 2025). Nenhum dos dois registros sustenta por si as proposições do artigo, mas ambos reforçam a tese de que a variável explicativa relevante não é o modelo empregado, e sim a qualidade do sistema que o cerca.

5. O MODELO ESAIA

O modelo proposto parte de uma premissa derivada da seção anterior: se a geração de código tornou-se abundante e a aceitação permaneceu escassa, então o valor marginal da engenharia deslocou-se da produção para a verificação. Um ambiente de desenvolvimento assistido deve, por consequência, tratar como requisito de primeira ordem a capacidade de decidir, com base em

evidências, se uma mudança pode ser incorporada ao produto. Essa decisão depende de quatro condições: critérios de aceite explícitos, contexto suficiente para que a implementação seja pertinente, verificação automatizada em múltiplos níveis e registro auditável do que foi feito, por quem e com qual autorização.

Quadro 2. Tradução de características de qualidade em critérios verificáveis e mecanismos de evidência

Característica de qualidade	**Exemplo de critério verificável**	**Mecanismo de evidência**
Adequação funcional	Totalidade dos critérios de aceite críticos aprovados	Testes unitários, de integração e ponta a ponta
Segurança	Nenhuma vulnerabilidade crítica ou alta sem aceitação formal	SAST, SCA, testes de autorização, revisão
Manutenibilidade	Complexidade e duplicação abaixo de limites definidos	Análise estática e regras de arquitetura
Eficiência de desempenho	Percentil 95 de latência dentro do orçamento	Benchmark e teste de carga
Confiabilidade	Recuperação e comportamento previsível em falhas	Testes de resiliência e observabilidade
Compatibilidade	Contratos preservados entre módulos e serviços	Testes de contrato e versionamento de APIs

Fonte: Elaboração do autor (2026), com base na ISO/IEC 25010:2023 e em Souppaya, Scarfone e Dodson (2022).

Com esse requisito como ponto de partida, propõe-se o ESAIA — Engenharia de Software Assistida por Inteligência Artificial —,

composto por sete camadas integradas e organizado pelos quatro eixos anunciados na introdução. O modelo não é uma arquitetura de produto nem depende de fornecedor: descreve funções que um ambiente de desenvolvimento deve cumprir para transformar intenção em mudança verificável. A camada de intenção formaliza problema, escopo, partes interessadas, criticidade e limites, delimitando a tarefa o suficiente para que sucesso e risco possam ser discutidos. A camada de especificação converte intenção em requisitos funcionais e não funcionais, critérios de aceite e definição de pronto, privilegiando condições verificáveis em lugar de adjetivos vagos como rápido, seguro ou moderno. A camada de arquitetura define componentes, responsabilidades, contratos e restrições, produzindo simultaneamente contexto para o agente e portões para o pipeline. A camada de contexto seleciona conhecimento relevante e confiável; a de execução abriga modelos, agentes e ferramentas; a de verificação converte afirmações do agente em evidências obtidas do ambiente; e a de governança, transversal às demais, mantém registros, proveniência, políticas, aprovações e auditoria. A Figura 2 representa o encadeamento, incluindo a realimentação das falhas de verificação sobre as camadas de contexto e execução.

Figura 2. Modelo ESAIA de sete camadas e seus eixos estruturantes



Fonte: Elaboração do autor (2026).

5.1. Arquitetura e Contexto Como Restrição Operacional

A arquitetura é a primeira barreira contra a geração localmente plausível e globalmente incoerente. Em sistemas construídos por equipes humanas, decisões arquiteturais delimitam responsabilidades, dependências, propriedades de qualidade e estratégias de evolução. Em sistemas assistidos por inteligência artificial, cumprem função adicional: restringem o espaço de busca do agente, informando-lhe não apenas o que pode fazer, mas o que não deve fazer. Para isso, a arquitetura precisa deixar de ser prosa e tornar-se informação legível por máquinas e, quando possível, regra executável. Um documento que ninguém consulta tem pouco valor operacional; já regras de dependência, testes arquiteturais, esquemas de API, contratos de eventos e registros de decisão podem ser recuperados durante a tarefa e verificados no pipeline.

A camada de contexto responde a uma assimetria estrutural. Modelos operam sobre uma janela finita e temporária, enquanto requisitos e decisões distribuem-se por issues, código, testes, documentação, histórico de commits, esquemas e incidentes. O problema de engenharia de contexto consiste em fornecer, no

momento correto, informação suficiente e confiável para uma decisão de implementação. A geração aumentada por recuperação é relevante porque separa o conhecimento persistente da janela imediata do modelo. Tao *et al.* (2025), em levantamento disponível como pré-print, observam que a produção em nível de repositório exige capturar dependências de longo alcance, coerência semântica e relações entre múltiplos arquivos — exigências que a recuperação externa atende sem carregar todo o repositório indiscriminadamente.

A presença de recuperação, no entanto, não garante a qualidade do contexto recuperado. Um pipeline pode retornar documentação obsoleta, exemplos incompatíveis ou trechos semanticamente próximos mas arquiteturalmente errados. Por isso, um mecanismo de recuperação voltado à engenharia deve registrar procedência, versão, escopo e prioridade das fontes: documentos normativos internos e contratos precisam ter precedência sobre trechos históricos, conteúdo depreciado precisa ser identificável e resultados recuperados precisam ser rastreáveis. Em lugar de um prompt monolítico, propõe-se que cada tarefa gere um pacote de contexto versionado, reunindo requisito e critérios de aceite, componentes afetados, decisões de arquitetura relacionadas, contratos, testes existentes, regras de segurança, riscos conhecidos e ferramentas autorizadas. O agente passa a operar sobre uma visão situada do sistema, e a colaboração reduz sua dependência da memória conversacional — mecanismo que Te'eni e Katz (2026) identificam como fonte recorrente de ruptura. O Quadro 3 organiza os artefatos correspondentes segundo sua dupla função, humana e computacional, e o risco que cada um controla.

Quadro 3. Artefatos de arquitetura e camadas de contexto recuperável

Artefato ou camada	Função para humanos	Função para agentes	Risco controlado
Registro de decisão de arquitetura	Registrar decisão e justificativa	Evitar rediscutir ou violar decisões	Solução incompatível com decisão formal
Mapa de módulos e dependências	Explicar responsabilidades	Direcionar arquivos e dependências permitidas	Alterar componente errado ou criar acoplamento
Contrato de API e de eventos	Coordenar integração	Gerar chamadas e testes compatíveis	Quebrar comportamento existente
Regra arquitetural executável	Preservar estrutura	Permitir portão automático	Erosão silenciosa da arquitetura
Modelo de ameaças	Antecipar abuso	Gerar testes e controles de segurança	Vulnerabilidade introduzida por omissão
Histórico de commits e incidentes	Preservar memória do projeto	Evitar repetir solução já rejeitada	Reintrodução de defeito conhecido
Logs, métricas e runbooks	Operar o sistema	Orientar diagnóstico e ações permitidas	Ignorar condições reais de produção

Fonte: Elaboração do autor (2026), com base em Tao *et al.* (2025) e Te'eni e Katz (2026).

5.2. Execução Controlada, Verificação e Governança

Contexto não é apenas texto. Agentes contemporâneos interagem com bancos de dados, sistemas de arquivos, APIs, repositórios, pipelines e ferramentas de observabilidade, e o Model Context Protocol consolidou-se como protocolo aberto para padronizar essa exposição: servidores fornecem três primitivas — prompts, recursos e ferramentas — e a hierarquia de controle distingue modelos de interação escolhidos pelo usuário, dados administrados pela aplicação e funções que o modelo pode invocar (Model Context Protocol, 2026). A padronização resolve a interoperabilidade, mas não a autorização: o protocolo organiza a chamada, enquanto a política de permissão permanece sob responsabilidade do sistema que o implementa. Daí a necessidade de aplicar aos agentes o princípio do menor privilégio. Um agente encarregado de atualizar documentação não precisa de credenciais de produção; um agente de análise pode operar somente em leitura; alterações destrutivas podem exigir aprovação humana; operações sobre dados sensíveis podem ser proibidas ou intermediadas. A disponibilidade de uma ferramenta não é argumento para seu uso, e validações críticas devem residir fora do modelo, em mecanismos determinísticos que não dependam do texto produzido pelo agente.

A mesma infraestrutura que permite ação permite verificação, e é essa simetria que sustenta a camada seguinte. Ferramentas de compilação, teste, análise estática, consulta a esquemas, execução de benchmarks e leitura de registros podem compor o ciclo do agente, de modo que o modelo deixe de supor que uma solução funciona e passe a obter retorno do ambiente — aproximando o fluxo do paradigma empírico da engenharia: hipótese, intervenção, observação e correção. Organiza-se a verificação em camadas que vão do determinístico ao contextual: checagens rápidas de compilação, tipagem e estilo; testes unitários e de componentes;

integração e contratos; segurança, com análise estática, análise de composição, varredura de segredos e testes de abuso; arquitetura e propriedades não funcionais; e, por fim, revisão humana proporcional ao risco. Nenhuma camada é suficiente isoladamente; em conjunto, reduzem a probabilidade de que uma falha escape por um único mecanismo.

O conjunto formado por especificações, contexto, ferramentas, verificações e políticas que circunda o modelo é aqui denominado harness, termo que a engenharia emprega para a estrutura que conecta, restringe, instrumenta e permite testar um componente. Lopopolo (2026) relata que, em ambiente orientado a agentes, sua equipe passou a investir em legibilidade do repositório, conhecimento versionado, imposição arquitetural, observabilidade e controle de entropia; embora específico, o relato ilustra o conceito como arquitetura de controle da produção automática. O harness não é um arquivo de instruções: compreende o repositório, o pipeline de integração contínua, os testes, os analisadores, as regras de arquitetura, a recuperação de contexto, os controles de acesso, os registros, os critérios de aceite e os pontos de aprovação. Seu ciclo pode ser descrito em oito passos — selecionar a tarefa, construir o pacote de contexto, gerar um plano, validá-lo contra as restrições, executar as alterações, acionar os portões, corrigir falhas com base no retorno e produzir evidências para aprovação. A diferença em relação ao vibe coding não é a ausência de conversação, mas a presença de estado, contratos e critérios externos ao diálogo.

A governança fecha o modelo conectando capacidade técnica a responsabilidade organizacional. Quanto maior a autonomia do agente, maior a necessidade de saber quais modelos foram utilizados, quais dados acessados, quais ferramentas invocadas,

quem aprovou cada mudança e quais evidências sustentaram a implantação. Para cada mudança assistida, a trilha mínima deveria responder a essa cadeia de perguntas, convertendo a autoria difusa do software gerado em responsabilidade de processo. A governança também responde a riscos próprios de agentes, que se somam aos erros tradicionais: injeção de instruções pelo prompt ou pelo próprio contexto recuperado, vazamento de credenciais em registros, execução de comandos destrutivos, risco de cadeia de suprimento por dependências sugeridas e ciclos de correção que expandem a alteração muito além do escopo. Tais riscos exigem isolamento, listas de permissão, sanitização, aprovação, limites de custo e trilhas de auditoria — nada disso implicando rigor uniforme: a governança deve ser proporcional ao risco, classificando tarefas por impacto, reversibilidade, exposição de dados, superfície de segurança e alcance arquitetural, conforme a escala apresentada na seção seguinte.

6. MATURIDADE, HIPÓTESES E PROTOCOLO DE VALIDAÇÃO

Para que o modelo seja utilizável como instrumento de diagnóstico, suas camadas precisam ser traduzidas em uma escala observável. O Quadro 4 propõe uma escala de maturidade de oito níveis, acompanhada da classe de criticidade para a qual cada faixa tende a ser adequada. A escala não implica obrigação de operar sempre no nível mais alto: os níveis iniciais podem ser suficientes em protótipos descartáveis, e o problema surge quando práticas de baixo controle são aplicadas a sistemas cujo risco exige garantias maiores. A maturidade é, portanto, contextual — uma mesma equipe pode operar no nível 3 para gerar scripts internos e no nível 7 para alterar o mecanismo de autenticação.

Quadro 4. Escala de maturidade ESAIA e autonomia proporcional ao risco

Nível	Característica dominante	Controle predominante	Criticidade compatível
0 — Manual	Código sem geração automática	Processo tradicional	Qualquer
1 — Assistência local	Autocompletar e explicações	Revisão humana direta	Baixa a moderada
2 — Prompt para código	Geração por instrução	Teste ad hoc	Baixa
3 — Prompt estruturado	Modelo de instrução com critérios	Testes definidos	Baixa a moderada
4 — Contexto persistente	Documentação e recuperação sobre repositório legível	Proveniência de contexto	Moderada
5 — Arquitetura executável	Decisões, regras e contratos verificáveis	Portões arquiteturais	Moderada a alta
6 — Agentes com ferramentas	Automação com protocolo e APIs	Permissões, registros e limites	Alta
7 — Harness governado	Ciclo autônomo com portões e políticas	Auditoria e autonomia por risco	Alta a crítica

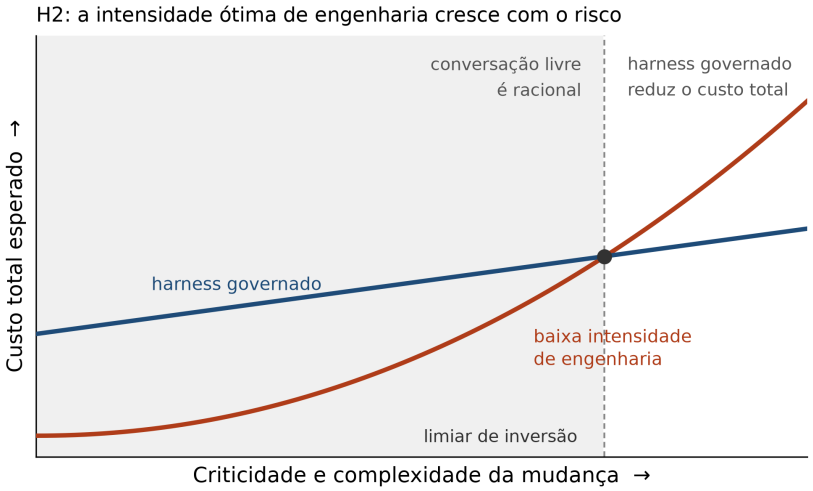
Fonte: Elaboração do autor (2026), em alinhamento com os princípios de desenvolvimento seguro de Souppaya, Scarfone e Dodson (2022).

6.1. Hipóteses e Métricas

O modelo permite formular hipóteses falsificáveis. A hipótese principal (H1) sustenta que maior estruturação do processo aumenta a qualidade global da mudança, mantidos constantes modelo, tarefa e orçamento, e desdobra-se em quatro proposições alinhadas aos eixos: arquitetura executável reduz violações de dependência (H1a); contexto recuperável reduz ciclos de correção por informação ausente (H1b); portões de segurança reduzem vulnerabilidades aceitas (H1c); e governança aumenta a rastreabilidade e a capacidade de diagnóstico pós-incidente (H1d). A segunda hipótese (H2) é de moderação: o efeito do harness é condicionado pela complexidade e pela criticidade da tarefa, de modo que, em tarefas simples e reversíveis, o custo de preparação pode superar o benefício. A terceira (H3) é de medição: a produtividade percebida diverge mais da medida quando o indicador ignora retrabalho e defeitos tardios, proposição sugerida pelo contraste documentado por Becker *et al.* (2025).

A hipótese de moderação pode ser expressa de forma compacta. Propõe-se que a intensidade necessária de engenharia E seja função da complexidade C, da criticidade K, do acoplamento A, do horizonte de manutenção H e do inverso da reversibilidade R, isto é, E = f(C, K, A, H, 1/R). A expressão não pretende oferecer coeficientes universais; organiza variáveis para estudos futuros e rejeita a ideia de um único modo ótimo de trabalhar com inteligência artificial. Sua consequência empírica é a existência de um limiar de inversão, representado na Figura 3, abaixo do qual a conversação livre é a escolha racional e acima do qual o harness governado reduz o custo total esperado.

Figura 3. Representação conceitual da hipótese de moderação (H2)



Fonte: Elaboração do autor (2026). Figura esquemática, sem dados empíricos: as curvas ilustram a relação postulada entre intensidade de engenharia e custo total esperado, e os eixos são qualitativos.

Testar essas hipóteses exige métricas que cubram simultaneamente resultado e custo, evitando que um ganho aparente de velocidade oculte retrabalho ou dívida. O Quadro 5 reúne seis indicadores, com definição operacional e justificativa, escolhidos por serem observáveis em pipelines já existentes.

Quadro 5. Métricas centrais para avaliação do desenvolvimento assistido por inteligência artificial

Métrica	Formulação simplificada	Interpretação
Taxa de correção funcional	critérios aprovados / critérios totais	Correção funcional
Densidade de vulnerabilidades	achados ponderados / KLOC alterado	Exposição de segurança
Taxa de violação arquitetural	violações novas / mudança	Aderência arquitetural
Índice de retrabalho	retrabalho / implementação inicial	Estabilidade da primeira solução

Tempo até aceite	intervalo até a aprovação de todos os portões	Produtividade de fluxo
Custo por mudança aceita	custo total de inteligência artificial / mudanças aceitas	Eficiência econômica

Fonte: Elaboração do autor (2026).

O protocolo de validação explora a propriedade mais útil do conceito de harness: sua manipulabilidade experimental. Propõe-se experimento controlado no qual o mesmo modelo, a mesma versão, o mesmo orçamento e o mesmo conjunto de tarefas são submetidos a quatro regimes cumulativos de engenharia, descritos no Quadro 6. Como as condições diferem apenas no processo, o desenho permite estimar o efeito daquilo que a organização efetivamente controla — qualidade da especificação, cobertura de contexto, regras arquiteturais e rigor de verificação — em lugar de comparar fornecedores. Recomenda-se estratificar a amostra por criticidade e complexidade, condição necessária para testar H2, e registrar as seis métricas do Quadro 5 em cada condição, uma vez que H1 e H3 preveem efeitos em direções distintas nos indicadores de resultado e de custo.

Quadro 6. Condições experimentais propostas para validação do modelo

Condição	**Regime de engenharia**	**Elementos presentes**	**Nível ESAIA aproximado**
G1	Conversação livre	Objetivo e agente	2
G2	Prompt estruturado	Requisitos, critérios de aceite e agente	3

G3	Engenharia de contexto	G2 com arquitetura e recuperação sobre documentação versionada	4 e 5
G4	Harness governado	G3 com ferramentas controladas, portões, registros e políticas	6 e 7

Fonte: Elaboração do autor (2026).

7. DISCUSSÃO

A leitura conjunta das evidências e do modelo sugere que o debate público sobre vibe coding tem sido mal formulado. A pergunta recorrente — a inteligência artificial aumenta ou reduz a produtividade? — admite respostas opostas conforme o indicador escolhido, e é esse grau de liberdade que a torna pouco informativa. Reformulada em termos de engenharia, a pergunta passa a ser: sob quais condições de processo a delegação de implementação produz mudanças aceitáveis a um custo justificável? Nessa formulação, o experimento da METR deixa de ser um veredicto sobre a tecnologia e torna-se um alerta sobre instrumentos de medição; o SUSVIBES deixa de ser uma condenação dos agentes e torna-se a demonstração de que correção e segurança são propriedades distintas, que exigem mecanismos distintos; e os achados de Te'eni e Katz convertem-se em requisitos de projeto para a camada de contexto.

O antipadrão que este artigo identifica não é, portanto, a conversação com modelos de linguagem, atividade legítima e frequentemente eficiente, mas a transferência de decisões de engenharia para um processo cuja única evidência de qualidade é o

comportamento aparente do artefato. O deslocamento é sutil porque cada passo isolado parece razoável: o código executa, o teste manual passa, a entrega acelera. O custo aparece adiante, sob a forma de dívida arquitetural, vulnerabilidades silenciosas, dificuldade de diagnóstico e perda de rastreabilidade — e é esse descasamento temporal entre benefício percebido e custo incorrido que o conjunto de métricas do Quadro 5 pretende tornar visível.

Três implicações práticas decorrem dessa posição. Para equipes, o caminho de evolução mais produtivo não começa pela adoção de agentes autônomos, mas pela preparação do repositório: documentação versionada e recuperável, regras arquiteturais executáveis, testes que funcionem como especificação e portões de segurança automatizados — condições que beneficiam o trabalho humano independentemente da automação, o que reduz o risco do investimento. Para a gestão, a consequência é substituir métricas de atividade por métricas de aceite, já que volume de código gerado é indicador de produção, não de valor. Para a pesquisa, a oportunidade está em tratar o processo como variável, conforme o protocolo de quatro condições.

Importa delimitar também onde o vibe coding é a escolha racional. Em protótipos descartáveis, provas de conceito, scripts de análise pontual e automações internas de baixo impacto, o custo de preparar especificação, contexto e verificação pode exceder o valor da garantia obtida. A proposta deste artigo não é eliminar o regime conversacional, mas tornar explícita a fronteira: se o artefato sobrevive ao experimento, entra em produção, toca dados de terceiros ou será mantido por outras pessoas, ele cruzou o limiar de inversão representado na Figura 3, e a intensidade de engenharia precisa acompanhá-lo.

7.1. Limitações e Ameaças à Validade

Quatro limitações devem ser consideradas. A primeira é de desenho: tratando-se de revisão integrativa, não há estimativa agregada de efeito nem triagem sistemática com dupla revisão independente, de modo que a seleção de fontes permanece sujeita a viés do revisor. A segunda é de validade temporal: parte das evidências quantitativas depende do estado das ferramentas no momento da coleta, como a própria METR reconheceu ao revisar seu desenho, e números específicos envelhecem mais rápido que os mecanismos que revelam — razão pela qual o argumento foi construído sobre os mecanismos. A terceira é de generalização: os estudos qualitativos aqui mobilizados derivam de amostras pequenas e contextos específicos, e seus percentuais devem ser lidos como prevalências temáticas em um caso, não como estimativas populacionais. A quarta é a mais relevante: o framework não foi validado empiricamente, constituindo construção teórica coerente com evidências convergentes cujo valor dependerá dos resultados do protocolo da Seção 6. Cabe antecipar, ainda, uma ameaça à validade do próprio protocolo: como as condições G1 a G4 diferem em custo de preparação, comparações que ignorem esse custo favoreceriam artificialmente os regimes mais estruturados, o que torna o custo por mudança aceita e o tempo até aceite parte do desenho, e não refinamento opcional.

8. CONSIDERAÇÕES FINAIS

Este artigo partiu da constatação de que a disseminação de modelos capazes de gerar código deslocou o gargalo do desenvolvimento da escrita para o julgamento, e propôs um modelo para reconstruir, nesse arranjo, as funções que a engenharia de

software sempre exerceu. A síntese das evidências de 2022 a 2026 mostrou que a percepção de produtividade pode divergir do efeito medido, que a correção funcional é indicador insuficiente de qualidade e que atributos não funcionais e a sustentação de contexto resistem ao controle por prompting. Essas regularidades sustentam o argumento central: a variável explicativa relevante não é a fluência do modelo, mas a densidade de mecanismos verificáveis que o circundam.

A contribuição consistiu em formalizar esses mecanismos no ESAIA, articulado em sete camadas e quatro eixos, e em torná-lo verificável por meio de uma escala de maturidade, seis métricas que cobrem resultado e custo e um protocolo de quatro condições que mantém modelo, tarefa e orçamento constantes. O modelo é independente de fornecedor e condicional: a intensidade de engenharia recomendada cresce com a criticidade, a complexidade, o acoplamento, o horizonte de manutenção e a irreversibilidade da mudança, de modo que a conversação livre permanece racional abaixo do limiar de inversão.

Como se trata de proposta teórica, a agenda de continuidade é clara: executar o protocolo de quatro condições em contextos industriais reais, com estratificação por criticidade; estimar empiricamente o limiar de inversão previsto por H2; validar as métricas quanto à confiabilidade e à sensibilidade; e investigar a qualidade do contexto recuperado como variável independente, uma vez que a presença de recuperação não assegura pertinência. Conclui-se que o problema não está em delegar implementação à inteligência artificial, mas em fazê-lo sem os mecanismos explícitos que tornam a produção verificável, rastreável e compatível com requisitos de qualidade. A engenharia de software não se tornou obsoleta com a

automação da escrita de código; tornou-se a condição de possibilidade de seu uso responsável.

REFERÊNCIAS BIBLIOGRÁFICAS

BECKER, Joel; RUSH, Nate; BARNES, Elizabeth; REIN, David. Measuring the impact of early-2025 AI on experienced open-source developer productivity. **arXiv,** Ithaca, 10 jul. 2025. Pré-print. DOI: 10.48550/arXiv.2507.09089. Disponível em: https://arxiv.org/abs/2507.09089. Acesso em: 1 out. 2026.

DEBELLIS, Derek *et al.* **State of AI-assisted software development 2025.** [S. l.]: DORA; Google Cloud, 2025. Relatório setorial. Disponível em: https://dora.dev/research/2025/dora-report/. Acesso em: 1 out. 2026.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION; INTERNATIONAL ELECTROTECHNICAL COMMISSION. **ISO/IEC 25010:2023: systems and software engineering — systems and software quality requirements and evaluation (SQuaRE) — product quality model.** Geneva: ISO, 2023. Disponível em: https://www.iso.org/standard/78176.html. Acesso em: 1 out. 2026.

JIMENEZ, Carlos E. *et al.* SWE-bench: can language models resolve real-world GitHub issues? In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 12., 2024, Viena. **Proceedings [...].** [S. l.]: ICLR, 2024. Trabalho de anais, utilizado como referência de coerência. Disponível em: https://proceedings.iclr.cc/paper_files/paper/2024/hash/edac78c3e300629acfe6cbe9ca88fb84-Abstract-Conference.html. Acesso em: 1 out. 2026.

LOPOPOLO, Ryan. Harness engineering: leveraging Codex in an agent-first world. **OpenAI Engineering,** San Francisco, 11 fev. 2026. Relato técnico. Disponível em: https://openai.com/index/harness-engineering/. Acesso em: 1 out. 2026.

METR. We are changing our developer productivity experiment design. **METR Blog,** Berkeley, 24 fev. 2026. Disponível em: https://metr.substack.com/p/2026-02-24-uplift-update. Acesso em: 1 out. 2026.

MODEL CONTEXT PROTOCOL. **Specification: overview of prompts, resources and tools.** [S. l.], 2026. Especificação técnica. Disponível em: https://modelcontextprotocol.io/specification/. Acesso em: 1 out. 2026.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Secure software development practices for generative AI and dual-use foundation models: an SSDF community profile.** Gaithersburg: NIST, 2024. NIST SP 800-218A. DOI: 10.6028/NIST.SP.800-218A. Disponível em: https://doi.org/10.6028/NIST.SP.800-218A. Acesso em: 1 out. 2026.

_____. **Secure Software Development Framework (SSDF) version 1.2: recommendations for mitigating the risk of software vulnerabilities.** Initial public draft. Gaithersburg: NIST, 17 dez. 2025. NIST SP 800-218r1 ipd. DOI: 10.6028/NIST.SP.800-218r1.ipd. Disponível em: https://doi.org/10.6028/NIST.SP.800-218r1.ipd. Acesso em: 1 out. 2026.

NOGUEIRA, Rodrigo Pato; VIEIRA, Marco; CAMPOS, João R. PROBE: benchmarking code generation in large language models. **Empirical Software Engineering,** Dordrecht, v. 32, n. 1, art. 25, 2026.

DOI: 10.1007/s10664-026-10904-5. Disponível em: https://doi.org/10.1007/s10664-026-10904-5. Acesso em: 1 out. 2026.

SARKAR, Advait; DROSOS, Ian. Vibe coding: programming through conversation with artificial intelligence. In: ANNUAL CONFERENCE OF THE PSYCHOLOGY OF PROGRAMMING INTEREST GROUP, 36., 2025. **Proceedings [...].** [S. l.]: PPIG, 2025. Trabalho de anais, utilizado como referência de coerência. Disponível em: https://www.microsoft.com/en-us/research/publication/vibe-coding-programming-through-conversation-with-artificial-intelligence/. Acesso em: 1 out. 2026.

SOMMERVILLE, Ian. **Software engineering.** 10. ed. Harlow: Pearson Education, 2016. Disponível em: https://elibrary.pearson.de/book/99.150005/9781292096148. Acesso em: 1 out. 2026.

SOUPPAYA, Murugiah; SCARFONE, Karen; DODSON, Donna. **Secure Software Development Framework (SSDF) version 1.1: recommendations for mitigating the risk of software vulnerabilities.** Gaithersburg: NIST, 2022. NIST SP 800-218. DOI: 10.6028/NIST.SP.800-218. Disponível em: https://doi.org/10.6028/NIST.SP.800-218. Acesso em: 1 out. 2026.

SUN, Xin; STÅHL, Daniel; SANDAHL, Kristian; KESSLER, Christoph. Quality assurance of LLM-generated code: addressing non-functional quality characteristics. **Journal of Systems and Software,** Amsterdam, v. 238, art. 112885, 2026. DOI: 10.1016/j.jss.2026.112885. Disponível em: https://doi.org/10.1016/j.jss.2026.112885. Acesso em: 1 out. 2026.

TAO, Yicheng; LI, Yuante; QIN, Yao; LIU, Yepang. Retrieval-augmented code generation: a survey with focus on repository-level approaches. **arXiv,** Ithaca, 2025. Pré-print revisado em 2026. Disponível em: https://arxiv.org/abs/2510.04905. Acesso em: 1 out. 2026.

TE'ENI, Dov; KATZ, Adi. Reliable vibe coding: the human-AI context gap in software development. In: INTERNATIONAL CONFERENCE ON HYBRID HUMAN-ARTIFICIAL INTELLIGENCE, 5., 2026. **Proceedings [...].** Amsterdam: IOS Press, 2026. (Frontiers in Artificial Intelligence and Applications). Trabalho de anais, utilizado como referência de coerência. DOI: 10.3233/FAIA260505. Disponível em: https://doi.org/10.3233/FAIA260505. Acesso em: 1 out. 2026.

ZHAO, Songwen; WANG, Danqing; ZHANG, Kexun; LUO, Jiaxuan; LI, Zhuo; LI, Lei. Is vibe coding safe? Benchmarking vulnerability of agent-generated code in real-world tasks. In: INTERNATIONAL CONFERENCE ON MACHINE LEARNING, 43., 2026. **Proceedings of Machine Learning Research,** v. 306, 2026. Trabalho de anais, utilizado como referência de coerência. Disponível em: https://proceedings.mlr.press/v306/zhao26ax.html. Acesso em: 1 out. 2026.

---

[1] Doutor em Filosofia pela Università degli Studi di Ferrara, Doutorando em Filosofia pela Pontifícia Universidade Católica do Paraná (PUC-PR), Doutorando em Ensino de Ciências e Tecnologia pela Universidade Tecnológica Federal do Paraná, Docente da Universidade Cesumar de Maringá (Unicesumar). E-mail: acesse o artigo original para visualizar o e-mail. Lattes: http://lattes.cnpq.br/8447571087123317. ORCID: https://orcid.org/0000-0003-0656-2226.

[2] Doutoranda em Ensino de Ciências e Tecnologia pela Universidade Tecnológica Federal do Paraná, Docente da Universidade Cesumar de Maringá (Unicesumar) e do Núcleo Regional. E-mail: acesse o artigo original para visualizar o e-mail. Lattes: http://lattes.cnpq.br/5453204090442136. ORCID: https://orcid.org/0000-0001-7178-1420.

[3] Mestre em Ciência da Computação pela Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), Coordenador de TI do Centro Universitário Cidade Verde (Unicive). E-mail: acesse o artigo original para visualizar o e-mail. Lattes: http://lattes.cnpq.br/5276614823227153.Orcid: https://orcid.org/0009-0005-6128-9930.

[4] Especialista em Arquitetura de Software, Especialista em Tecnologias Educacionais Para Sala de Aula e User Experience (UX) e User Iterface (UI) pelo Instituto Facuminas, Graduação em Engenharia da Computação pelo Centro Universitário Ingá (UNINGA). E-mail: acesse o artigo original para visualizar o e-mail. Lattes: http://lattes.cnpq.br/7965166942454275. ORCID: https://orcid.org/0009-0001-6526-2752.