UNIVERSIDADE FEDERAL DE SERGIPE
DEPARTAMENTO DE COMPUTAÇÃO

Engenharia de Software II
Glauco de Figueiredo Carneiro

Diagnóstico e Auditoria do Pipeline de Integração Contínua do Projeto Crawn4AI

Carlos Daniel Lima de Gois
Felipe Osni Santos Moura
João Pedro Cardoso Arruda
Nicolas Matheus Ferreira de Jesus
Samuel Bastos Borges Pinho
Vinícius Vasconi Villas Boas Micska
Vitor Leonardo Sena de Lima

SÃO CRISTÓVÃO
27/01/2026

1. Histórico de Atividades

Aluno	Atividade
Felipe Osni Santos Moura - 202100011397	Criação do slide sobre riscos do processo atual.
João Pedro Cardoso Arruda - 202300027418	Contribuiu na parte de mapeamento do fluxo para identificar o workflow de releases evidenciando sua arquitetura e suas automações. Elaborou diagramas de workflow. Contribuiu na busca por evidências. Contribuiu na gravação do vídeo.
Nicolas Matheus Ferreira de Jesus - 202200014444	Criação da introdução e responsável por uma parte da metodologia de análise e busca por evidências Criação de parte do slide
Samuel Bastos Borges Pinho - 202300083945	Análise manual do repositório e de Pull Requests; Adição de informações no documento referente à riscos e gargalos identificados, e captura de tela do fluxograma da automatização de release (CD); Gravação de vídeo discorrendo sobre as evidências encontradas e debatendo os resultados; Criação de parte do slide.
Vinícius Vasconi Villas Boas Micska -	Discussão sobre o funcionamento do github

202300038940	actions para justificar os resultados encontrados nas demais etapas do projeto. Análise da sintaxe do .yaml e integração da linguagem com o servidor runner do github. Comparação com XML e JSON. Ajuda nos slides e produção de notas de estudo em yaml e CI/CD com githubactions. Concatenação dos artefatos no git.
Vitor Leonardo Sena de Lima - 202200014622	Criação dos diagramas “fluxo atual”, “Arquitetura do github actions” e “Timeline de eventos”. Além de fazer a análise no repositório para conseguir modelar os diagramas, descrevi o passo a passo de como funciona cada diagrama modelado. Criei parte das apresentações do slide (minha parte)
Carlos Daniel Lima de Gois - 202200078746	Ajudei com Análise da situação e atual do desenvolvimento e uma parte da metodologia de análise
David Silva Santana - 202200013993	Gravei o vídeo explicando os riscos e gargalos Identificados.

2. Introdução

A evolução contínua de sistemas de software exige mecanismos que garantam estabilidade, qualidade e segurança a cada nova modificação incorporada ao código. À medida que projetos crescem e passam a receber contribuições de múltiplos desenvolvedores, torna-se essencial adotar práticas que reduzam riscos de regressão e facilitem a manutenção do software ao longo do tempo. Nesse contexto, práticas de integração contínua e entrega contínua desempenham um papel fundamental ao automatizar verificações, testes e validações a cada alteração realizada no repositório.

Este projeto tem como foco a análise e auditoria do repositório Crawl4AI, uma ferramenta voltada para automação de coleta e processamento de informações, avaliando o nível de maturidade do seu processo de desenvolvimento sob a perspectiva de DevOps. A partir de uma investigação detalhada do ciclo de vida atual do software, busca-se identificar a existência (ou não) de mecanismos automatizados de verificação, bem como os impactos dessas práticas na confiabilidade do projeto e na experiência de novos contribuidores.

3. Projeto Escolhido

O projeto escolhido pela equipe foi o Crawl4AI (disponível em *[1]*) que consiste em síntese de um web scrapper para LLMs. Ele faz uma busca profunda e otimizada em diversos sites e retorna um texto markdown para melhor treinamento destas redes neurais. Escolhemos o projeto pelo interesse na área de web scrapping e seu contexto emergente na computação.

4. Metodologia de Análise

Este tópico detalha como o diagnóstico foi conduzido, descrevendo as etapas seguidas para investigar o repositório, como a verificação de ferramentas de CI/CD, a análise de arquivos de configuração e o histórico de Pull Requests. Também são indicadas as ferramentas e recursos utilizados durante a análise.

A análise foi realizada por meio da navegação no repositório do projeto Crawl4AI. Inicialmente, acessou-se o diretório .github/workflows/, sendo identificados múltiplos arquivos .yml que indicam a presença de pipelines automatizados.

O arquivo docker-release.yml automatiza a construção e publicação de imagens Docker, sendo acionado por releases ou tags docker-rebuild-v*. Executa build multi-arquitetura (amd64/arm64) com push para Docker Hub, utilizando cache para otimização.

O arquivo release.yml é o pipeline principal de release, acionado por tags v*. Executa verificação de versão, build do pacote Python e upload para PyPI, finalizando com criação de GitHub Release. Não inclui execução de testes automatizados.

O arquivo main.yml, diferentemente do que o nome sugere, tem função exclusiva de notificações para Discord, sendo acionado por eventos de issues, PRs e stars. Não executa validações de código.

Arquivos como release.yml.backup e test-release.yml.disabled indicam pipelines desativados, sugerindo evolução do processo de automação. Documentação em .github/workflows/docs/README.md explica a separação dos pipelines: builds Docker consomem 10-15 minutos, então a separação permite publicação rápida no PyPI (2-3 minutos) enquanto imagens Docker são construídas em paralelo.

A aba Actions registra mais de 13.000 execuções, incluindo pipelines de IA (Claude Code, Copilot Code Review) e integrações externas. Porém, a análise confirmou a ausência de testes automatizados obrigatórios em Pull Requests.

A análise de mais de 25 PRs recentes revelou padrões consistentes: nenhum apresenta checks automáticos de CI visíveis, campos de revisores automáticos e auto_merge estão vazios, e não há labels automáticas. Diversos PRs incluem testes executados localmente pelos contribuidores (ex: PRs #1730, #1700, #1707 mencionam testes passando localmente), mas não validados pelo CI.

O PR #1712 (release v0.8.0) foi mergeado com 58 arquivos alterados e correções de segurança, porém com checklist de testes não marcada. PRs duplicados (#1721, #1703, #1698) resolvem o mesmo issue sem detecção automática. PRs #1713-#1723 foram gerados com Claude Code e dependem apenas de revisão humana.

Os riscos identificados incluem: integração de código sem validação automatizada, releases publicadas sem testes, gargalos por dependência de revisão manual, e ausência de análise de segurança automatizada apesar de vulnerabilidades recentes (RCE, LFI).

As ferramentas utilizadas incluíram navegação no repositório GitHub, leitura dos arquivos YAML, consulta à API do GitHub para dados de PRs, e inspeção da aba Actions.

5. Breve discussão sobre Github actions

O GitHub Actions é uma plataforma de automação integrada ao GitHub que permite a criação de pipelines de Integração Contínua (CI) e Entrega Contínua (CD). Por meio de arquivos de configuração no formato YAML, é possível definir fluxos automatizados que são executados em resposta a eventos do repositório, como push de código ou abertura de pull requests.

Essa ferramenta tem como objetivo automatizar tarefas repetitivas, como compilação, execução de testes e análise de qualidade de código, contribuindo para a padronização e confiabilidade do processo de desenvolvimento.

Workflows

Os workflows do GitHub Actions são definidos por arquivos .yml localizados no diretório .github/workflows. Esses arquivos descrevem, de forma declarativa, como e quando as ações automatizadas devem ser executadas. Um workflow é composto por quatro elementos principais: eventos, jobs, steps e ações reutilizáveis, que em conjunto definem o comportamento do pipeline.

Eventos

Os eventos determinam quando um workflow será executado. No projeto analisado, observou-se o uso de eventos como push e pull_request, que disparam a execução do pipeline sempre que há alterações no código ou propostas de modificação no repositório. Essa estratégia garante que cada mudança seja automaticamente verificada antes de ser incorporada ao projeto principal, reduzindo a probabilidade de introdução de erros.

Jobs e Steps

Os jobs representam unidades de execução independentes dentro de um workflow, executadas em ambientes controlados fornecidos pelo GitHub, geralmente baseados em sistemas Linux. Cada job é composto por uma sequência de steps, que correspondem a comandos ou ações específicas a serem executadas. No projeto analisado, os steps incluem o checkout do código-fonte e a execução de ferramentas automatizadas de verificação, responsáveis por validar aspectos como estilo, organização e possíveis inconsistências no código.

Ações e Ferramentas Utilizadas

As ações utilizadas no workflow consistem em componentes reutilizáveis, mantidos pela comunidade ou pelo próprio GitHub, que encapsulam tarefas comuns do processo de CI. Essas ações facilitam a implementação de pipelines sem a necessidade de scripts complexos, promovendo simplicidade e reprodutibilidade. As ferramentas integradas ao pipeline permitem a identificação precoce de problemas, fornecendo feedback imediato aos desenvolvedores sobre a qualidade das alterações realizadas.

Relação com o Projeto Analisado

A utilização do GitHub Actions no projeto analisado contribui diretamente para a melhoria da qualidade do software, uma vez que automatiza verificações que antes dependiam de inspeção manual. A presença desses workflows demonstra a adoção de práticas modernas de desenvolvimento, alinhadas aos princípios de Integração Contínua.

6. Análise da Situação Atual do Desenvolvimento

O desenvolvimento do Crawl4AI apresenta um fluxo bem ativo, com muitos Pull Requests e automações já configuradas no GitHub Actions. Porém, a automação existente está mais focada em processos de release e notificações, e não em validação contínua do código antes do merge.

- Atualmente, quando um contribuidor abre um Pull Request, não há evidências de um pipeline de CI padrão (como execução automática de testes, lint ou checagens de segurança) rodando obrigatoriamente. Assim, a

qualidade do código depende principalmente de verificações manuais feitas pelos maintainers e de testes executados localmente pelos contribuidores.

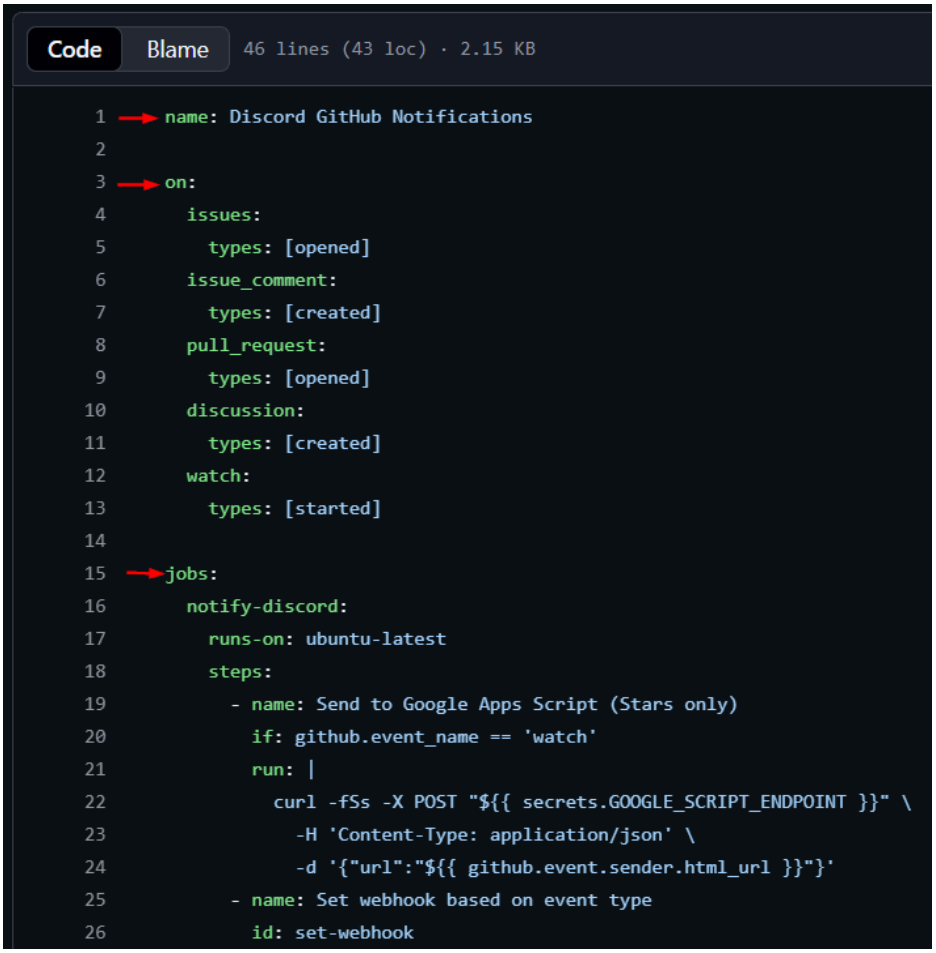
- Os workflows identificados reforçam essa característica:
  - release.yml: automatiza a publicação do pacote no PyPI quando uma tag v* é criada, mas não executa uma suíte de testes como etapa obrigatória.
  - docker-release.yml: constrói e publica imagens Docker quando uma release é publicada ou quando tags específicas são utilizadas.
  - main.yml: atua como um fluxo de notificações (ex.: Discord) para eventos do repositório, sem realizar validações técnicas.
- Dessa forma, apesar de existir automação, o projeto ainda mostra baixa maturidade em Integração Contínua, pois não utiliza checks automatizados como “gate de qualidade” antes do merge. Isso aumenta o risco de regressões e falhas chegarem à branch principal e, eventualmente, às releases.

7. Evidências

Este tópico explica quais evidências foram coletadas para sustentar o diagnóstico, como capturas de tela, arquivos de configuração ou registros do GitHub. Também indica onde essas evidências estão organizadas no repositório.

As evidências coletadas e utilizadas na análise foram os arquivos dentro do diretório ".github/workflows”, nele se encontram arquivos de configuração de extensão .yml que configuram as automações nos workflows.

(A imagem evidencia a presença de múltiplos arquivos de configuração no formato .yml. Esse diretório é o local padrão utilizado pelo GitHub para armazenar workflows do GitHub Actions)



(O print do arquivo main.yml mostra a definição de um workflow principal do projeto. Esse tipo de arquivo normalmente centraliza etapas essenciais do pipeline de Integração Contínua, como execução de scripts, validações e verificações automáticas acionadas por eventos como push ou pull request.)

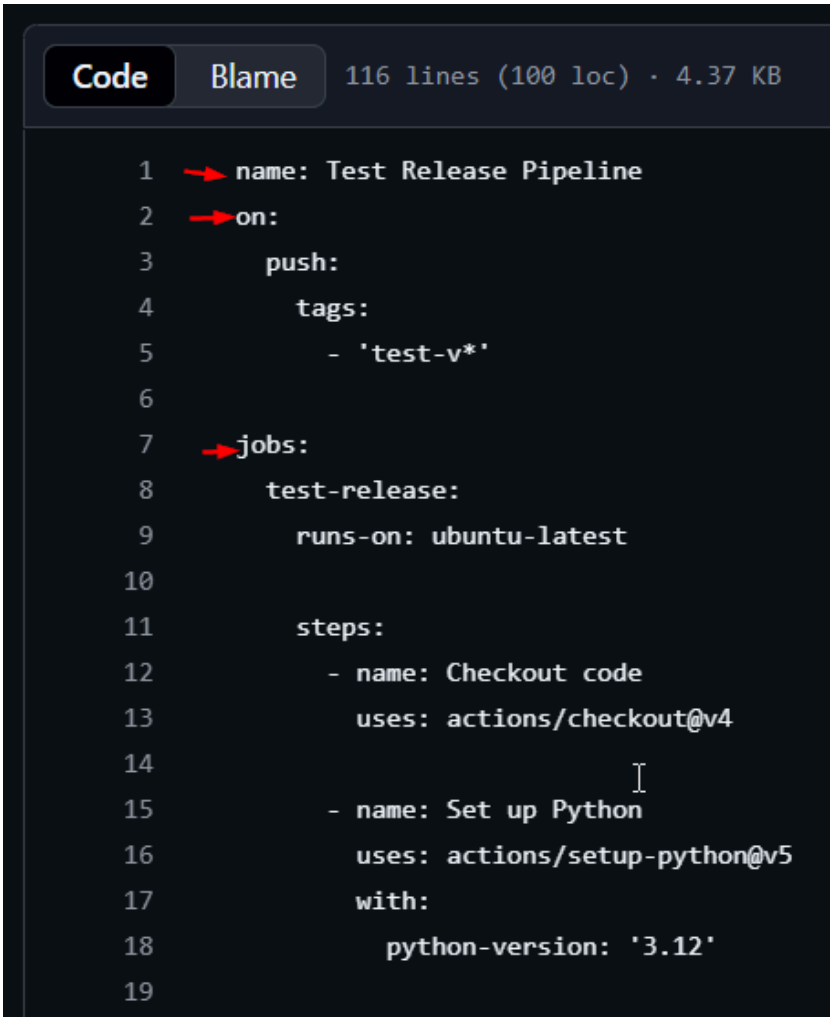
Code Blame 113 lines (94 loc) · 4.06 KB

```yaml
name: Release Pipeline
on:
  push:
    tags:
      - 'v*'
      - '!test-v*'  # Exclude test tags

jobs:
  release:
    runs-on: ubuntu-latest
    permissions:
      contents: write  # Required for creating releases

```

Code Blame 113 lines (94 loc) · 4.06 KB

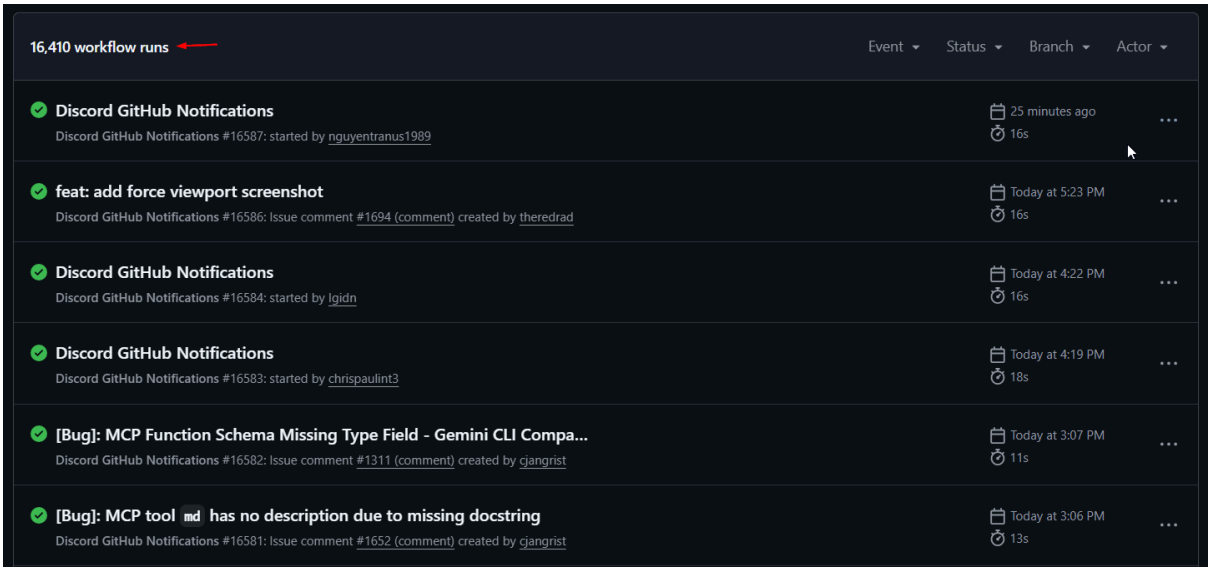
```yaml
name: Release Pipeline
on:
  push:
    tags:
      - 'v*'
      - '!test-v*'  # Exclude test tags

jobs:
  release:
    runs-on: ubuntu-latest
    permissions:
      contents: write  # Required for creating releases

```

(As capturas de tela dos arquivos release.yml e docker-release.yml mostram workflows específicos voltados para o processo de release do projeto e para a construção/publicação de imagens Docker)



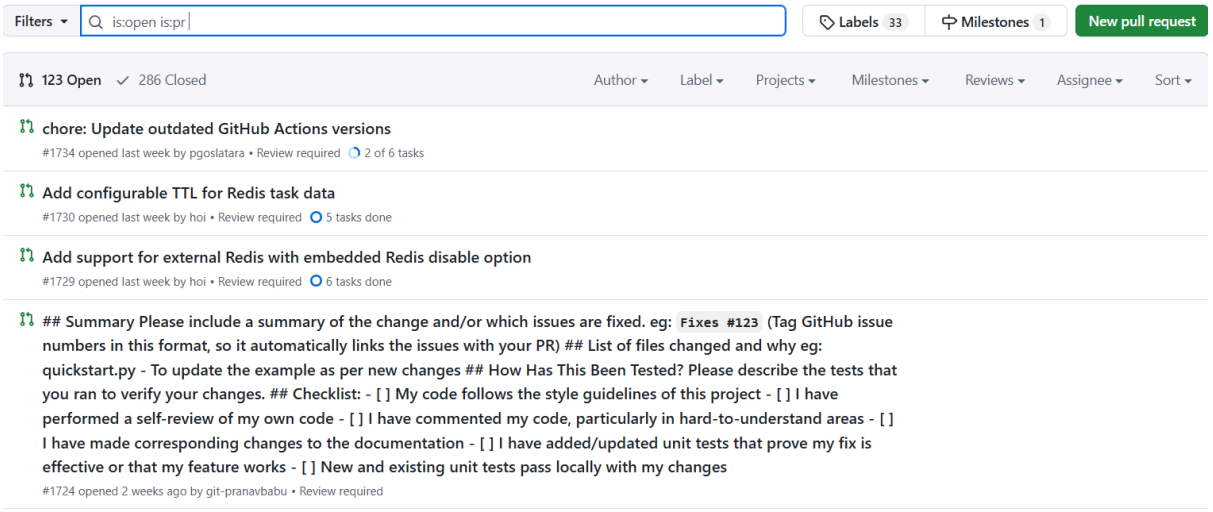
(A imagem apresenta o arquivo de workflow denominado “Test Release Pipeline”, configurado para ser executado quando ocorre um *push* de tags que seguem o padrão test-v*. O arquivo define um job chamado test-release, executado em ambiente Linux (ubuntu-latest), contendo etapas como o checkout do código-fonte e a configuração do ambiente Python)

Adicionalmente também foi utilizado o histórico de de Actions do repositório.



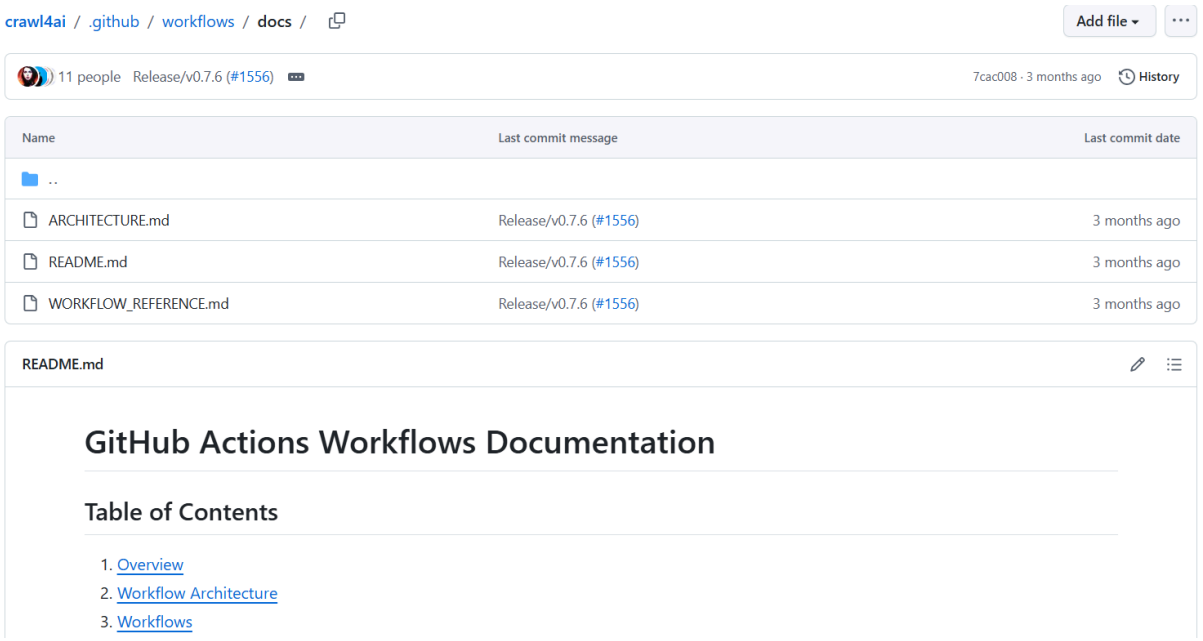
(O print da aba Actions do repositório Crawl4AI mostra o histórico de execuções dos workflows, com milhares de execuções registradas ao longo do tempo.)

Além disso, o histórico de Pull Requests também foi levado em consideração e utilizado com evidência.



(Histórico de Pull Requests)

Complementarmente, a análise também buscou extrair informações da documentação contida no diretório “.github/workflows/docs”.



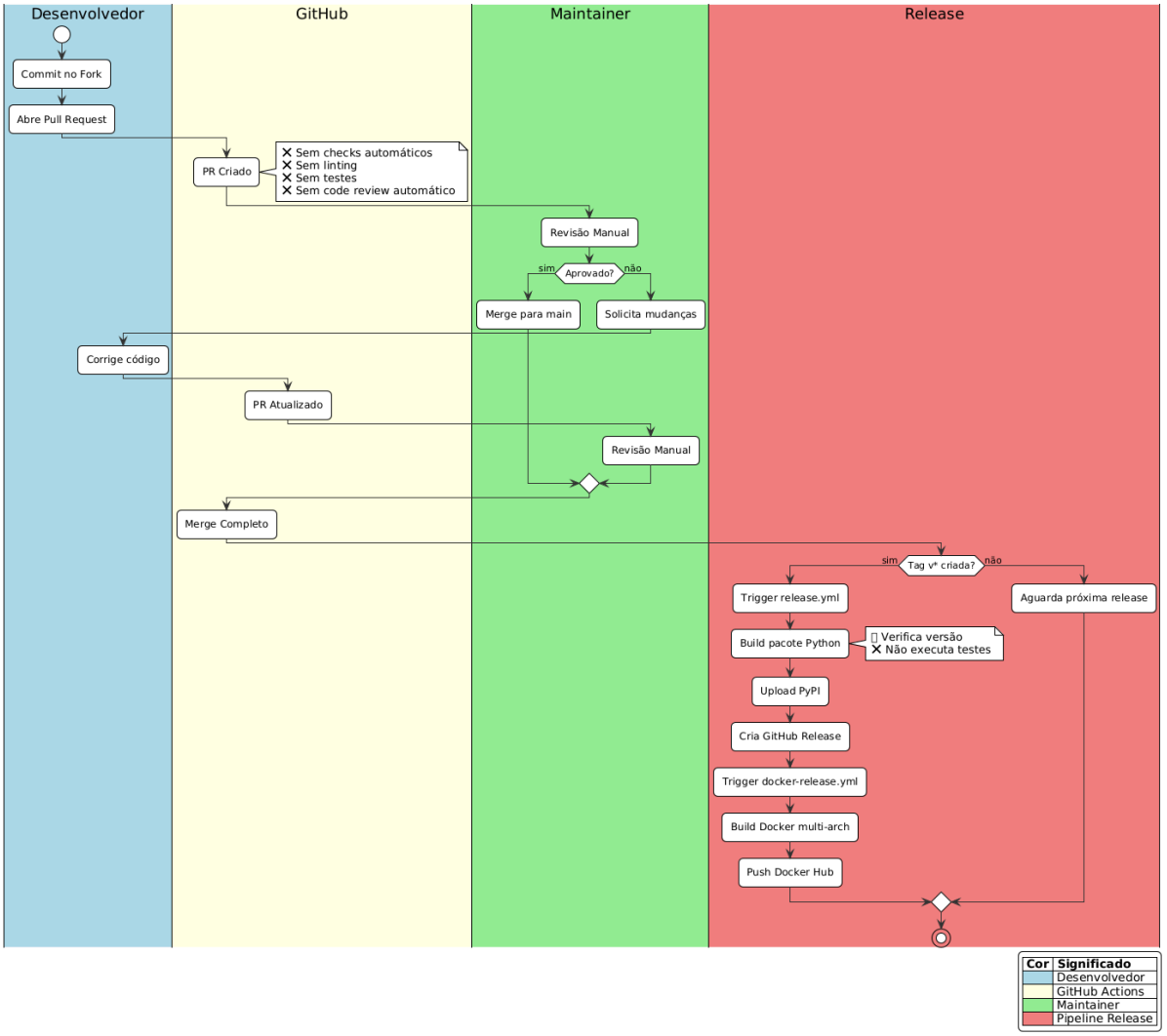
(Diretório “.github/workflows/docs”)

8. Mapeamento do Fluxo Atual

Aqui é descrito o fluxo de desenvolvimento do projeto conforme identificado na análise, desde a criação de código até o merge no repositório principal. O foco é mostrar de forma clara como as etapas se encadeiam e onde há dependência de ações manuais ou ausência de validações automáticas.

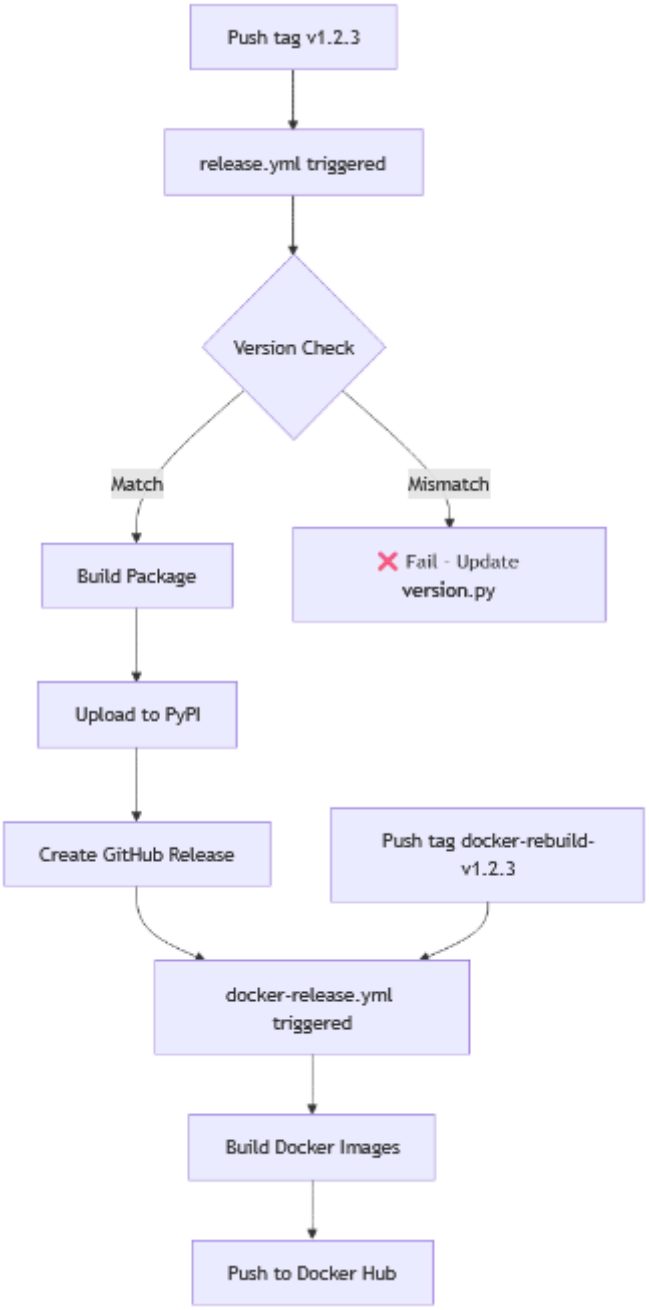
No repositório foi feito um mapeamento técnico do **fluxo de trabalho atual do projeto**, detalhando a **arquitetura de GitHub Actions** e a jornada do código desde a criação do Pull Request até a publicação. A análise evidencia um cenário onde a automação existe apenas para a etapa de entrega (CD para PyPI e Docker), enquanto a validação de qualidade (CI) é inexistente, tornando o processo dependente exclusivamente de revisões manuais sem a execução de testes automatizados antes da integração.

Fluxo Atual do Projeto Crawl4AI
(Sem CI de Testes)



Cor	Significado
	Desenvolvedor
	GitHub Actions
	Maintainer
	Pipeline Release

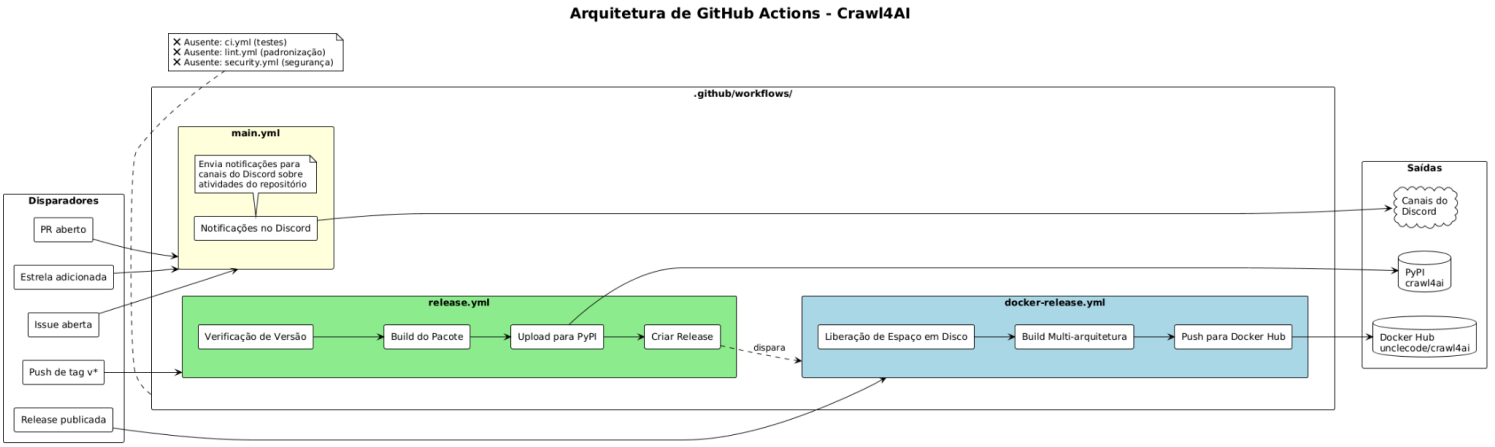
Fluxo para publicação de release:



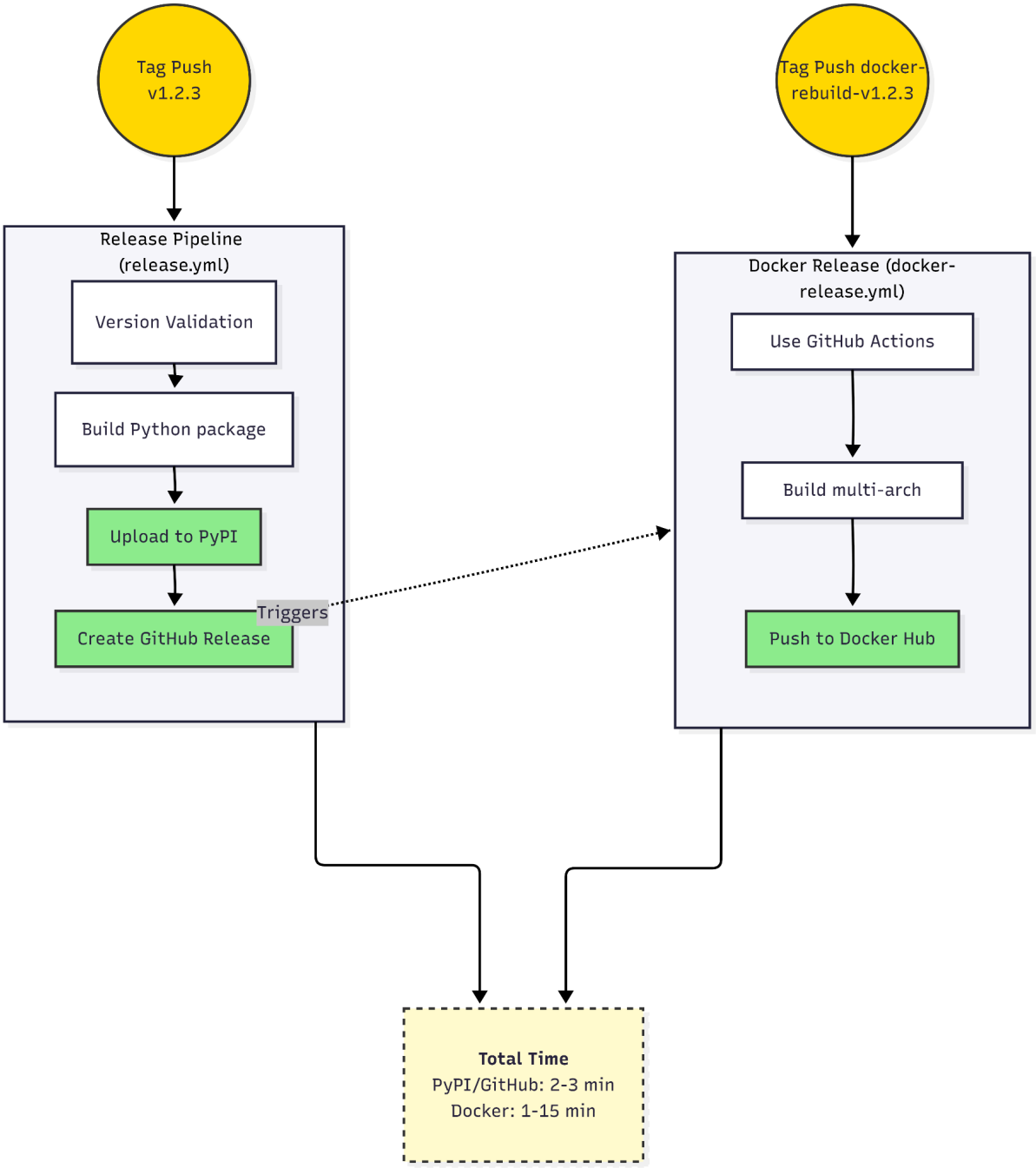
Passo a passo:

- ➢ Um desenvolvedor faz commits no fork/local e abre um Pull Request (PR) para o repositório principal.
- ➢ O PR é criado no GitHub. Pelo estado atual do repositório, o PR NÃO tem checks automáticos obrigatórios (não há um job de CI padrão que rode testes/lint).
- ➢ A revisão é feita manualmente por um maintainer (code review humano). Se aprovado, o maintainer faz merge para a branch principal (main). Se houver solicitações de mudança, o desenvolvedor faz ajustes e atualiza o PR.

- ➢ Separadamente, quando um tag de release (por ex. v0.8.0) é criado/pushado, o workflow release.yml é acionado: ele faz checkout, checa versão, constrói o pacote Python, e faz upload para PyPI e cria a GitHub Release.
- ➢ O Docker build é realizado por outro workflow (docker-release.yml) que é acionado quando a release é publicada (ou por uma tag específica docker-rebuild-v*). Esse workflow executa builds multi-arch e push para Docker Hub.



(Esse workflow cuida do build/entrega de imagens Docker, isolado do release do pacote Python.)

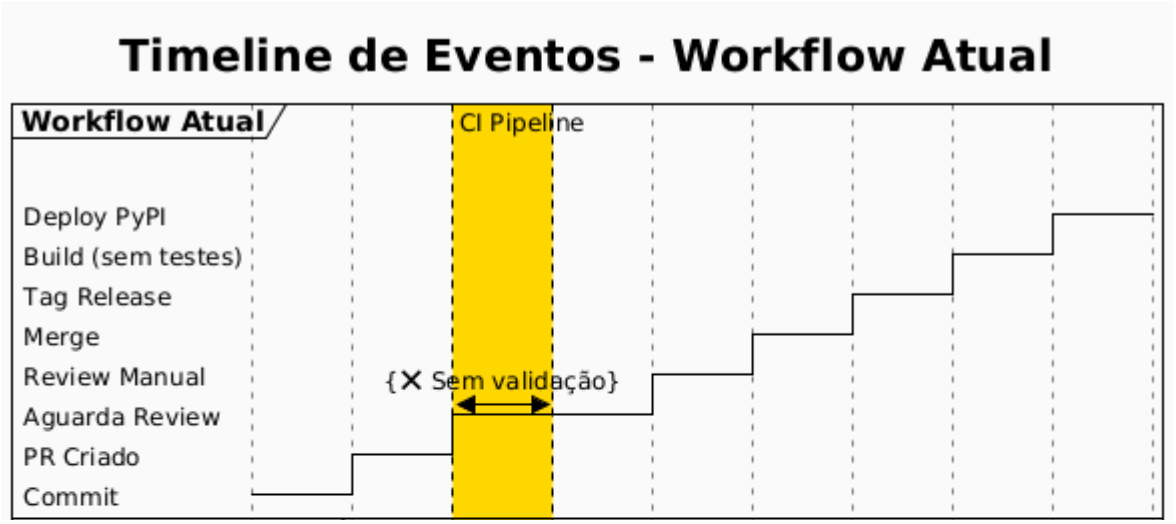


(Workflow de releases, que utiliza da split releases pipeline. A utilização dessa arquitetura permite que as builds de Docker, que podem demorar mais tempo em execução e bloquear os releases de pacotes mais rápidos, sejam executadas em paralelo.)

O que cada workflow faz:

- main.yml: workflow leve que reage a eventos (PRs, issues, discussions, stars) e envia notificações (configurado para postar em canais do Discord). É um workflow orientado a observabilidade/alerta, não a validação.
- release.yml: ativado por push de tag (v*). Executa:
  - checkout do código;
  - setup do Python;

    - extração/cheque de versão (concordância entre tag e crawl4ai/version.py);
    - instalação de dependências e build do pacote (python -m build);
    - twine check e upload para PyPI;
    - criação de GitHub Release com corpo de release. Esse workflow automatiza a publicação do pacote Python.
- docker-release.yml: ativado quando uma release é publicada (ou por tag docker-rebuild-v*):
    - prepara runner (limpa disco);
    - checkout;
    - configura Docker buildx e login no Docker Hub;
    - build multi-arch (linux/amd64, linux/arm64) e push;



Sequência temporal real (o que realmente ocorre):

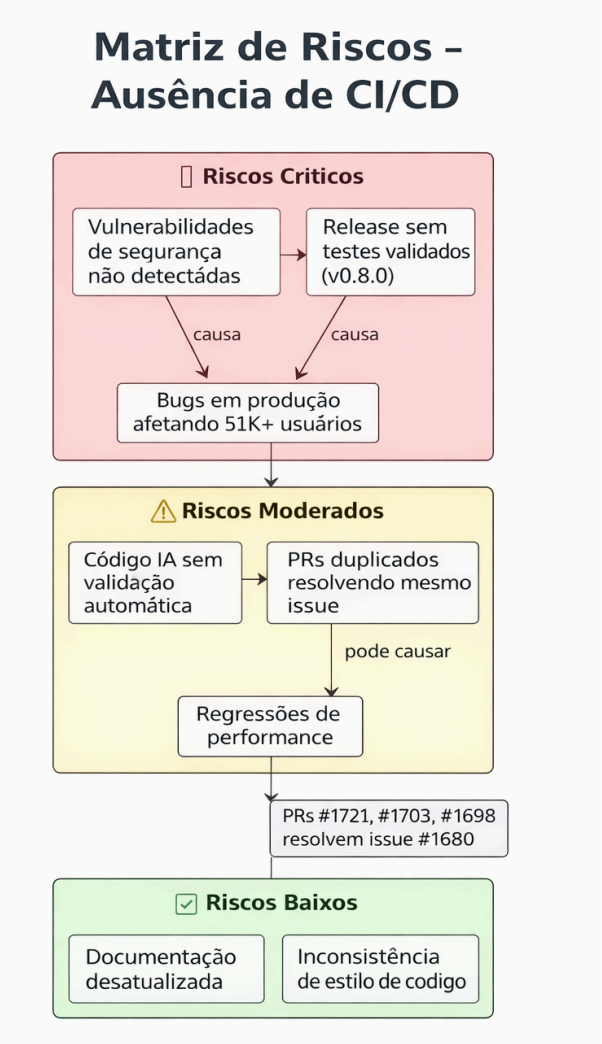
i. Commit: o desenvolvedor envia mudanças (commit → push) e abre um PR.
ii. PR Criado: o PR fica disponível no repositório.
iii. Aguarda Review: Não há execução automática de checks de qualidade padrão (nenhum job obrigatório de teste/lint é acionado), então o PR fica aguardando revisão humana.
iv. Review Manual: um maintainer realiza revisão de código manual — essa etapa pode levar tempo porque depende de disponibilidade humana e não há gates automáticos para priorizar ou bloquear merges.
v. Merge: após aprovação manual, o maintainer faz merge para a branch main.
vi. Tag Release (opcional): quando decidem liberar uma versão, criam uma tag (vX.Y.Z).
vii. Build: o release.yml é acionado para construir e publicar o pacote Python; este build não executa testes automatizados como parte do

workflow (consequência: a publicação pode ocorrer sem validações automatizadas).

viii. Deploy PyPI e Docker: pacote sobe para PyPI e a criação de release aciona docker-release.yml que constrói e publica imagens Docker separadamente.

9. Riscos e Gargalos Identificados

Este tópico discute os principais problemas e riscos associados ao cenário atual do projeto, como possibilidade de regressões, baixa confiabilidade do código, aumento do esforço de manutenção e dificuldades para novos contribuidores entenderem e evoluírem o software.



A partir do diagnóstico realizado sobre o uso de práticas de CI/CD no projeto Crawl4AI, foi possível identificar alguns riscos e gargalos associados ao cenário atual. Embora o projeto possua múltiplos workflows configurados no GitHub Actions, a ausência ou limitação de etapas voltadas à validação automatizada do código, como testes contínuos em Pull Requests, pode impactar diretamente a confiabilidade e a manutenibilidade do software.

Como a imagem mostra na matriz de riscos apresentada, os riscos críticos estão relacionados principalmente à possibilidade de falhas não detectadas durante o

processo de integração. A inexistência de validações automáticas robustas pode permitir que alterações sejam incorporadas ao código sem a devida verificação, aumentando a probabilidade de introdução de vulnerabilidades de segurança e bugs em produção. Esse cenário pode resultar em falhas que afetam diretamente os usuários finais, exigindo correções emergenciais e aumentando o custo de manutenção do projeto.

No nível de riscos moderados, vale a pena destacar a falta de validação automática do código como um fator que contribui para retrabalho e inconsistências durante o desenvolvimento. A resolução repetida de problemas semelhantes, bem como a possibilidade de regressões de desempenho, são consequências comuns quando não há um pipeline de testes que valide continuamente novas alterações. Esses fatores reduzem a eficiência da equipe e dificultam a evolução sustentável do software.

Um exemplo concreto desse retrabalho operacional é o risco de versões inutilizadas no processo de publicação. Conforme identificado na documentação de Troubleshooting do projeto, o pipeline de release atual valida a consistência da versão apenas após o acionamento da tag de lançamento. Devido à política de imutabilidade do repositório PyPI (Python Package Index), que proíbe o re-upload de arquivos com o mesmo nome, uma falha tardia no pipeline obriga a equipe a incrementar a versão (ex: alterar de v1.2.3 para v1.2.4) apenas para corrigir o fluxo. Esse cenário gera ruído no histórico de lançamentos e desperdício de tempo, algo que seria facilmente evitado com verificações de consistência implementadas diretamente na etapa de Pull Request.

Já os riscos baixos estão associados a aspectos como documentação desatualizada e inconsistências no estilo de código. Apesar desses problemas não causarem falhas imediatas no funcionamento do sistema, eles impactam diretamente a experiência de novos contribuidores. A ausência de padronização clara e de validações automáticas pode aumentar a curva de aprendizado, tornando mais difícil compreender o funcionamento do projeto e contribuir de forma segura e eficiente.

De forma geral, os riscos e gargalos identificados indicam que, apesar do projeto Crawl4AI apresentar um nível relevante de automação voltado a processos de release e integração com ferramentas externas, ainda há espaço para aprimorar a maturidade do pipeline de CI/CD. A inclusão de validações automáticas, especialmente testes executados em Pull Requests, poderia reduzir significativamente a ocorrência de regressões, aumentar a confiabilidade do código e facilitar a entrada de novos contribuidores no projeto.

10. Link do Vídeo de Apresentação e do repositório

URL: https://www.canva.com/design/DAG_nLbhyKU/pW91kVQZuaIJ1Bv5k-61uw/edit
URL: https://github.com/ViniciusVasconi/3-etata-DiagnosticoPipelineCI-Crawl4ai-