UNIVERSIDADE FEDERAL DE SERGIPE

DEPARTAMENTO DE COMPUTAÇÃO

ENGENHARIA DE SOFTWARE 2

ATIVIDADE 3

BRENO HENRIQUE DO CARMO SANTOS

CARLA STEFANY ROBERTA SANTOS

FERNANDA KAROLINY SANTOS SILVA

JOÃO PAULO MENEZES MACHADO

JOÃO VINÍCIUS DE ALMEIDA ARGOLO

JOSÉ ARTHUR CALIXTO DA ROCHA COSTA

VINÍCIUS AZEVEDO PEROBA

WENDEL ALEXSANDER GOMES MENEZES

PROF. DRº GLAUCO DE FIGUEIREDO CARNEIRO

São Cristóvão - SE

2026

CONTRIBUIÇÃO DOS INTEGRANTES:

Nome	Matrícula	Contribuição
Breno Henrique Do Carmo Santos	202200078737	Pesquisa e edição do documento.
Carla Stefany R. Santos	202400060148	Pesquisa e edição do documento.
Fernanda Karoliny Santos Silva	202200092431	Pesquisa e edição do documento.
João Paulo Menezes Machado	202300038743	Pesquisa e edição do documento.
João Vinícius De Almeida Argolo	202200025573	Pesquisa e edição do documento.
José Arthur Calixto Da Rocha Costa	202300038770	Pesquisa e edição do documento.
Vinícius Azevedo Peroba	201900076892	Pesquisa e edição do documento.
Wendel Alexsander Gomes Menezes	202300027740	Pesquisa e edição do documento.

Repositório: Engenharia_Software_2025-2_Anything_llm_atividade3_parte1

Link do vídeo: Apresentação.mp4

Sumário

1. Introdução

A evolução sustentável de um software depende de mecanismos que garantam a integridade do sistema diante de mudanças contínuas. Este trabalho propõe uma auditoria de maturidade em um projeto real sob a perspectiva de DevOps, com foco na implementação de pipelines de Integração Contínua (CI) para mitigação de riscos de regressão.

A investigação pauta-se na análise técnica da infraestrutura de automação existente, utilizando como metodologia a prospecção de arquivos de configuração em repositórios (como workflows do GitHub Actions) e a inspeção do histórico de *Pull Requests*. O estudo visa não apenas implementar ferramentas, mas avaliar o impacto destas práticas na manutenibilidade do sistema e na redução de barreiras para novos contribuidores.

Como resultado, busca-se o mapeamento do fluxo atual de gerenciamento de ciclo de vida do software, identificando gargalos operacionais e riscos inerentes a processos manuais, propondo, por fim, uma solução automatizada que eleve o padrão de confiabilidade do projeto.

2. Fundamentação Teórica

2.1. Integração Contínua (CI)

A Integração Contínua é uma prática de desenvolvimento de software onde os membros de uma equipe integram seu trabalho frequentemente. Cada integração é verificada por um build automatizado, incluindo testes, para detectar erros de integração o mais rápido possível. Essa prática reduz drasticamente problemas de integração e permite que a equipe desenvolva software coeso mais rapidamente.

2.2. DevOps e Automação de Pipeline

O movimento DevOps busca a unificação do desenvolvimento (Dev) e da operação (Ops) de software. A automação é um de seus pilares fundamentais (CAMS - Culture, Automation, Measurement, Sharing). O Pipeline de CI/CD funciona como uma esteira automatizada que valida o código desde o commit até o deploy, garantindo que o software esteja sempre em um estado "pronto para produção".

2.3. Manutenibilidade e Regressão

- Regressão: Refere-se ao surgimento de defeitos em funcionalidades que anteriormente funcionavam corretamente, geralmente após uma alteração no código.

- Manutenibilidade: É a facilidade com que um sistema de software pode ser modificado para corrigir defeitos, melhorar o desempenho ou adaptar-se a novos requisitos. A automação de testes dentro do pipeline é o principal mecanismo para garantir que a manutenibilidade não seja prejudicada pelo crescimento da complexidade do projeto.

2.4. Ferramentas de Orquestração (GitHub Actions)

Plataformas modernas de hospedagem de código oferecem ferramentas de orquestração integradas. O GitHub Actions, por exemplo, permite a criação de fluxos de trabalho (*workflows*) baseados em eventos (como *push* ou *pull request*). Esses fluxos são definidos em arquivos de configuração YAML, que especificam as etapas de compilação, teste e validação de segurança.

2.5. Projeto Analisado

O **Anything LLM** é uma plataforma open source que permite criar assistentes de IA capazes de conversar com usuários e compreender documentos, integrando grandes modelos de linguagem com bases de conhecimento locais. Ela transforma arquivos e textos em dados pesquisáveis, possibilitando que o chatbot responda com base nesses conteúdos e até execute tarefas automatizadas por meio de agentes de IA. Pode ser usada localmente ou via servidor, oferecendo flexibilidade, privacidade e personalização para empresas e desenvolvedores que desejam construir seus próprios sistemas de chat inteligentes.

3. Auditoria

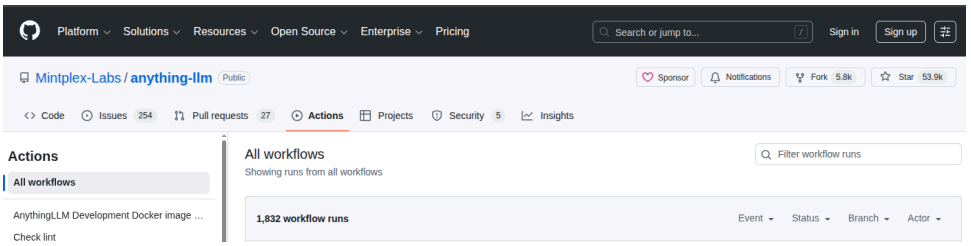
3.1. Diagnóstico de CI/CD e Evidências

3.1.1. Uso de CI/CD

O projeto AnythingLLM faz uso intensivo de práticas de Integração Contínua e Entrega Contínua (CI/CD) por meio da ferramenta GitHub Actions. Essa utilização foi identificada diretamente pela aba “Actions” do repositório, que apresenta um histórico expressivo de execuções automatizadas, com mais de 1.832 workflow runs, evidenciando a adoção contínua e recorrente de automações no ciclo de desenvolvimento do projeto.

A recorrência das execuções demonstra que o CI/CD não é utilizado de forma pontual, mas integrado ao fluxo diário de desenvolvimento e manutenção do software.

Figura 1 – Uso do GitHub Actions e o alto volume de execuções automatizadas.



Fonte: Mintplex-Labs (2025).

3.1.2. Ferramenta utilizada

A ferramenta de CI/CD identificada no projeto é o GitHub Actions, integrada nativamente pelo próprio repositório GitHub. Não foram encontradas evidências de uso de ferramentas externas ou alternativas, como Jenkins, Travis CI ou CircleCI, indicando que toda a automação do projeto está centralizada na infraestrutura oferecida pelo GitHub.

Essa escolha favorece a padronização do processo e reduz a complexidade de integração entre ferramentas distintas.

3.1.3. Localização dos arquivos de configuração

Os arquivos responsáveis pela definição dos pipelines automatizados estão organizados na pasta padrão:

.github/workflows/

Nessa pasta encontram-se arquivos no formato .yml, que descrevem os gatilhos, etapas e ações executadas pelos workflows. A presença dessa estrutura confirma a adoção das boas práticas recomendadas pelo GitHub para configuração de pipelines de CI/CD.

3.1.4. Tipos de workflows existentes

A organização dos workflows pode ser observada diretamente no menu lateral da aba Actions, onde são listados 15 tipos distintos de workflows que são agrupados conforme suas finalidades.

Entre os principais, podemos destacar:

- Check lint: workflow voltado à verificação de estilo e qualidade do código-fonte.

- Check package versions: responsável pela checagem de versões e consistência das dependências do projeto.

- Dependabot Updates: automação dedicada à atualização de dependências, contribuindo para segurança e manutenção do software.

- Run backend tests: execução automatizada de testes no backend, geralmente acionada por *pull requests*.

- Verify translations files: verificação de arquivos de tradução, garantindo consistência em funcionalidades de internacionalização.

- Workflows de build e publicação Docker: pipelines responsáveis pela construção e publicação de imagens Docker, incluindo:

    - Imagens de desenvolvimento;

    - Imagens primárias;

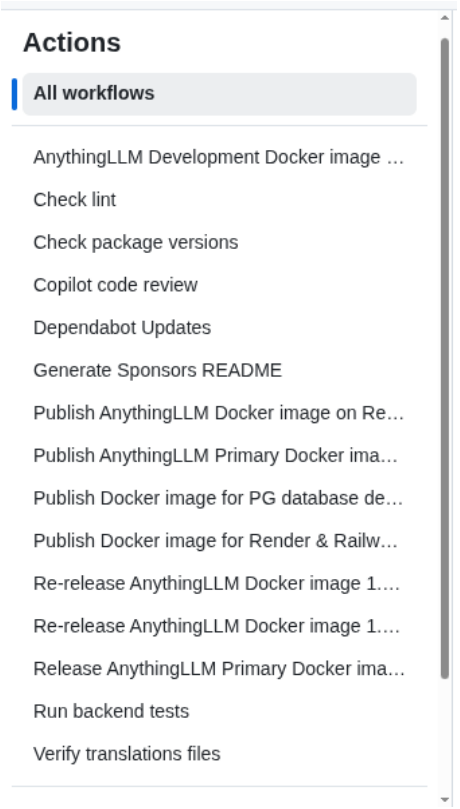
    - Releases e re-releases;

    - Ambientes específicos como PG, Render e Railway.

- Copilot code review: workflow de apoio à revisão automatizada de código.

- Generate Sponsors README: workflow utilitário para geração automática de seções do arquivo README.

Figura 2 – Lista de tipos de workflows do GitHub Actions do projeto AnythingLLM.



Fonte: Mintplex-Labs (2025).

Essa quantidade e diferentes tipos de workflows indica que o projeto utiliza o CI/CD não apenas para validação de código, mas também para geração de artefatos, testes, manutenção de dependências e suporte ao processo de entrega.

3.1.5. Análise de Histórico de Pull Request

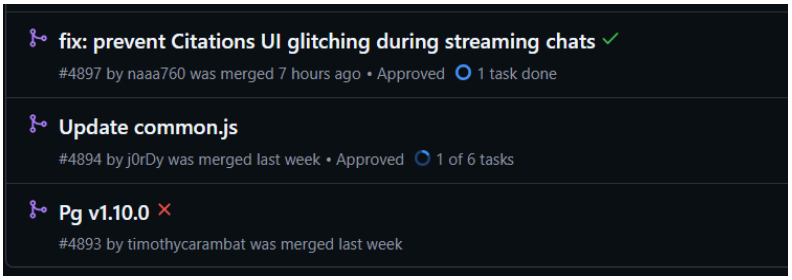
A inspeção das Pull Requests (PRs) no repositório *AnythingLLM* revela como o ciclo de vida do software é governado na prática. Ao analisar PRs recentes observam-se os seguintes padrões que caracterizam a maturidade do projeto:

- Verificação de Status e Gatekeeping Informativo: Observou-se que quase todas as PRs acionam automaticamente o conjunto de checks definidos nos workflows (Lint, Backend Tests, Docker Build). No entanto, um traço característico do projeto é que esses checks funcionam como indicadores informativos e não necessariamente como travas impeditivas (Required Status Checks).

  Evidência: PRs com falhas pontuais que após revisão humana justificando o erro, foram integradas (merged). Isso confirma que a autoridade final no ciclo de vida do

projeto reside na revisão por pares (Peer Review) em detrimento da automação estrita.

Figura 3 – Evidência de Pull Requests aprovadas e mescladas apesar de falhas em workflows de automação.



Fonte: Mintplex-Labs (2025).

3.2. Fluxo Atual, Riscos e Gargalos

3.2.1. Mapeamento de Fluxo Atual

O projeto adota um fluxo de contribuições baseado em Forking Workflow, ou seja, alterações de terceiros são realizadas através de forks do repositório, o que protege a base de código original. Foram identificadas, ainda, grandes distinções de privilégios entre o time de desenvolvimento e desenvolvedores externos, bem como filtros de caminho para a execução de workflows.

Para os desenvolvedores do time, commits podem ser realizados diretamente nas branches do repositório. Após o conclusão do trabalho, uma pull request é aberta, onde workflows relacionados aos arquivos alterados são executados. Pull requests envolvendo o back-end executam rotinas diferentes do front-end. Deploys, caracterizados por merges na branch master disparam rotinas que constroem uma imagem Docker para múltiplas arquiteturas e a versionam em caso de release. Para builds de desenvolvimento, há ausência de automatização e necessidade de reconfiguração manual de branches.

Para desenvolvedores externos, contribuições são possíveis apenas através de forks. O pipeline de testes automatizados disparados a partir de pull requests, contudo, são idênticos, senão pela necessidade de aprovação manual de segurança antes da execução destes. Além disso, verificou-se que workflows de construção de imagem são configurados para não executar em merges de PRs submetidas por esses desenvolvedores, o que previne as imagens de falhas em forks.

Conclui-se que o fluxo de desenvolvimento prioriza a validação humana, uma vez que os mencionados testes automatizados

não são bloqueantes para a efetivação de um merge, mas apenas informativos. É delegada, portanto, a garantia de qualidade às revisões de código.

3.2.2. Riscos de Regressão

A análise dos arquivos de workflow do GitHub Actions e do histórico recente de execuções permite identificar alguns riscos no processo atual de CI/CD do projeto, pois commits são integrados com alta frequência à branch principal e rapidamente seguidos por builds e publicações de imagens Docker, o que indica que falhas não conectadas nos testes automatizados podem chegar a versão distribuída caso não identificadas pela revisão manual. Essa conclusão decorre da observação de pipelines de teste e publicação em curtos intervalos de tempo.

Também foi identificado risco de regressões causadas por dependências externas a partir da estrutura de arquivos YAML responsáveis pelo build. Os workflows mostram reconstruções constantes de imagens Docker e uso de bibliotecas de terceiros sem evidência de validação rígida de versões, o que pode introduzir comportamentos inesperados entre builds se diferentes execuções do pipeline incorporarem versões distintas dessas bibliotecas.

Além disso, há um risco organizacional associado à centralização de permissões e segredos, já que etapas críticas de publicação dependem de credenciais armazenadas no repositório e acessíveis apenas a poucos mantenedores. Essa dinâmica aumenta a dependência de indivíduos específicos para liberação de versões e pode comprometer a continuidade normal do processo em caso de indisponibilidade, além de, possivelmente, sobrecarregar o time interno. A centralização citada pode ser vista pela análise dos arquivos de workflow em etapas como autenticação em registries e publicação de imagens.

3.2.3. Gargalos e Limitações

O principal gargalo reside na ausência de qualidades bloqueantes e na consequente dependência de revisão manual como principal garantia de qualidade. Além de ser limitada pela capacidade humana, o que pode resultar em falhas não detectadas, quando essa abordagem é aplicada em projetos grandes e/ou de alta escalabilidade acaba gerando um gargalo diante das diversas mudanças ocorrendo em paralelo.

Esse problema se agrava devido à centralização de privilégios, o que implica na concentração de decisões críticas aos membros da equipe. Resultando no aumento do custo de

coordenação, redução da autonomia do pipeline automatizado e reforço da dependência de validação humana para assegurar a integridade do sistema.

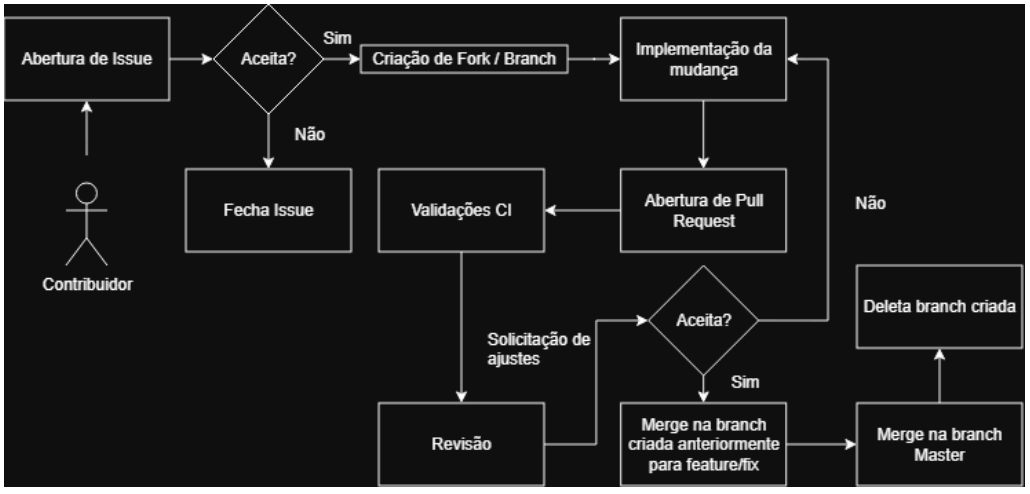
Esse cenário é agravado pela assimetria no tratamento de contribuições internas e externas, especialmente no que se refere à construção e validação de imagens Docker. A não execução de workflows de build para merges provenientes de forks, embora reduza riscos de segurança, reduz a cobertura de validação do artefato final, criando lacunas na verificação de integridade do sistema implantável.

Por fim, a inexistência de automação consistente para builds de desenvolvimento, aliada à necessidade de reconfigurações manuais de branches, impõe um gargalo operacional significativo. Essa dependência de procedimentos manuais compromete a reprodutibilidade do ambiente, eleva o custo de manutenção do pipeline e dificulta a escalabilidade do processo de desenvolvimento.

3.3. Diagramas de Processo e de Pipeline

A figura 4 representa o fluxo atual de contribuição do projeto AnythingLLM, desde a abertura de issues até o merge na branch principal, incorporando validações automáticas e revisão humana.

Figura 4: Diagrama de processos



Fonte: autor

As evidências do processo As-Is foram obtidas a partir da análise do arquivo CONTRIBUTING.md, do histórico de Issues e Pull Requests do repositório, bem como das interações de revisão e validações

automáticas associadas aos PRs.

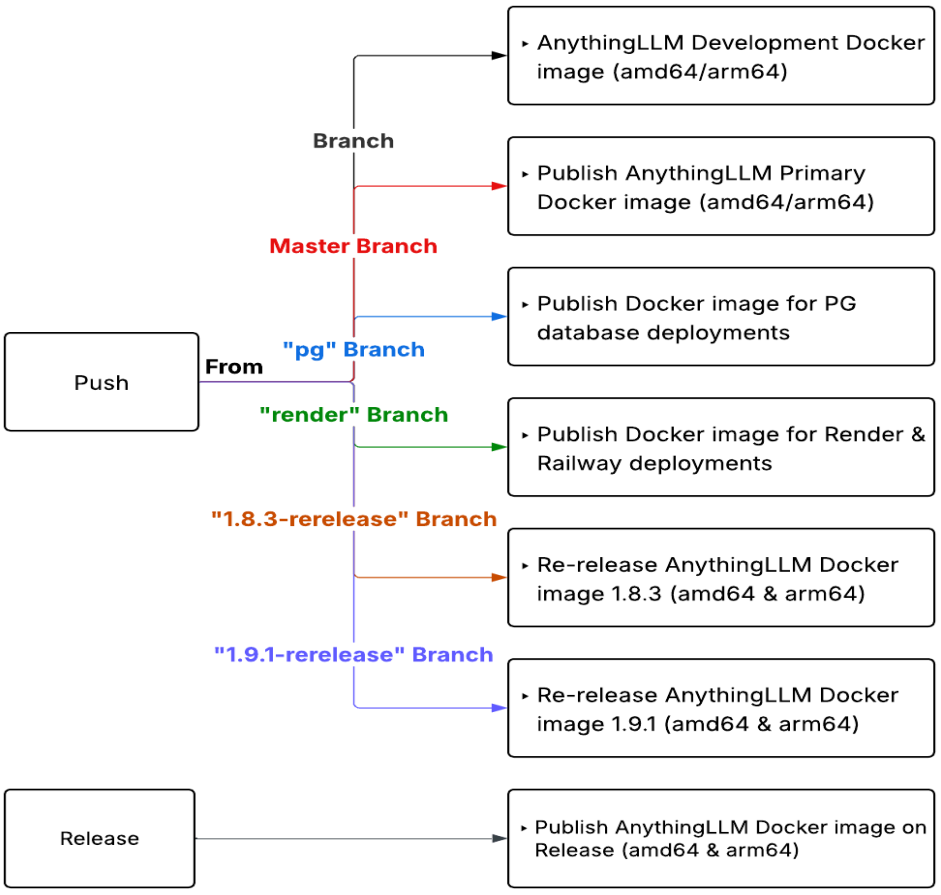
Alguns PRs:

https://github.com/Mintplex-Labs/anything-llm/pull/4897

https://github.com/Mintplex-Labs/anything-llm/pull/4929

A automação com os scripts de workflow atuam em quatro casos distintos, ao realizar um push, ao publicar uma release, programando um dia e horário para acontecer ou dando início a um pull request. As automações em um push ou release envolvem primariamente a publicação de imagens para o Docker do projeto, havendo algumas exceções no caso de um push de alguma branch específica, a qual deve utilizar um script próprio para a publicação da imagem.

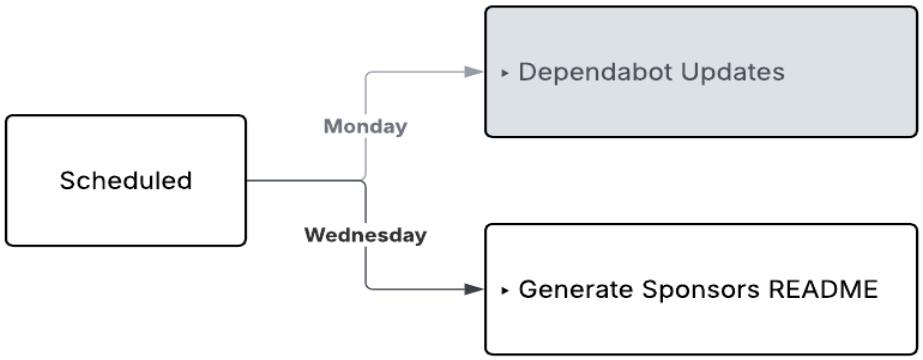
Figura 5: Automações em push e release



Fonte: autor

Para as rotinas programadas, a automatização se dá a partir da atualização da lista de patrocinadores do projeto no arquivo "readme.md", acontecendo semanalmente todas as quartas-feiras, e o Dependabot. O "Dependabot" foi usado como uma ferramenta de atualização automática de dependências no repositório, utilizando pull requests automáticos para evitar a permanência de dependências fracas ou desatualizadas, mas seu uso só foi registrado em três segundas-feiras em Dezembro de 2024.

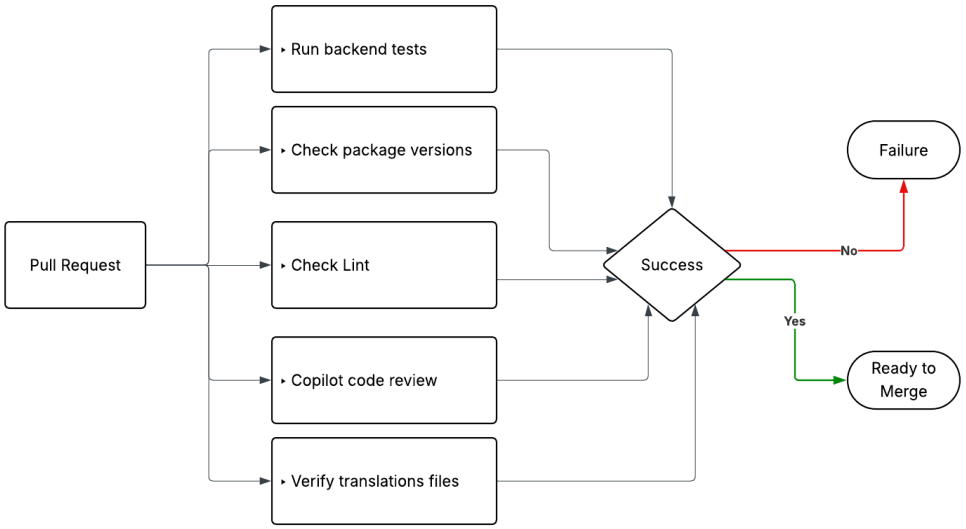
Figura 6: Automações de rotina programada



Fonte: autor

Ao iniciar um pull request, dependendo da natureza e do local dos arquivos modificados, alguns testes podem ser executados sobre as mudanças, envolvendo verificações de tradução, pacotes, backend e código fonte antes que a ação do merge possa ser realizada.

Figura 7: Automação de testes



Fonte: autor

4. Tutorial

- Clonar o repositório

  git clone https://github.com/ArtorioXP/Engenharia_Software_2025-2_Anything_llm_atividade3_parte1

5. Referências

[1] Marco Tulio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, Editora: Independente, 2020.

[2] VALENTE, M. T. Cap. 10: DevOps – Engenharia de Software Moderna. Disponível em: <https://engsoftmoderna.info/cap10.html>.

[3] GITHUB. Entendendo o GitHub Actions. Documentação GitHub (GitHub Docs). Disponível em: https://docs.github.com/pt/actions/get-started/understand-github-actions. Acesso em: 27 jan. 2026.

[4] CARNEIRO, Glauco de Figueiredo. Engenharia de Software II: materiais didáticos. São Cristóvão: Universidade Federal de Sergipe, 2025. Material de aula apresentado à turma de Engenharia de Software II.