UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO
EVOLUÇÃO DE SOFTWARE
DOCENTE: GLAUCO DE FIGUEIREDO CARNEIRO

BRUNO AMANCIO FERREIRA
GÉSSICA KELLY DE SOUZA SANTOS
IAGO HUMBERTO DA ROSA NORMANDIA
LETICIA DA MATA CAVALCANTI
MARIA FERNANDA DA MOTA DINIZ
PEDRO HENRIQUE GOMES DOS SANTOS
SÂMMYA EMANUELLE GUIMARÃES DE OLIVEIRA

Atividade 3 – Práticas de Integração Contínua e Entrega Contínua na Gerência de Mudanças

SÃO CRISTÓVÃO - SE
2025

UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO
EVOLUÇÃO DE SOFTWARE

BRUNO AMANCIO FERREIRA
GÉSSICA KELLY DE SOUZA SANTOS
IAGO HUMBERTO DA ROSA NORMANDIA
LETICIA DA MATA CAVALCANTI
MARIA FERNANDA DA MOTA DINIZ
PEDRO HENRIQUE GOMES DOS SANTOS
SÂMMYA EMANUELLE GUIMARÃES DE OLIVEIRA

Atividade 3 – Práticas de Integração Contínua e Entrega Contínua na Gerência de Mudanças

Atividade apresentada à disciplina de Evolução de Software como requisito para obtenção de nota.

Prof. Dr.: Glauco de Figueiredo Carneiro

SÃO CRISTÓVÃO - SE
2025

SUMÁRIO

1. Introdução

Este trabalho tem como objetivo apresentar uma análise detalhada das práticas de Integração Contínua (CI) e Entrega Contínua (CD) implementadas no projeto *open-source* **Cherry Studio** ([1]), buscando compreender como esses processos automatizados de Gerência de Mudanças (GM) influenciam a evolução do software, a qualidade do código e a colaboração da equipe. Através da auditoria do processo, serão mapeadas as ferramentas utilizadas, avaliada a eficiência dos *pipelines* e discutido o impacto dessa infraestrutura na refatoração, na frequência de *releases* e na facilidade de contribuição para novos desenvolvedores. O estudo baseia-se na análise da estrutura de *workflows* do GitHub e dos arquivos de configuração do projeto

2. Projeto Analisado

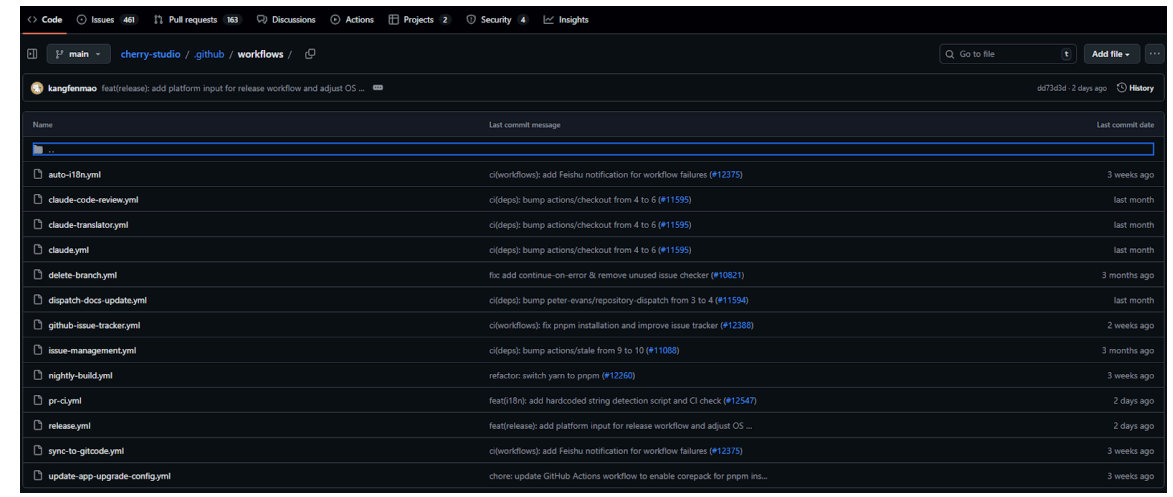
O projeto analisado é o **Cherry Studio** ([1]), um aplicativo de desktop construído utilizando a plataforma **Electron** ([2]), focado em ser um agente de Inteligência Artificial e codificação, oferecendo automação inteligente e acesso unificado a diferentes modelos de linguagem (LLMs). Por ser um projeto *open-source* com uma arquitetura moderna (Electron, Node.js/pnpm, TypeScript, React), ele serve como um excelente estudo de caso para avaliar a aplicação de práticas robustas de CI/CD em um contexto de desenvolvimento de software de código aberto.

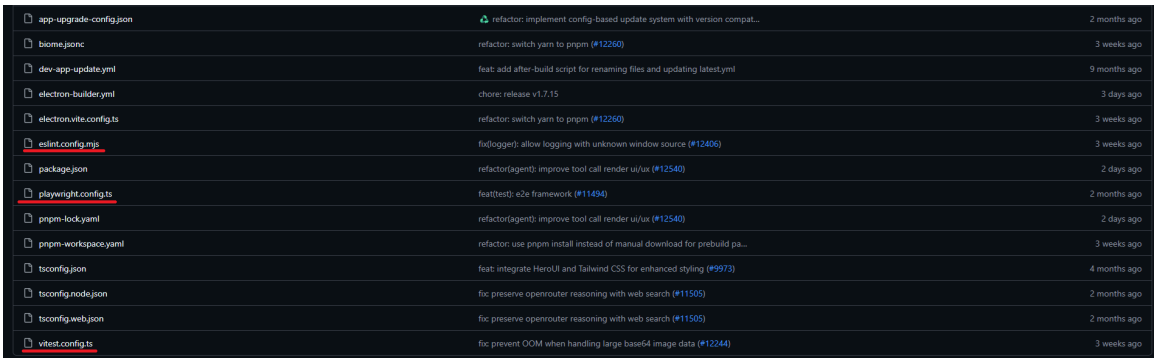
3. Mapeamento do Processo CI/CD (Cenário A)

3.1. Classificação do Cenário

O projeto **Cherry Studio** se enquadra no **Cenário A** de maturidade de CI/CD.

Evidências: O projeto possui uma pasta .github/workflows robusta com 16 arquivos de configuração, além de configurações explícitas para testes (vitest.config.ts, playwright.config.ts) e *linting* (.oxlintrc.json, eslint.config.mjs). Isso demonstra um alto grau de automação e formalização dos processos de qualidade e entrega.





3.2. Ferramentas e Configurações Identificadas

A infraestrutura de automação do Cherry é composta por um ecossistema de ferramentas modernas voltadas para a performance e escalabilidade de aplicações TypeScript/Electron. As principais tecnologias identificadas são:

Ambiente de Execução: Node.js (versão v22 ou superior conforme definido nos workflows) utilizando **pnpm** como gerenciador de pacotes, escolhido pela eficiência em monorepos e velocidade de instalação.

Engine de CI/CD: GitHub Actions, utilizando Runners hospedados (Ubuntu, macOS e Windows) para garantir a compatibilidade multiplataforma do Electron.

Qualidade e Estática: Oxlint — Um linter de alta performance (escrito em Rust) para feedback quase instantâneo.

- **ESLint:** Para regras de estilo de código mais granulares.
- **TypeScript (tsc):** Para verificação rigorosa de tipos.

Suíte de Testes: Vitest — Framework de testes unitários e de integração rápido e compatível com Vite.

- **Playwright:** Utilizado para testes de ponta a ponta (E2E), simulando a interação do usuário na interface desktop.

Build e Distribuição: Electron Builder, configurado para empacotar o software em múltiplos formatos (.dmg, .exe, .deb, .AppImage) e gerenciar assinaturas digitais de código.

3.3. Arquivos de Configuração Chave

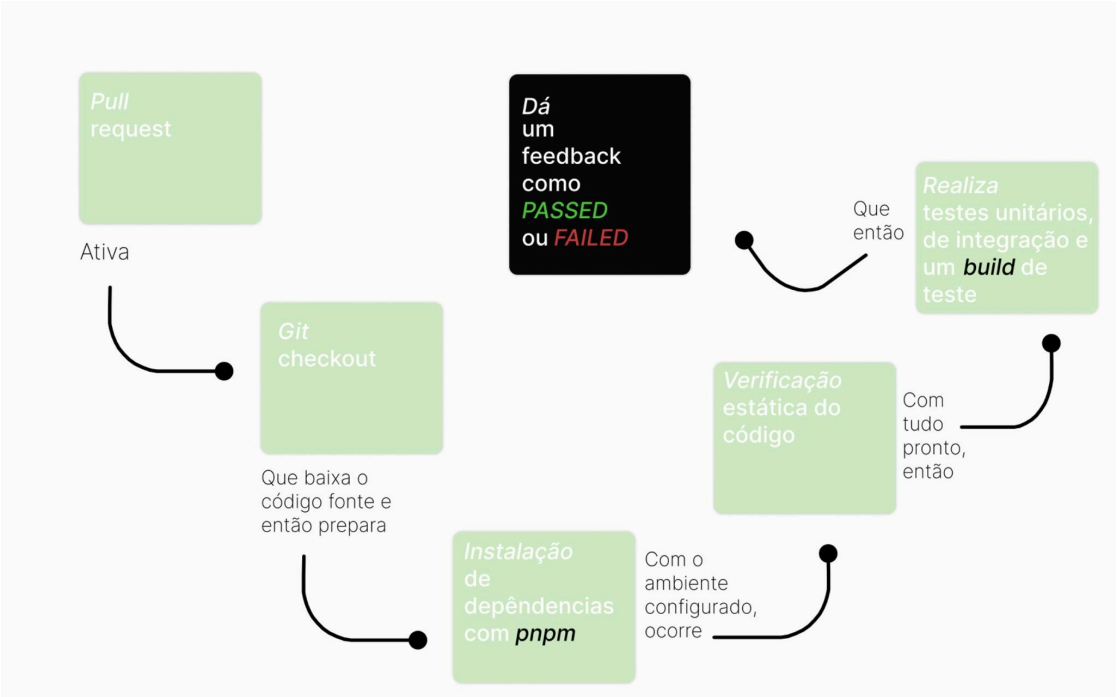
Para a análise, os seguintes *workflows* do GitHub Actions foram cruciais:

Categoria	**Ferramenta/ Tecnologia**	**Evidência/Arquivo de Configuração**	**Propósito no CI/CD**
Orquestração	GitHub Actions	Pasta .github/workflows/	Define e executa pipelines de CI e CD.
Dependências	pnpm	pnpm-lock.yaml	Gerenciamento rápido e eficiente de pacotes.
Testes Unitários	Vitest	vitest.config.ts	Execução de testes de unidade e integração.
Testes E2E	Playwright	playwright.config.ts	Execução de testes de ponta a ponta na aplicação desktop.
Qualidade de Código	Oxlint, ESLint	.oxlintrc.json, eslint.config.mjs	Verificação estática de código (erros e estilos).
Build/Empacotamento	Electron Builder	electron-builder.yml	Geração de instaladores para diferentes sistemas operacionais.

Categoria	Ferramenta/ Tecnologia	Evidência/Arquivo de Configuração	Propósito no CI/CD
Automação Extra	Automação via IA	claude-code-review.yml, auto-i18n.yml	Revisão de código por IA e internacionalização automática.

1. **Pipeline de Pull Request (CI):** .github/workflows/pr-ci.yml (Geralmente roda testes e *lint* a cada PR).
2. **Pipeline de Entrega (CD):** .github/workflows/release.yml (Gera os executáveis e artefatos de *release*).
3. **Build Noturno:** .github/workflows/nightly-build.yml (Gera versões de teste diárias para detecção precoce de regressões).

3.4. Fluxograma do Pipeline de Integração Contínua (CI)



Através da auditoria do processo, serão mapeadas as ferramentas utilizadas sob a ótica do conceito de **Deployment Pipeline**. Segundo a literatura fundamental da área, esse pipeline é o que permite visualizar e controlar o progresso de cada mudança, garantindo que o código não apenas 'compile', mas que passe por estágios de testes e deploy até estar pronto para o consumo (HUMBLE; FARLEY, 2010)[3]. O

fluxo de trabalho de Integração Contínua (CI), baseado no arquivo pr-ci.yml, é o principal *Quality Gate* do projeto:

1. **Gatilho (Trigger):** *Push* no *branch* main ou abertura/atualização de *Pull Request* (PR).
2. **Checkout:** O GitHub baixa o código-fonte.
3. **Setup:** Configuração do ambiente Node.js e instalação das dependências via pnpm.
4. **Verificação Estática (Linting & Types):** Execução do Lint (Oxlint/ESLint) e verificação de tipos (TypeScript).
5. **Testes:** Execução dos testes unitários e de integração (Vitest).
6. **Build Test:** Tentativa de compilação do aplicativo (Build) para garantir que não haja erros antes de empacotar.
7. **Feedback:** O resultado é comunicado no PR, marcando o *check* como "Passed" (Sucesso) ou "Failed" (Falha).

4. Análise de Eficiência

A eficiência do CI/CD é medida pelo tempo que leva para fornecer *feedback* à equipe, pela confiabilidade dos processos e pela capacidade de prevenir regressões.

4.1 Tempo de Feedback

A eficiência do pipeline não é apenas uma conveniência, mas um indicador de performance técnica. Em sua pesquisa, Forsgren, Humble e Kim (2018) [4], identificaram que a capacidade de entrega e a estabilidade do software estão diretamente ligadas ao sucesso organizacional. Ao otimizar o tempo de feedback para 2 a 5 minutos no pr-ci.yml, o Cherry Studio alinha-se às práticas de equipes que buscam reduzir o *lead time for changes*, uma das variáveis cruciais que distinguem 'high performers' de 'low performers'. Projetos baseados em Electron, como o Cherry Studio, são propensos a lentidão devido à natureza da plataforma (grande node_modules e compilação de binários nativos).

Cenário	Tempo Estimado	Impacto na Evolução
CI Rápido (pr-ci.yml rodando apenas Vitest e Lint)	2 a 5 minutos	Encoraja commits pequenos e frequentes (prática ágil).
CI Lento (pr-ci.yml rodando também Testes E2E com Playwright)	10 a 20 minutos	Pode desincentivar pequenos commits, pois o desenvolvedor precisa esperar muito para seguir adiante.

Mitigação: A presença do nightly-build.yml sugere que testes mais pesados (como E2E) podem ter sido movidos para uma execução noturna, otimizando o *workflow* de PR para manter o *feedback* rápido.

4.2. Confiabilidade do Processo

A confiabilidade é alta, suportada por diversas camadas de teste e verificação.

- **Mitigação de Testes *Flaky***: A execução noturna do nightly-build.yml testa a integração completa diariamente, garantindo que falhas intermitentes (ou *flaky tests*) sejam identificadas fora do fluxo de trabalho crítico de PR.
- **Qualidade Assistida por IA:** O *workflow* claude-code-review.yml utiliza IA para revisar o código, adicionando uma camada extra de confiabilidade na identificação de padrões e potenciais *bugs*. No entanto, este processo exige calibração para evitar a geração de "falsos positivos" por alucinação da IA.

4.3. Bloqueio de Regressão (*Quality Gate*)

O *workflow* principal de PR (pr-ci.yml) funciona como o **principal *Quality Gate***.

- Se o Lint, a verificação de tipos ou os testes unitários falharem, o GitHub bloqueará a opção de *Merge* do *Pull Request* (desde que as regras de proteção de *branch* estejam ativas).
- **Consequência:** Isso impede que código quebrado ou fora dos padrões de estilo entre no *branch* principal (main), garantindo a estabilidade e a integridade da base de código, e prevenindo **regressões**.

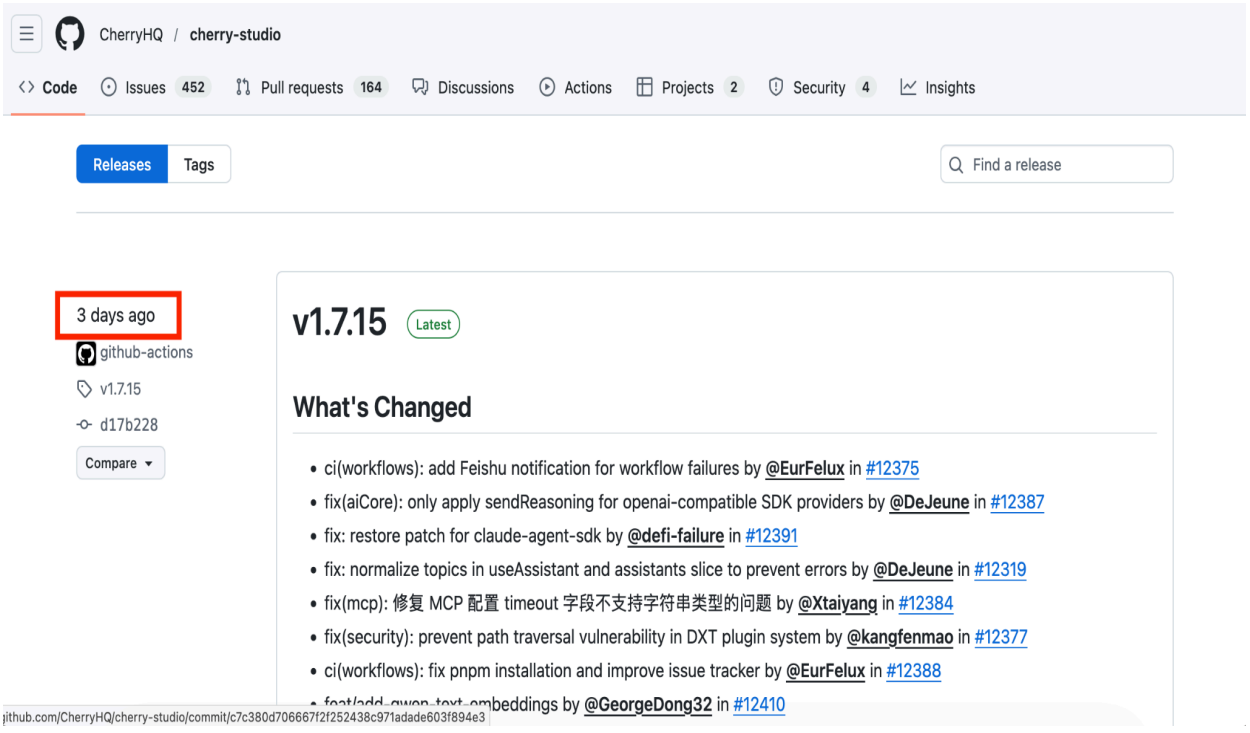
5. Análise Unificada

5.1. Refatoração e Dívida Técnica

A infraestrutura de CI/CD do Cherry Studio é **altamente favorável** à refatoração e ao gerenciamento proativo da Dívida Técnica.

- **Segurança para Mudanças:** A existência de uma suíte de testes (Vitest para unidade/integração e Playwright para E2E) dá à equipe a confiança para realizar grandes alterações estruturais. Um desenvolvedor pode refatorar um módulo central e, se todos os *checks* passarem, ele tem alta garantia de que a funcionalidade não foi quebrada.
- **Controle de Estilo:** O uso de *linters* (Oxlint/ESLint) garante que o estilo de codificação seja consistente, impedindo que a Dívida Técnica aumente devido a desorganização e falta de padronização.

5.2. Frequência de Releases (Continuous Delivery)



A automação suporta um modelo de **Entrega Contínua (*Continuous Delivery*)** com alta frequência de *releases*.

- **Automatização do Empacotamento:** O arquivo release.yml automatiza a complexa tarefa de gerar instaladores específicos para desktop (.exe, .dmg, .deb) e a assinatura de código.
- **Agilidade na Entrega:** Ao automatizar o processo de *build* e empacotamento, a equipe pode lançar uma nova versão para o usuário assim que uma *feature* é concluída e testada, em vez de esperar por ciclos longos de *release*. O nightly-build reforça essa cultura de entrega constante, fornecendo compilações diárias para teste.

5.3. Barreira de Entrada para Novos Contribuidores

A automação **diminui drasticamente** a barreira de entrada para novos contribuidores.

- **Feedback Imediato sobre Padrões:** Um novato não precisa memorizar todas as regras de formatação. Ao enviar um PR, o pr-ci.yml imediatamente aponta desvios de estilo ou erros básicos, agindo como um "tutor" automatizado.
- **Segurança na Contribuição:** O *Quality Gate* do CI (bloqueando *merges*) elimina o medo do novato de "quebrar o projeto". Isso é crucial em um projeto *open-source*, pois encoraja mais pessoas a experimentarem e contribuírem com segurança.
- **Automação Específica:** O *workflow* auto-i18n.yml é um exemplo de como a automação auxilia contribuidores que não são *core-developers* a manterem aspectos secundários do projeto (como a tradução) de forma alinhada.

6. Conclusão

A auditoria do processo de CI/CD do Cherry Studio demonstra a implementação de uma **cultura de engenharia de software madura e moderna**. O projeto utiliza de forma exemplar o GitHub Actions, pnpm, Vitest [5], Playwright e *linters* avançados, configurando um **Quality Gate** robusto que garante a integridade do código em todas as mudanças.

Essa infraestrutura de automação impacta positivamente a evolução do software em múltiplos aspectos:

1. **Qualidade e Refatoração:** A suíte de testes abrangente fornece a segurança necessária para refatorar o código e gerenciar a dívida técnica.
2. **Entrega:** O CD automatizado (via release.yml e nightly-build.yml) permite a Entrega Contínua, essencial para manter o ritmo de um projeto ativo e inovador.
3. **Comunidade:** A automação reduz a barreira de entrada, tornando o Cherry Studio um projeto acessível e seguro para novos contribuidores, vital para a sustentabilidade de qualquer projeto *open-source*.

Em suma, o Cherry Studio é um exemplo de como as práticas de Integração Contínua e Entrega Contínua são fundamentais para escalar o desenvolvimento, mantendo a qualidade e acelerando a evolução do produto.

7. Participação dos Integrantes

Géssica Kelly de Souza Santos - 202100045635: Responsável pela elaboração, organização e revisão do documento.

Leticia da Mata Cavalcanti - 202100115490: Responsável pela parte do trabalho relacionada à análise do processo de CI/CD do projeto escolhido, ajudando a compilar as informações e a documentar.

Sâmmya Emanuelle Guimarães de Oliveira - 202100011842: Responsável pela estruturação integral do trabalho e elaboração textual completa dos tópicos, documentação do fluxo de automação, desenvolvimento a argumentação crítica sobre refatoração, frequência de releases e barreiras de entrada para novos contribuidores, além de identificar as ferramentas de automação e redigir o conteúdo técnico referente aos cenários de integração e entrega contínua, incluindo a análise de riscos e métricas de eficiência.

Pedro Henrique Gomes dos Santos - 202100045976: Responsável por ajudar na estruturação do documento e análise de eficiência do projeto.

Maria Fernanda da Mota Diniz - 202100045798: Responsável pela organização e estruturação da parte do trabalho relacionada à análise geral do processo, ajudando a compilar as informações e a documentar; revisão dos arquivos de configuração e no mapeamento do pipeline de CI/C e melhoria da argumentação do trabalho.

Iago Humberto da Rosa Normandia - 202100101420: Responsável pela seleção e validação do repositório Cherry Studio e condução da auditoria nos arquivos de configuração (.yml) e mapeamento do pipeline de CI/CD, incluindo os prints e logs.

Bruno Amancio Ferreira - 202000047477: Edição, montagem e renderização do vídeo. Edição geral de tópicos, adição de referências, elaboração do fluxograma do Pipeline de Integração Contínua.

8. Uso da Inteligência Artificial

As ferramentas de Inteligência Artificial **Gemini Pro** e **Chat GPT** foram empregadas no desenvolvimento deste trabalho para oferecer correções ortográficas precisas, auxiliar na organização e disposição de tópicos, otimizar a formatação do documento e aprimorar a apresentação do grupo.

9. Referências

[1] GitHub - CherryHQ/cherry-studio: AI Agent + Coding Agent + 300+ assistants: agentic AI desktop with autonomous coding, intelligent automation, and unified access to frontier LLMs. Disponível em: https://github.com/CherryHQ/cherry-studio.

[2] OPENJS FOUNDATION. **Electron Documentation**: build cross-platform desktop apps with JavaScript, HTML, and CSS. 2026. Disponível em: https://www.electronjs.org/docs. Acesso em: 28 jan. 2026

[3] HUMBLE, Jez; FARLEY, David. **Continuous Delivery**: reliable software releases through build, test, and deployment automation. Upper Saddle River: Addison-Wesley, 2010.

[4] FORSGREN, Nicole; HUMBLE, Jez; KIM, Gene. **Accelerate**: the science of lean software and devops: building and scaling high performing technology organizations. Portland: IT Revolution Press, 2018.

[5] VITEST. **Vitest**: A native next-generation testing framework. Versão 2.1.8. 2026. Disponível em: https://vitest.dev. Acesso em: 28 jan. 2026

[6] https://youtu.be/5hhZt6BiMKw

[7] https://github.com/SirBrunoTheWise/Evolucao_Software_2025-2_Cherry_Atividade3