Universidade Federal de Ouro Preto
Instituto de Ciências Exatas e Aplicadas
Departamento de Computação e Sistemas

Automação sob Restrições: Estrutura e Eficácia de Pipelines GitHub Actions em Sistemas Operacionais de Tempo Real FLOSS

Laura Chaves Soares

TRABALHO DE CONCLUSÃO DE CURSO

ORIENTAÇÃO:
Igor Muzetti Pereira

Fevereiro, 2026
João Monlevade – MG

Laura Chaves Soares

Automação sob Restrições: Estrutura e Eficácia de Pipelines GitHub Actions em Sistemas Operacionais de Tempo Real FLOSS

Orientador: Igor Muzetti Pereira

Monografia apresentada ao curso de Sistemas de Informação do Instituto de Ciências Exatas e Aplicadas, da Universidade Federal de Ouro Preto, como requisito parcial para aprovação na Disciplina "Trabalho de Conclusão de Curso II".

Universidade Federal de Ouro Preto

João Monlevade

Fevereiro, 2026

SISBIN - SISTEMA DE BIBLIOTECAS E INFORMAÇÃO

S676a Soares, Laura Chaves.
Automação sob restrições [manuscrito]: estrutura e eficácia de pipelines GitHub Actions em sistemas operacionais de tempo real FLOSS. / Laura Chaves Soares. - 2026.
54 f.: il.: gráf., tab..

Orientador: Prof. Dr. Igor Muzetti Pereira.
Monografia (Bacharelado). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Aplicadas. Graduação em Sistemas de Informação .

1. Desenvolvimento ágil de software. 2. Engenharia de software. 3. GitHub (Programa de computador). 4. Processamento eletrônico de dados em tempo real. 5. Sistemas operacionais (Computadores) - Confiabilidade. 6. Software livre. 7. Software - Testes. I. Pereira, Igor Muzetti. II. Universidade Federal de Ouro Preto. III. Título.

CDU 004.451.9:004.41

Bibliotecário(a) Responsável: Flavia Reis - CRB6/2431

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
REITORIA
INSTITUTO DE CIENCIAS EXATAS E APLICADAS
DEPARTAMENTO DE COMPUTACAO E SISTEMAS

FOLHA DE APROVAÇÃO

Laura Chaves Soares

Automação sob Restrições: Estrutura e Eficácia de Pipelines GitHub Actions em Sistemas Operacionais de Tempo Real FLOSS

em

Monografia apresentada ao Curso de Sistemas de Informação da Universidade Federal
de Ouro Preto como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação

Aprovada em 4 de Março de 2026

Membros da banca

Doutor Igor Muzetti Pereira - Orientador - Universidade Federal de Ouro Preto
Doutor Fernando Bernardes de Oliveira - Universidade Federal de Ouro Preto
Doutor Roberto Gomes Ribeiro - Universidade Federal de Ouro Preto

Igor Muzetti Pereira, orientador do trabalho, aprovou a versão final e autorizou seu depósito na Biblioteca Digital de Trabalhos de Conclusão de Curso da UFOP em 10/03/2026

Documento assinado eletronicamente por **Igor Muzetti Pereira**, **PROFESSOR DE MAGISTERIO SUPERIOR**, em 10/03/2026, às 19:46, conforme horário oficial de Brasília, com fundamento no art. 6º, § 1º, do Decreto nº 8.539, de 8 de outubro de 2015.

A autenticidade deste documento pode ser conferida no site http://sei.ufop.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0 , informando o código verificador **1073094** e o código CRC **4BDFB234**.

R. Diogo de Vasconcelos, 122, - Bairro Pilar Ouro Preto/MG, CEP 35402-163
Telefone: (31)3808-0819 - www.ufop.br

Dedico este trabalho à minha família e aos meus amigos, alicerces que sustentaram cada passo dessa caminhada.

Agradecimentos

Com imensa gratidão, agradeço primeiramente a Deus, por me conceder força, saúde e sabedoria ao longo de toda esta caminhada. Em seguida, agradeço aos meus pais, Luciene e Romeu, por tornarem possível a realização do meu maior sonho: estudar em uma Universidade Federal. O amor incondicional, o esforço diário e os sacrifícios feitos por vocês foram os alicerces que sustentaram cada passo dessa trajetória.

Aos meus irmãos, Michele e Victor, deixo meu eterno agradecimento por estarem sempre ao meu lado, oferecendo apoio e incentivo nos momentos em que mais precisei. À minha afilhada, Maria Cecília, e à minha sobrinha, Maitê, agradeço por iluminarem meus dias e por representarem a forma mais pura de amor em minha vida. À minha sobrinha Lavínia, que em breve chegará, deixo um agradecimento especial pela certeza de que também trará ainda mais luz e alegria à nossa família.

Ao meu namorado, melhor amigo e maior incentivador, Kaique, sou profundamente grata pela presença constante, pelo apoio incondicional e por acreditar em mim até mesmo quando eu duvidei. Seu amor, suas palavras e seus gestos foram fundamentais para que eu permanecesse firme ao longo de toda essa jornada.

Aos amigos que caminharam comigo, em especial Nathália e Emily, agradeço por serem abrigo, refúgio e fortaleza nos momentos mais difíceis, tornando os dias desafiadores mais leves e as conquistas ainda mais significativas. À República Cazamiga, que se tornou meu lar e me presenteou com uma família, agradeço por cada experiência vivida, pelas risadas, pelas conversas, pelos desafios compartilhados e pelas vitórias celebradas juntas. Tudo o que ali foi construído permanece em mim como parte da minha história. À República Sparta, agradeço pela amizade, pelo apoio nas adversidades e pelas inúmeras risadas que trouxeram leveza aos meus dias.

Agradeço à Universidade Federal de Ouro Preto pela oportunidade de cursar esta graduação; ao meu orientador, Igor, pelo suporte na construção deste trabalho e pelo conhecimento compartilhado ao longo do processo; e a todos os docentes que dividiram seus saberes e contribuíram significativamente para minha formação acadêmica.

Por fim, agradeço a todos que, de alguma forma, fizeram parte dessa conquista e contribuíram para minha formação pessoal e profissional.

"Life is a series of building, testing, changing and iterating."

— Lauren Mosenthal

Resumo

Sistemas Operacionais de Tempo Real (RTOS) livres desempenham papel essencial em aplicações críticas, pois exigem previsibilidade e confiabilidade em seus processos. A crescente adoção de práticas de integração e entrega contínua (IC/EC) em projetos FLOSS torna relevante compreender como esses mecanismos são estruturados e aplicados em ambientes de alta exigência. A pesquisa analisou workflows automatizados em projetos RTOS hospedados no GitHub, utilizando dados coletados pela API da plataforma. Foram categorizadas as Actions empregadas nos pipelines e avaliadas métricas operacionais, incluindo número de execuções, taxas de sucesso e falha, tempo médio de execução e ocorrência de reexecuções. Os resultados revelaram diferentes níveis de maturidade e confiabilidade entre os projetos. Iniciativas com maior suporte institucional apresentaram elevado volume de execuções e workflows complexos, mas enfrentaram desafios de estabilidade e tempo de processamento. Projetos conduzidos por comunidades abertas demonstraram equilíbrio entre desempenho e consistência, enquanto aqueles com menor suporte evidenciaram limitações operacionais, como baixa atividade ou elevada taxa de falhas. A eficácia dos pipelines mostrou-se dependente de três dimensões observadas nos resultados: o grau de automação, evidenciado pela complexidade dos workflows e diversidade de Actions utilizadas; a robustez da infraestrutura, refletida nas métricas de tempo médio de execução, taxas de falha e necessidade de reexecuções; e o suporte institucional, associado ao volume de execuções e à manutenção contínua dos projetos. Esses elementos, em conjunto, explicam as diferenças de maturidade e confiabilidade identificadas entre os projetos analisados. Um achado interessante é a existência de Actions personalizadas não categorizadas oficialmente pelo GitHub Actions, dificultando comparações sistemáticas e sugerindo investigações futuras sobre sua disseminação e impacto na automação de projetos FLOSS. Conclui-se que o estudo contribui para a compreensão da aplicação de práticas de IC/EC em RTOS de código aberto, oferecendo subsídios práticos para comunidades e desenvolvedores, além de referências empíricas para pesquisadores da área de engenharia de software aplicada a sistemas críticos.

Palavras-chaves: sistemas operacionais de tempo real. software livre. github actions. integração contínua. entrega contínua.

Abstract

Open-source Real-Time Operating Systems (RTOS) play an essential role in critical applications, as they require predictability and reliability in their processes. The growing adoption of Continuous Integration and Continuous Delivery (CI/CD) practices in FLOSS projects makes it relevant to understand how these mechanisms are structured and applied in high-demand environments. This research analyzed automated workflows in RTOS projects hosted on GitHub, using data collected through the platform's API. The Actions employed in the pipelines were categorized, and operational metrics were evaluated, including number of executions, success and failure rates, average execution time, and occurrence of reruns. The results revealed different levels of maturity and reliability among the projects. Initiatives with greater institutional support showed a high volume of executions and more complex workflows but faced challenges related to stability and processing time. Community-driven projects demonstrated a balance between performance and consistency, while those with less support exhibited operational limitations, such as low activity or high failure rates. The effectiveness of the pipelines proved to depend on three dimensions observed in the results: the degree of automation, evidenced by the complexity of workflows and diversity of Actions used; the robustness of the infrastructure, reflected in metrics such as average execution time, failure rates, and need for reruns; and institutional support, associated with the volume of executions and continuous project maintenance. Taken together, these elements explain the differences in maturity and reliability identified among the analyzed projects. An interesting finding is the existence of customized Actions not officially categorized by GitHub Actions, which hinders systematic comparisons and suggests future investigations into their dissemination and impact on FLOSS project automation. It is concluded that this study contributes to the understanding of CI/CD practices in open-source RTOS, offering practical insights for communities and developers, as well as empirical references for researchers in the field of software engineering applied to critical systems.

Key-words: real-time operating systems. open-source software. GitHub actions. continuous integration. continuous delivery.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

EC	Entrega Contínua
FLOSS	Software Livre e de Código Aberto
IC	Integração Contínua
RTOS	Sistemas Operacionais de Tempo Real

Sumário

1 Introdução

Os Sistemas Operacionais de Tempo Real (RTOS) desempenham papel central em aplicações críticas, como sistemas de controle de tráfego aéreo, dispositivos médicos e ambientes industriais embarcados. Diferentemente dos sistemas operacionais convencionais, os RTOS são projetados para garantir previsibilidade e confiabilidade, assegurando que tarefas sejam concluídas dentro de prazos estritos. Conforme exposto por Sommerville (2011), tais sistemas organizam o software em processos cooperativos e utilizam arquiteturas específicas que permitem o cumprimento rigoroso de restrições temporais, constituindo-se em elementos essenciais para a segurança e a estabilidade de sistemas críticos.

No cenário contemporâneo de desenvolvimento de software livre e de código aberto (*FLOSS - Free/Libre Open Source Software*), a adoção de práticas de integração e entrega contínua (IC/EC) tornou-se imprescindível para assegurar qualidade, consistência e agilidade nos processos de desenvolvimento. Plataformas amplamente utilizadas, como o GitHub, disponibilizam mecanismos nativos de automação, entre os quais se destaca o GitHub Actions, que possibilita a execução de builds, testes e deploys de forma automatizada.

Embora os RTOS sejam amplamente reconhecidos por sua relevância em sistemas críticos, ainda são escassos os estudos que abordam de forma sistemática como projetos FLOSS desse domínio configuram e mantêm seus pipelines de IC/EC. A ausência de investigações aprofundadas sobre padrões de workflows e métricas de eficácia operacional dificulta a compreensão acerca da adequação dessas práticas às exigências de confiabilidade e previsibilidade que caracterizam sistemas de tempo real. Nesse sentido, este trabalho tem como propósito central investigar a estrutura e a eficácia dos pipelines em projetos RTOS FLOSS hospedados no GitHub, examinando os padrões de configuração dos workflows (eventos, ações e gatilhos) e avaliando métricas de eficácia operacional, incluindo tempo médio de execução, taxas de sucesso e falha, e frequência de builds.

Objetivo geral: analisar empiricamente como projetos RTOS FLOSS hospedados no GitHub estruturam e operam seus pipelines de IC/EC, identificando padrões de configuração e avaliando sua eficácia operacional.

Objetivos específicos:

- Identificar e categorizar as Actions utilizadas nos workflows dos projetos RTOS FLOSS;
- Avaliar métricas de desempenho e confiabilidade dos pipelines, como número de execuções, taxa de sucesso, taxa de falha, tempo médio de execução e reexecuções;

- Comparar práticas adotadas em projetos com diferentes níveis de suporte institucional e comunitário;
- Discutir os resultados à luz da literatura existente, destacando semelhanças e diferenças em relação a outros domínios de software crítico.

A partir desse propósito, duas questões de pesquisa orientam o estudo: RQ1: Como os projetos RTOS FLOSS estruturam seus *pipelines* no GitHub Actions, considerando os eventos que acionam os workflows, as categorias de Actions utilizadas e os padrões de configuração empregados? RQ2: Quão eficazes são esses *pipelines* em termos de desempenho e estabilidade, levando em conta o tempo médio de execução, as taxas de sucesso e de falha, a ocorrência de reexecuções e a frequência de builds?

A escolha do tema se fundamenta na relevância dos RTOS em aplicações críticas, nas quais a previsibilidade e a confiabilidade são requisitos indispensáveis. A crescente utilização de práticas de IC/EC em projetos FLOSS torna pertinente investigar como tais mecanismos são aplicados em projetos que lidam com requisitos tão rigorosos quanto os de sistemas de tempo real. A análise da estrutura dos pipelines e de suas métricas de eficácia operacional permite compreender se esses processos atendem às demandas de desempenho e estabilidade esperadas em ambientes críticos. Além disso, possibilita identificar padrões de configuração e práticas recorrentes que podem servir como referência para outros projetos, contribuindo para a consolidação de boas práticas de engenharia de software.

Do ponto de vista metodológico, o estudo adota uma abordagem quantitativa e descritiva, baseada na coleta de dados públicos por meio da API do GitHub. Foram extraídos metadados dos repositórios, arquivos YAML de workflows e métricas de execução dos pipelines. A análise foi conduzida com apoio de scripts automatizados e organizada em tabelas e gráficos, permitindo identificar padrões, comparar projetos e discutir os resultados em relação ao referencial teórico.

As principais contribuições deste trabalho situam-se no campo empírico. Do ponto de vista prático, os resultados oferecem subsídios para que comunidades de software livre e desenvolvedores de RTOS aprimorem seus pipelines, incorporando práticas que aumentem a confiabilidade, a rastreabilidade e a eficiência operacional. No âmbito da pesquisa, o estudo amplia o conhecimento sobre a integração entre RTOS e mecanismos de IC/EC em plataformas abertas, fornecendo evidências empíricas sobre padrões de workflows e métricas de desempenho. Dessa forma, contribui para o avanço das discussões sobre engenharia de software em sistemas críticos e abre espaço para investigações futuras voltadas à adaptação de práticas de automação às especificidades dos sistemas de tempo real.

O restante deste trabalho é organizado como se segue. O Capítulo 2 apresenta os fundamentos teóricos necessários para a compreensão da pesquisa, discutindo os conceitos

relacionados a sistemas operacionais de tempo real, a plataforma GitHub e o recurso GitHub Actions. Além disso, são abordadas as práticas de integração contínua e entrega contínua, bem como as métricas utilizadas para avaliar pipelines de IC/EC, incluindo a seção de trabalhos correlatos que contextualiza a pesquisa em relação a estudos anteriores.

O Capítulo 3 descreve os procedimentos adotados para a realização do estudo. São detalhados os critérios de seleção dos projetos analisados, o processo de coleta de dados, a extração e categorização das Actions, as métricas de IC/EC utilizadas e, por fim, a abordagem empregada na análise dos dados.

O Capítulo 4 apresenta os achados da pesquisa. Inicialmente, são descritas a estrutura dos pipelines e suas características, incluindo eventos de acionamento dos workflows, categorias das Actions e frequência de uso. Em seguida, é realizada a avaliação da eficácia operacional dos pipelines, considerando métricas como total de execuções, taxa de sucesso, taxa de falha, reexecuções, tempo médio de execução e a média anual das métricas.

Por fim, o Capítulo 6 sintetiza os principais resultados obtidos, destacando as contribuições da pesquisa e apontando possíveis direções para trabalhos futuros.

2 Fundamentação teórica

Este capítulo tem como objetivo apresentar os conceitos fundamentais relacionados aos sistemas operacionais de tempo real, às práticas de integração e entrega contínua e à automação de workflows em plataformas abertas, com destaque para o GitHub Actions. A revisão da literatura busca consolidar o estado da arte sobre o tema, fornecendo embasamento teórico para a análise desenvolvida neste trabalho.

2.1 Conceitos básicos

2.1.1 Sistemas operacionais de tempo real (RTOS)

Os sistemas operacionais de tempo real são uma categoria específica de sistemas operacionais desenvolvidos para atender requisitos de tempo determinístico. Isso significa que, além de executar funções básicas de gerenciamento de processos e recursos, eles precisam garantir que determinadas tarefas sejam concluídas dentro de prazos rígidos e previsíveis. Em ambientes críticos, como sistemas embarcados em automóveis, aeronaves ou dispositivos médicos, a previsibilidade temporal é tão importante quanto a capacidade do sistema de executar suas funções de maneira precisa e confiável.

Segundo Sommerville (2011), sistemas de software que operam em contextos críticos devem priorizar confiabilidade e tempo de resposta, pois atrasos ou falhas podem comprometer a segurança ou a funcionalidade do produto. Um RTOS, portanto, é construído com mecanismos de escalonamento que asseguram que tarefas prioritárias sejam executadas antes das demais, além de oferecer baixa latência para responder a interrupções externas. Essa combinação de características torna os RTOS fundamentais em aplicações onde o tempo de resposta não pode ser apenas rápido, mas garantidamente dentro de limites estabelecidos.

2.1.2 Software Livre

O conceito de software livre está fundamentado na defesa da liberdade dos usuários em relação ao uso e ao controle dos programas que executam. De acordo com a Foundation (2021), um software é considerado livre quando assegura quatro liberdades essenciais: a liberdade de executar o programa para qualquer finalidade; a liberdade de estudar seu funcionamento e adaptá-lo às necessidades específicas, o que pressupõe acesso ao código-fonte; a liberdade de redistribuir cópias para auxiliar outras pessoas; e a liberdade de modificar o programa e disponibilizar essas versões aprimoradas para a comunidade.

Essas garantias não se relacionam ao custo do software, mas sim à autonomia do usuário, sendo "livre" entendido no sentido de liberdade de expressão e não de gratuidade.

Os programas proprietários, ao restringirem essas liberdades, limitam o poder dos usuários e concentram o controle nos desenvolvedores. Em contraste, o software livre promove colaboração, transparência e compartilhamento de conhecimento, permitindo que comunidades inteiras se beneficiem de melhorias coletivas. É importante destacar que o caráter livre não exclui a possibilidade de exploração comercial: softwares podem ser distribuídos mediante cobrança ou associados a serviços de suporte e consultoria, desde que as liberdades fundamentais sejam preservadas.

2.1.3 GitHub

O GitHub[1] é uma plataforma de hospedagem e colaboração de código que se consolidou como uma das principais ferramentas de desenvolvimento de software em escala global. Baseado no sistema de controle de versão Git, o GitHub permite que desenvolvedores mantenham o histórico de alterações de seus projetos, colaborem em tempo real e integrem práticas modernas de engenharia de software.

De acordo com a documentação oficial, o GitHub oferece funcionalidades como criação de repositórios, gerenciamento de *branches*, abertura de *issues* para rastreamento e uso de *pull requests* para revisão e integração de código. Além disso, a plataforma integra mecanismos de discussão e colaboração que tornam o processo de desenvolvimento mais transparente e participativo. A combinação entre versionamento distribuído e colaboração social transformou o GitHub em um espaço central para projetos de código aberto, permitindo que comunidades inteiras contribuam para o avanço de softwares complexos.

2.1.4 GitHub Actions

O GitHub Actions[2] é o sistema de automação nativo da plataforma GitHub, criado para descrever workflows em arquivos YAML, nos quais são especificadas as etapas que devem ser executadas, os ambientes de execução e os eventos que disparam essas automações.

Segundo a documentação oficial, cada workflow pode ser composto por múltiplos jobs, que por sua vez contêm steps responsáveis por executar ações específicas, como compilar código, rodar testes ou realizar *deploys*. Esses workflows podem ser acionados por eventos como um *push* em determinada *branch*, a abertura de um *pull request* ou até mesmo por agendamentos periódicos. A flexibilidade do GitHub Actions permite

[1] https://github.com/
[2] https://github.com/features/actions?locale=pt-BR

que equipes adaptem seus *pipelines* às necessidades do projeto, desde tarefas simples de validação até processos complexos de compilação cruzada e testes em múltiplos ambientes.

Outro aspecto relevante é a integração com a comunidade: milhares de Actions reutilizáveis estão disponíveis publicamente, permitindo que desenvolvedores incorporem soluções prontas em seus workflows. Isso reduz o esforço de configuração e promove boas práticas de automação.

2.1.5 Integração contínua (IC)

A Integração Contínua (*Continuous Integration – CI*) é uma prática consolidada na engenharia de software que busca integrar alterações de código de forma frequente e automatizada, reduzindo riscos e aumentando a confiabilidade do processo de desenvolvimento. Cada modificação realizada por um desenvolvedor é incorporada ao projeto principal em ciclos curtos, acompanhada de processos automáticos de compilação e execução de testes. Essa abordagem evita o acúmulo de grandes integrações, que historicamente resultavam em conflitos complexos e demorados de resolver, fenômeno descrito por Valente (2020) como *merge hell*.

O princípio fundamental da IC é que quanto mais cedo e mais frequentemente o código é integrado, menores são os riscos e custos associados ao processo. Em vez de concentrar grandes blocos de alterações em períodos longos, a prática recomenda integrações diárias ou múltiplas vezes ao longo do dia. Dessa forma, cada integração envolve apenas pequenas mudanças, o que facilita a detecção precoce de erros e a resolução de conflitos. Conforme enfatiza Valente (2020), atrasar a integração aumenta a imprevisibilidade e o esforço necessário para corrigir problemas, tornando o processo mais oneroso do que a própria implementação da funcionalidade.

Segundo Fowler (2006), a integração contínua deve ser entendida não apenas como uma técnica de automação, mas como uma prática cultural que promove transparência e colaboração. O autor destaca que a IC só alcança seu potencial quando acompanhada de disciplina da equipe e de mecanismos que assegurem feedback rápido, permitindo que erros sejam identificados e corrigidos imediatamente.

Para que a IC seja eficaz, é imprescindível o uso de mecanismos de automação que assegurem a execução de compilação e testes a cada integração. Esse processo permite identificar falhas rapidamente e corrigi-las antes que comprometam o trabalho coletivo. Fowler (2006) reforça que um dos princípios centrais da IC é manter o código em um estado sempre “executável”, o que exige prioridade máxima na correção de qualquer *build* quebrado.

2.1.6 Entrega e Implantação Contínua (EC)

A Entrega e Implantação Contínua (*Continuous Delivery and Deployment – CD*) é uma prática de engenharia de software que tem como objetivo manter o sistema sempre pronto para ser colocado em produção. Cada alteração realizada no código, denominada *commit*, deve ser tratada como potencialmente liberável. Após cada *commit*, o código passa por processos automatizados de compilação e testes, garantindo que o *software* permaneça estável e apto para implantação. Nesse contexto, é importante distinguir dois conceitos: *delivery* corresponde ao processo de preparar uma nova versão para implantação, enquanto *deployment* refere-se à efetiva disponibilização dessa versão para os usuários finais Valente (2020).

Segundo Fowler (2010), a entrega contínua deve ser entendida como uma extensão natural da integração contínua, cujo propósito é assegurar que o sistema esteja sempre em condições de ser implantado. O autor enfatiza que cada alteração no código deve passar por uma cadeia de validações automatizadas, de modo que o software esteja constantemente em um estado confiável. Essa perspectiva complementa a visão de Valente (2020), que destaca a importância de tratar cada *commit* como potencialmente liberável, reduzindo riscos e aumentando a previsibilidade do processo de desenvolvimento.

Na prática da entrega contínua, todo *commit* pode ser colocado em produção imediatamente, mas o *deployment* depende de uma decisão manual, geralmente tomada por um gerente de projeto ou responsável por *releases*. Essa abordagem é especialmente útil em sistemas que não podem adotar *deployment* contínuo, como aplicações desktop, móveis ou embarcadas, que exigem processos de instalação mais complexos e não transparentes para os usuários. Assim, a entrega contínua assegura que o software esteja sempre pronto para ser lançado, mas o momento da liberação é definido por fatores estratégicos, como demandas de mercado, planejamento interno ou necessidades específicas da organização.

Para viabilizar essa prática, utiliza-se com frequência o recurso de *feature flags* (ou *feature toggles*). Esses mecanismos permitem que funcionalidades em desenvolvimento sejam integradas ao código principal sem que sejam imediatamente disponibilizadas aos usuários. Com isso, evita-se liberar implementações parciais e mantém-se a estabilidade do sistema. Além disso, os *feature flags* possibilitam estratégias como *releases* canários, em que novas funcionalidades são liberadas gradualmente para pequenos grupos de usuários, e testes A/B, que permitem comparar versões distintas de uma mesma funcionalidade.

2.1.7 Métricas IC/EC

As métricas utilizadas neste trabalho foram obtidas diretamente dos registros de execução dos workflows dos projetos analisados, por meio da GitHub API. Essa interface permite acessar de forma estruturada os dados referentes às execuções, como status de

conclusão, tempo de início e término e demais informações relevantes. Para a coleta, foram desenvolvidos scripts específicos que consultam os endpoints da API e consolidam os resultados.

Essas métricas foram definidas com o propósito de fornecer evidências para responder à RQ2, que trata da eficácia dos *pipelines* em termos de desempenho e confiabilidade. Cada indicador foi selecionado para refletir aspectos específicos dessa eficácia: frequência de execuções, proporção de sucesso e falha, tempo médio de execução e ocorrência de reexecuções.

As métricas utilizadas neste trabalho dialogam com as métricas DORA (*DevOps Research and Assessment*), reconhecidas como padrão da indústria para avaliar o desempenho de equipes de desenvolvimento de software (CLOUD, 2024). Enquanto o *total_runs* aproxima-se da métrica de frequência de deploy, *success_rate* e *failure_rate* refletem a taxa de falhas de mudança. O *avg_duration_min* pode ser comparado ao lead time para mudanças, e o *reruns* guarda relação com o tempo médio de recuperação. Essa correspondência reforça a validade das métricas adotadas, permitindo que os resultados sejam interpretados em consonância com práticas consolidadas de engenharia de software e DevOps.

A seguir, são apresentadas as métricas utilizadas, acompanhadas de suas definições e métodos de cálculo.

2.1.7.1 Total de Execuções (total_runs)

O *total_runs* corresponde ao número total de execuções de *pipelines* concluídas em um determinado período de tempo. Essa métrica fornece uma visão quantitativa da atividade do projeto, indicando a frequência com que integrações e automações são realizadas.

Método de cálculo

Para cada execução registrada, verifica-se o campo *conclusion*, que é um atributo retornado pela GitHub API e indica o resultado final da execução do workflow. Esse campo pode assumir valores como *success* (execução bem-sucedida), *failure* (execução com falha), *cancelled* (execução interrompida), entre outros. Quando o campo possui um valor definido, significa que a execução foi finalizada. A contagem dessas ocorrências resulta no número total de execuções no período.

2.1.7.2 Taxa de Sucesso (success_rate)

A *success_rate* representa a proporção de execuções concluídas com sucesso em relação ao total de execuções realizadas em um período.

Método de cálculo

Considera-se uma execução bem-sucedida quando o campo conclusion possui o valor "*success*". A taxa é obtida pela razão entre o número de execuções com sucesso e o total de execuções.

$$success\_rate = \frac{execuções\_com\_sucesso}{total\_runs}$$

2.1.7.3 Taxa de Falha (failure_rate)

A *failure_rate* corresponde à proporção de execuções que foram concluídas com falha em relação ao total de execuções no período.

Método de cálculo

Considera-se uma execução com falha quando o campo *conclusion* possui o valor "*failure*". A taxa é calculada pela razão entre o número de execuções com falha e o total de execuções.

$$failure\_rate = \frac{execuções\_com\_falha}{total\_runs}$$

2.1.7.4 Tempo Médio de Execução (avg_duration_min)

O *avg_duration_min* expressa o tempo médio, em minutos, necessário para a conclusão das execuções que foram finalizadas com sucesso em um determinado período.

Método de cálculo

A duração de cada execução é obtida pela diferença entre o horário de início (*run_started_at* ou *created_at*) e o horário de término (*updated_at*). Para evitar distorções estatísticas, são consideradas apenas execuções:

- concluídas com sucesso;
- com duração maior que zero;
- com duração inferior a 24 horas.

2.1.7.5 Reexecuções (reruns)

O *reruns* corresponde ao número de *commits* que tiveram mais de uma execução dentro do mesmo mês.

Método de cálculo

Cada execução é vinculada ao identificador único do commit (*head_sha*). Para cada mês, contabiliza-se a frequência de ocorrência de cada SHA. Sempre que um mesmo SHA aparece mais de uma vez, considera-se que houve reexecução para aquele *commit*.

2.2 Trabalhos correlatos

Pesquisas recentes têm buscado compreender como ferramentas de automação influenciam os processos de colaboração em projetos de software. Nesse contexto, destaca-se o estudo desenvolvido por Wessel et al. (2023), que analisou os efeitos da introdução de práticas automatizadas no fluxo de *pull requests*. O trabalho investigou como tais recursos impactam a interação entre desenvolvedores, afetando diretamente a qualidade das contribuições, a agilidade na integração de código e a dinâmica de revisão. Os resultados apresentados evidenciam que a automação contribui para reduzir barreiras técnicas e aumentar a eficiência das interações entre colaboradores. Além disso, demonstram que práticas automatizadas favorecem a confiabilidade do processo de integração, tornando o desenvolvimento mais ágil e organizado. Essa perspectiva reforça a importância de compreender não apenas os aspectos técnicos da automação, mas também seus efeitos sobre a colaboração e a produtividade em ambientes de código aberto.

Decan et al. (2022) realizaram uma análise em larga escala sobre o uso do GitHub Actions em mais de 68 mil repositórios, identificando que 43,9% deles já utilizavam workflows automatizados. O estudo mostra que a reutilização de Actions é prática comum, embora concentrada em um conjunto limitado de componentes, e discute aspectos de segurança e versionamento. A principal contribuição é oferecer uma visão panorâmica do ecossistema emergente do GitHub Actions, constituindo um primeiro passo para compreender seus impactos no desenvolvimento colaborativo.

Calefato, Lanubile e Quaranta (2022) investigaram práticas de MLOps (Operações de Machine Learning) em repositórios GitHub, com foco em workflows automatizados via GitHub Actions e CML. O CML (*Continuous Machine Learning*) é uma ferramenta de código aberto voltada à integração contínua em projetos de aprendizado de máquina, permitindo automatizar tarefas como treinamento de modelos, avaliação e geração de relatórios. Os autores identificaram baixa adoção de pipelines específicos para aprendizado de máquina, ressaltando desafios de automação em projetos que envolvem dados e modelos. O trabalho evidencia que a integração contínua em *Machine Learning* (aprendizado de máquina) precisa lidar não apenas com código, mas também com dados e parâmetros de modelos, o que torna os pipelines mais complexos.

Bernardo et al. (2024) analisaram 185 projetos open source (93 de Aprendizado de Máquina (ML) e 92 não-ML) para investigar diferenças na adoção de práticas de integração contínua. Os resultados indicam que projetos de ML apresentam maior duração de builds e menor cobertura de testes em comparação com projetos tradicionais. Além disso, identificaram que pequenos e médios projetos de ML exibem maior tendência de aumento na duração dos builds ao longo do tempo. A análise qualitativa revelou discussões específicas sobre falhas relacionadas a IC em ML, indicando desafios próprios desse domínio.

Ghaleb, Abduljalil e Hassan (2026) estudaram práticas de configuração de IC/EC em 2.557 aplicativos Android open source, abrangendo GitHub Actions, Travis CI, CircleCI e GitLab CI/CD. O trabalho mostra a ausência de padronização nas configurações, complexidade dos arquivos YML e necessidade de manutenção frequente. Os autores verificaram que a maioria das configurações concentra-se nas etapas de build, enquanto apenas 9% dos projetos incorporam processos de deploy. Além disso, observaram que as configurações são modificadas, em média, a cada dois meses, o que reflete o esforço contínuo de manutenção dos pipelines.

Considerando os trabalhos existentes, observa-se que o GitHub Actions tem sido amplamente estudado em diferentes domínios, como colaboração em *pull requests* Wessel et al. (2023), ecossistemas de software livre em geral Decan et al. (2022), aprendizado de máquina (Calefato, Lanubile e Quaranta (2022); Bernardo et al. (2024)) e aplicações móveis Ghaleb, Abduljalil e Hassan (2026). Contudo, nenhum deles aborda de forma sistemática os projetos RTOS FLOSS e suas particularidades, como compilação cruzada, monitoramento de binários e integração com hardware restrito. Este estudo contribui ao preencher essa lacuna, evidenciando como pipelines são estruturados em sistemas embarcados e de tempo real, destacando tanto práticas comuns quanto adaptações específicas necessárias nesse contexto.

3 Metodologia

Este capítulo apresenta os procedimentos adotados para o desenvolvimento da pesquisa. O estudo foi conduzido com base em dados públicos de repositórios de RTOS hospedados no GitHub, buscando compreender como esses projetos estruturam e operam seus *pipelines* de integração e entrega contínua.

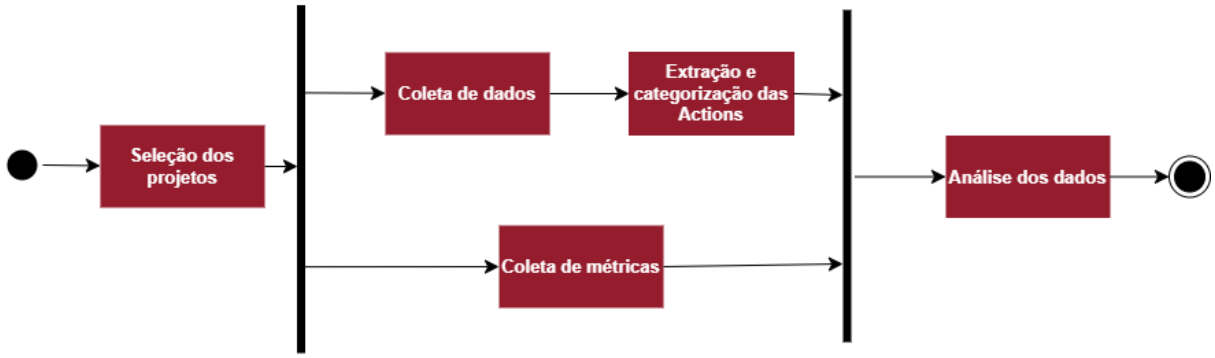

Figura 1 – Diagrama de atividades da metodologia

3.1 Seleção dos projetos

A primeira etapa consistiu na definição da amostra de projetos a serem analisados. Para garantir relevância, foi utilizado como critério de inclusão a popularidade dos repositórios, medida pelo número de estrelas atribuídas pela comunidade. Foram selecionados os projetos com mais de 1.000 estrelas.

A adoção desse critério se justifica pelo fato de que o número de estrelas em um repositório do GitHub não representa apenas uma métrica de popularidade superficial, mas também reflete aspectos como visibilidade, reputação e engajamento da comunidade. Estudos demonstram que o ato de atribuir estrelas está associado tanto ao reconhecimento da utilidade do projeto quanto ao interesse em acompanhar sua evolução, funcionando como um indicador de relevância dentro da plataforma Borges et al. (2016). Dessa forma, a escolha por repositórios com maior quantidade de estrelas aumenta a probabilidade de analisar pipelines de CI/CD mais estruturados e representativos.

A amostra final compreendeu os seguintes repositórios:

- aws/Amazon-FreeRTOS[1]
- contiki-os/Contiki[2]

[1] https://github.com/aws/Amazon-FreeRTOS
[2] https://github.com/contiki-os/Contiki

- contiki-ng/Contiki-NG[3]
- FreeRTOS/FreeRTOS[4]
- ARMmbed/mbed-os[5]
- apache/nuttx[6]
- RIOT-OS/RIOT[7]
- RT-Thread/rt-thread [8]
- zephyrproject-rtos/zephyr[9]

3.2 Coleta de dados

A coleta de dados foi realizada por meio da API pública do GitHub, utilizando o script **coleta_metadados.py**, responsável pela extração dos metadados dos repositórios. Para garantir acesso estruturado e seguro às informações, foi utilizada a biblioteca PyGithub[10], que fornece uma interface em Python para interação com a API. O processo de coleta de dados na API do GitHub exige autenticação para garantir acesso confiável e evitar restrições impostas a requisições anônimas. Para isso, é necessário gerar um *personal access token* diretamente na plataforma GitHub, o qual funciona como uma credencial única vinculada à conta do usuário. Esse token assegura maior segurança e permite realizar um número maior de requisições sem sofrer limitações de taxa (*rate limits*). Após a autenticação com o token, o script percorreu os repositórios selecionados e coletou atributos como identificação do projeto, linguagem principal, indicadores de popularidade (estrelas, *forks*, *watchers*), atividade (*issues* abertas, data de criação e atualização) e métricas de colaboração (*commits*, *pull requests*, número de contribuidores). Esses dados foram consolidados e armazenados para posterior análise.

Todos os scripts encontram-se disponíveis no repositório[11] do projeto, organizado em um diretório no Google Drive, garantindo transparência e possibilidade de reprodução dos resultados. O período considerado para a coleta compreendeu de 1º de dezembro de 2024 a 31 de dezembro de 2025, totalizando 13 meses.

[3] https://github.com/contiki-ng/Contiki-NG
[4] https://github.com/freeRTOS/FreeRTOS
[5] https://github.com/ARMmbed/mbed-os
[6] https://github.com/apache/nuttx
[7] https://github.com/RIOT-OS/RIOT
[8] https://github.com/RT-Thread/rt-thread
[9] https://github.com/zephyrproject-rtos/zephyr
[10] https://pygithub.readthedocs.io/en/stable/index.html
[11] https://drive.google.com/drive/folders/1d3DSJkv0cke21QD4er-FBnwRGHm_Vf2B?usp=drive_link

3.3 Extração e categorização das Actions

A etapa seguinte consistiu na coleta dos arquivos YAML de configuração dos workflows, realizada por meio do script **coleta_workflows.py**. Após essa coleta, foi feita a identificação manual das Actions e sua categorização segundo três critérios:

- Nome da *Action*.
- Tags associadas no GitHub Marketplace.
- Frequência de utilização nos projetos analisados.

Essa categorização permitiu identificar padrões de automação e práticas recorrentes entre os projetos RTOS.

3.4 Métricas de CI/CD

Para avaliar a eficácia operacional dos *pipelines*, as métricas foram coletadas por meio do script **coleta_metricasCICD.py**. As métricas consideradas foram: *total_runs, success_rate, failure_rate, avg_duration_min* e *reruns*.

3.5 Análise dos dados

Os dados coletados foram organizados e analisados de forma quantitativa e descritiva. As informações foram apresentadas em gráficos e tabelas, permitindo a comparação entre os projetos e a interpretação dos indicadores de desempenho e confiabilidade.

Essa análise possibilitou compreender tanto a estrutura dos *workflows* quanto o comportamento operacional dos *pipelines*, oferecendo uma visão abrangente sobre o uso do GitHub Actions em projetos RTOS de código aberto.

4 Resultados

4.1 Estrutura dos pipelines

Os resultados apresentados nesta seção estão diretamente relacionados à RQ1, que trata da forma como os projetos estruturam seus *pipelines* no GitHub Actions. A análise foi conduzida a partir da coleta e sistematização dos arquivos de configuração dos *workflows* presentes nos repositórios, permitindo identificar os elementos que compõem esses pipelines e os padrões adotados em diferentes projetos.

O objetivo aqui é compreender como os *workflows* são organizados e quais práticas se destacam em termos de acionamento, execução e manutenção. Para isso, foram observados os eventos que disparam os *workflows*, as ações utilizadas e a frequência com que determinados padrões se repetem entre os projetos. Essa abordagem possibilita avaliar não apenas a diversidade de configurações, mas também o grau de padronização existente entre iniciativas distintas.

A seguir, são apresentados os resultados que descrevem a forma como os *pipelines* analisados são estruturados no GitHub Actions. Essa análise contempla três dimensões principais: os eventos responsáveis pelo acionamento dos *workflows*, a categorização das *Actions* utilizadas e a frequência com que determinadas *Actions* aparecem nos *steps* (passos) dos *pipelines*.

4.1.1 Eventos de acionamento dos workflows

A Tabela 1 apresenta os 12 eventos identificados nos repositórios analisados que podem acionar *workflows* no GitHub Actions. Os resultados evidenciam a predominância dos eventos *push* e *pull_request*, presentes em todos os repositórios e responsáveis por 21,3% e 25,0% dos *workflows*, respectivamente. Essa predominância demonstra que os *pipelines* são majoritariamente ativados por alterações diretas no código ou por solicitações de integração, o que está em conformidade com práticas consolidadas de desenvolvimento contínuo.

Além desses, o evento *repository_dispatch* também aparece em 100% dos repositórios, representando 16,0% dos *workflows*. Esse resultado indica que os projetos fazem uso de chamadas externas para acionar *pipelines*, ampliando a flexibilidade da automação. O evento *schedule*, presente em 62,5% dos repositórios (13,3% dos *workflows*), evidencia o uso de execuções programadas, recurso importante para tarefas recorrentes, como testes periódicos ou verificações de segurança.

Eventos menos frequentes, como *workflow_dispatch, workflow_call* e *pull_request_target*, aparecem em proporções reduzidas, mas demonstram diversidade nas estratégias de acionamento. Essa variedade sugere que, embora exista um núcleo de práticas comuns, cada projeto adapta seus *workflows* às necessidades específicas de sua comunidade e arquitetura.

Tabela 1 – Distribuição dos eventos de acionamento dos workflows nos projetos

Evento	**% Workflows**	**% Repositórios**
pull_request	25,0	100,0
push	21,3	100,0
repository_dispatch	16,0	100,0
schedule	13,3	62,5
pull_request_target	6,4	50,0
issues	2,1	37,5
workflow_call	1,1	25,0
merge_group	3,2	12,5
workflow_run	2,1	12,5
workflow_dispatch	1,1	12,5
pull_request_review	0,5	12,5
branch_protection_rule	0,5	12,5

Fonte: Elaborado pelo autor.

Cada evento corresponde a uma atividade específica registrada pela plataforma e desempenha um papel distinto na execução dos *pipelines*. A seguir, descrevemos brevemente suas funções:

Tabela 2 – Eventos do GitHub Actions e suas funções

Evento	**Descrição**
pull_request	Aciona workflows quando uma *pull request* é aberta, atualizada ou sincronizada. Uma *pull request* é uma solicitação para que alterações feitas em um *branch* sejam revisadas e integradas ao *branch* principal do projeto.
push	Dispara workflows quando há envio (*push*) de commits para um *branch*.
repository_dispatch	Permite acionar workflows de forma manual ou externa, por meio de eventos personalizados enviados via API.
schedule	Executa workflows em horários pré-definidos, utilizando sintaxe cron. A sintaxe cron usa números e símbolos para indicar minutos, horas, dias e meses em que a tarefa deve ser executada.
pull_request_target	Similar ao *pull_request*, mas executa no contexto do *branch* base da PR. Um *branch* é uma linha de desenvolvimento paralela dentro do repositório.
issues	Aciona workflows quando uma *issue* é aberta, editada ou fechada. Uma *issue* é usada para relatar problemas, sugerir melhorias ou organizar tarefas.
workflow_call	Permite que um workflow seja chamado por outro, promovendo modularidade e reutilização.
merge_group	Dispara workflows quando um grupo de *pull requests* é preparado para merge em conjunto. O merge integra alterações de um branch em outro.
workflow_run	Executa workflows em resposta à conclusão de outro workflow, possibilitando encadeamento de processos.
workflow_dispatch	Permite disparar workflows manualmente pela interface do GitHub ou via API, com possibilidade de fornecer parâmetros personalizados.
pull_request_review	Aciona workflows quando uma revisão de *pull request* é submetida, aprovada ou comentada. A revisão é o processo de avaliar o código antes de integrá-lo ao branch principal.
branch_protection_rule	Dispara workflows quando regras de proteção de branch são criadas, alteradas ou removidas. Essas regras definem políticas de segurança, como exigir revisões antes de permitir merges em branches críticos.

4.1.2 Categorias das Actions

A distribuição das ações por categoria está apresentada na Tabela 3. A categoria *utilities* concentra 51,2% das Actions identificadas, evidenciando que grande parte dos *workflows* está voltada à execução de tarefas auxiliares, como manipulação de arquivos, controle de fluxo e operações genéricas.

Além da predominância da categoria *utilities*, observa-se que aproximadamente

Tabela 3 – Categorias das Actions utilizadas nos workflows

Categoria	**% Actions**
utilities	51,2
continuous-integration	14,0
dependency-management	13,2
uncategorized	12,0
deployment	2,7
container-ci	1,6
open-source-management	1,6
security	1,2
reporting	1,2
code-review	0,8
code-quality	0,4
project-management	0,4

Fonte: Elaborado pelo autor.

27,6% das Actions empregadas estão diretamente relacionadas a testes, compilação e análise de código, distribuídas entre as categorias *continuous-integration* (14,0%), *dependency-management* (13,2%) e *code-quality* (0,4%).

A categoria *dependency-management* corresponde a 13,2%, refletindo o uso de ações voltadas ao armazenamento, compartilhamento e reaproveitamento de artefatos, além da utilização de cache para otimizar a execução dos workflows.

A presença de *uncategorized* (12,0%) sugere o uso de soluções personalizadas ou menos documentadas, o que pode indicar flexibilidade, mas também desafios relacionados à padronização e rastreabilidade.

Outras categorias aparecem em proporções menores, mas revelam preocupações complementares. *Open-source-management* e *reporting*, ambas com 1,6% e 1,2% respectivamente, indicam práticas voltadas à gestão de projetos de código aberto e ao monitoramento de métricas.

A análise das categorias menos presentes nos pipelines evidencia que *deployment* (2,7%), *container-ci* (1,6%), *security* (1,2%) e *code-review* (0,8%) possuem participação bastante limitada. Apesar de os projetos de RTOS FLOSS demonstrarem uma estrutura consistente em tarefas auxiliares, integração e gerenciamento de dependências, observa-se que algumas dimensões permanecem pouco exploradas. O uso restrito dessas categorias sugere que práticas fundamentais para a confiabilidade e evolução dos sistemas, como entrega automatizada, segurança integrada, e revisão sistemática, ainda ocupam papel secundário ou não foram plenamente ajustadas às especificidades dos sistemas de tempo real. Essa limitação pode estar relacionada a fatores como a complexidade técnica da automação em ambientes embarcados, a priorização de aspectos considerados mais críticos para a estabilidade imediata do sistema ou ainda à ausência de ferramentas específicas que

atendam às exigências de sistemas de tempo real, como suporte a testes determinísticos e integração com hardware especializado.

Esse cenário aponta para uma lacuna, que pode ser compreendida como uma oportunidade de evolução. A ampliação do uso dessas categorias nos pipelines tem potencial para fortalecer a rastreabilidade e a robustez dos projetos, além de elevar seu nível de maturidade. Dessa forma, seria possível alinhar de maneira mais consistente as práticas de desenvolvimento às exigências de ambientes críticos, nos quais previsibilidade e estabilidade constituem requisitos fundamentais.

Cada categoria reflete o tipo de tarefa que as Actions desempenham nos *workflows*:

utilities – Ações voltadas a tarefas auxiliares e genéricas, como manipulação de arquivos, controle de fluxo e operações de suporte.

continuous-integration – Ações relacionadas à execução de testes, *builds* e validações automáticas durante o processo de integração contínua.

dependency-management – Ações que instalam, atualizam ou verificam dependências de software, garantindo que bibliotecas externas estejam corretas e atualizadas.

uncategorized – Ações que não possuem classificação definida no Marketplace, mas que ainda podem ser utilizadas nos workflows.

deployment – Ações voltadas ao processo de entrega e publicação de aplicações em ambientes de produção ou teste.

container-ci – Ações específicas para criação, gerenciamento e execução de containers, geralmente em ambientes de integração contínua.

open-source-management – Ações que auxiliam na gestão de projetos de código aberto, como automação de licenças ou manutenção de contribuições.

security – Ações voltadas à análise de vulnerabilidades, verificação de segurança e conformidade de código.

reporting – Ações que geram relatórios de execução, métricas ou resultados de testes, facilitando o acompanhamento dos *pipelines*.

code-review – Ações que apoiam o processo de revisão de código, automatizando verificações ou comentários em *pull requests*.

code-quality – Ações que avaliam a qualidade do código, verificando padrões, estilo e boas práticas de programação.

project-management – Ações que integram os *workflows* com ferramentas de gestão de projetos, como organização de tarefas e acompanhamento de progresso.

4.1.3 Frequência de uso das Actions

A frequência de utilização das Actions está detalhada na Tabela 4. A ação *checkout* é a mais recorrente, utilizada em 34,1% dos *steps* e presente em todos os repositórios, confirmando sua função essencial de disponibilizar o código-fonte para execução das etapas subsequentes.

Em seguida, destacam-se *setup-python* (12,9% dos *steps*, em 87,5% dos repositórios) e *upload-artifact* (11,5% dos *steps*, também em 87,5% dos repositórios), evidenciando a importância da configuração de ambientes de desenvolvimento e do armazenamento de artefatos gerados durante o processo de integração.

Outras Actions com presença relevante incluem *download-artifact* (3,4% dos *steps*, em 50% dos repositórios) e *executable-monitor* (5,2% dos *steps*, em 12,5% dos repositórios). Esta última merece destaque por estar voltada ao monitoramento de binários, aspecto particularmente relevante em projetos embarcados, uma vez que o binário gerado corresponde ao firmware final que será gravado no dispositivo. O acompanhamento desse artefato garante integridade, rastreabilidade e reprodutibilidade, além de permitir validações em hardware real.

Embora a maioria das Actions esteja concentrada em tarefas utilitárias e de integração contínua, foram identificadas também Actions específicas para o contexto embarcado e de IoT, como *arm-none-eabi-gcc-action*, *setup-qemu-action*, *action-zephyr-setup*, *riot-action* e *localhost-mqtt-broker*. Apesar de representarem menos de 2% dos steps, as Actions específicas para sistemas embarcados e IoT sugerem algum nível de especialização nos pipelines analisados, voltado a demandas como compilação de firmware e simulação em hardware. Esse resultado pode ser contrastado com o estudo de Calefato, Lanubile e Quaranta (2022), que identificaram escassa evidência de pipelines especializados em projetos de aprendizado de máquina, nos quais predominam tarefas genéricas de integração contínua. É possível que essa diferença decorra das particularidades de cada domínio, enquanto sistemas embarcados exigem suporte direto a hardware e firmware, em MLOps fatores como a complexidade do re-treinamento de modelos ou a prevalência de ambientes privados podem limitar a adoção de workflows especializados em repositórios públicos.

Foram identificadas 68 Actions distintas nos projetos analisados, abrangendo desde ações essenciais e amplamente difundidas até funções altamente específicas voltadas a compiladores, segurança, containers e monitoramento de binários. Essa diversidade demonstra que, embora exista um núcleo comum de práticas consolidadas, cada projeto adapta seus workflows às necessidades particulares de seu contexto técnico. Essa heterogeneidade reflete tanto a complexidade dos sistemas embarcados quanto a flexibilidade oferecida pelo GitHub Actions para integrar diferentes ferramentas e processos em um mesmo pipeline.

Tabela 4 – Frequência de uso das principais Actions nos repositórios

Action	**% Steps**	**% Repositórios**
checkout	34,1	100,0
setup-python	12,9	87,5
upload-artifact	11,5	87,5
download-artifact	3,4	50,0
cache	1,3	37,5
labeler	0,8	37,5
codeql-action	1,6	25,0
setup-buildx-action	1,0	25,0
login-action	1,0	25,0
build-push-action	1,0	25,0
github-script	0,5	25,0
executable-monitor	5,2	12,5
setup-msbuild	2,1	12,5
configure-aws-credentials	1,6	12,5
setup-qemu-action	1,3	12,5
changed-files	1,3	12,5
arch	1,0	12,5
action-download-artifact	1,0	12,5
action-zephyr-setup	1,0	12,5
docker-run-action	0,8	12,5
publish-unit-test-result-action	0,8	12,5
spellings	0,5	12,5
formatting	0,5	12,5
ssl-credential-creator	0,5	12,5
localhost-echo-server	0,5	12,5
uuid-action	0,5	12,5
setup-node	0,5	12,5
ssh-agent	0,5	12,5
alls-green	0,5	12,5
ccache-action	0,3	12,5
setup-java	0,3	12,5
filter-sarif	0,3	12,5
arm-none-eabi-gcc-action	0,3	12,5
artifact-backup	0,3	12,5
set_up_cbmc_runner	0,3	12,5
run_cbmc	0,3	12,5

Continua na próxima página

Continuação da Tabela 4

Action	**% Steps**	**% Repositórios**
manifest-verifier	0,3	12,5
doxygen	0,3	12,5
gh-action-get-changed-files	0,3	12,5
localhost-mqtt-broker	0,3	12,5
ci-container	0,3	12,5
setup-msys2	0,3	12,5
free-disk-space	0,3	12,5
pr-size-labeler	0,3	12,5
super-linter	0,3	12,5
check-labels-action	0,3	12,5
Actions-gh-pages	0,3	12,5
riot-action	0,3	12,5
upload-pages-artifact	0,3	12,5
deploy-pages	0,3	12,5
action-commit-push	0,3	12,5
action-pull-request	0,3	12,5
pkgs-action	0,3	12,5
action-backport	0,3	12,5
codecov-action	0,3	12,5
ready-to-merge	0,3	12,5
check-valid-pr	0,3	12,5
action-first-interaction	0,3	12,5
summarize-issues	0,3	12,5
action_scancode	0,3	12,5
action-manifest	0,3	12,5
github-Actions-ensure-sha-pinned-Actions	0,3	12,5
reuse-action	0,3	12,5
create-release	0,3	12,5
upload-release-asset	0,3	12,5
scorecard-action	0,3	12,5
stale	0,3	12,5
delete-old-workflow-runs	0,3	12,5

Fonte: Elaborado pelo autor.

A análise dos pipelines evidencia que os projetos RTOS FLOSS utilizam o GitHub Actions de maneira estruturada e consistente, combinando práticas comuns de integração contínua com adaptações específicas às suas necessidades. Os workflows são acionados

principalmente por eventos ligados à evolução do código, como push e pull request, mas também incorporam execuções programadas e disparos externos para ampliar a automação. No uso das Actions, observa-se predominância de tarefas utilitárias e de configuração de ambientes, complementadas por ações voltadas à gestão de dependências, segurança e suporte a containers. A frequência de utilização mostra que algumas Actions, como checkout, setup-python e upload-artifact, são centrais para o funcionamento dos pipelines. Em conjunto, o levantamento de 68 Actions distintas demonstra que, embora exista um núcleo de práticas recorrentes, cada projeto molda seus workflows de acordo com seu contexto técnico, revelando tanto padronização quanto diversidade na forma de estruturar a automação.

Além das Actions de caráter mais genérico, como configuração de ambientes e armazenamento de artefatos, também foram identificadas outras diretamente relacionadas ao contexto embarcado e de IoT. Entre elas, destacam-se *setup-qemu-action*, voltada à emulação de hardware; *arch* e *arm-none-eabi-gcc-action*, associadas à compilação para arquiteturas embarcadas como ARM Cortex-M; *action-zephyr-setup*, destinada à configuração do Zephyr RTOS; *localhost-mqtt-broker*, que viabiliza a comunicação por meio do protocolo MQTT, amplamente utilizado em aplicações IoT; e *riot-action*, voltada ao RIOT OS, outro sistema operacional de tempo real para dispositivos com recursos limitados. A presença dessas actions evidencia que os pipelines analisados não apenas incorporam práticas comuns de integração contínua, mas também se adaptam às necessidades específicas de sistemas embarcados e IoT, reforçando a diversidade e especialização das automações observadas.

4.2 Avaliação da eficácia operacional dos pipelines

Nesta seção, buscamos responder à RQ2. A análise parte da observação de métricas que permitem avaliar a eficácia operacional dos pipelines, considerando aspectos como volume de execuções, taxas de falha e sucesso e tempo médio de execução. Esses indicadores fornecem uma visão sobre o desempenho e a estabilidade dos workflows, permitindo identificar tanto práticas consolidadas quanto fragilidades que impactam a confiabilidade da automação.

O total de execuções mensais, por exemplo, evidencia diferentes níveis de maturidade e adoção entre os projetos, refletindo a intensidade de uso dos pipelines. Já as taxas de falha e sucesso permitem avaliar a robustez dos processos, indicando se os workflows conseguem sustentar a integração contínua sem interrupções frequentes. Por fim, o tempo médio de execução revela a eficiência dos pipelines, mostrando se eles conseguem entregar resultados dentro de prazos aceitáveis para o desenvolvimento embarcado.

Ao reunir essas métricas, torna-se possível construir uma visão completa sobre o

funcionamento dos pipelines, considerando tanto o desempenho quanto a estabilidade. Essa análise evidencia os pontos fortes já consolidados, como práticas que garantem confiabilidade e eficiência, mas também revela limitações que ainda precisam ser enfrentadas para que a automação alcance níveis mais elevados de maturidade e consistência.

4.2.1 Total de execuções

A métrica de total de execuções representa o volume mensal de atividades dos pipelines nos projetos analisados. Conforme ilustrado na Figura 2, observa-se uma disparidade significativa entre os projetos, evidenciando diferentes níveis de maturidade, adoção e suporte institucional.

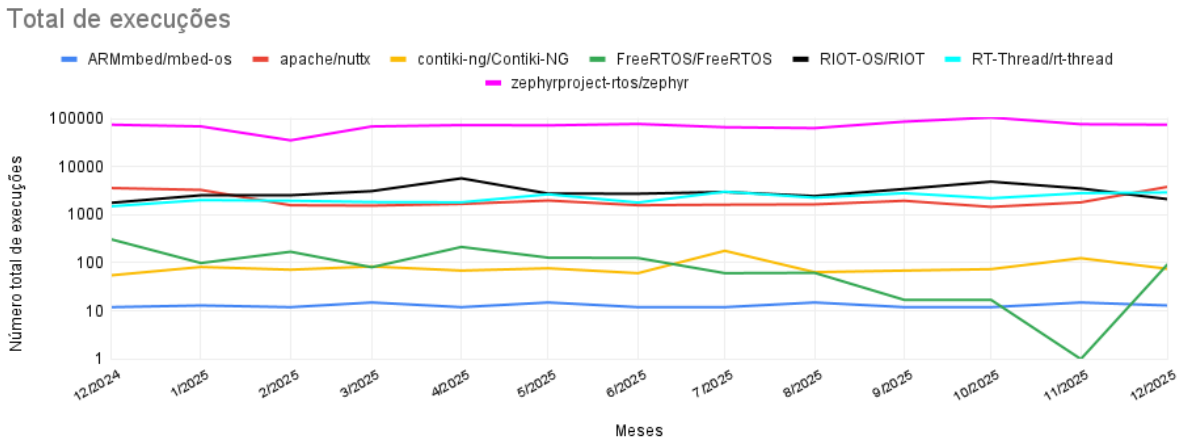

Figura 2 – Total de execuções

O projeto zephyr apresenta, de forma consistente, o maior número de execuções mensais, com valores que ultrapassam 100 mil execuções em determinados períodos. Este comportamento pode estar diretamente relacionado ao seu modelo de governança robusto, mantido pela Linux Foundation, e ao envolvimento de grandes corporações como Intel, Nordic Semiconductor e Texas Instruments. A elevada frequência de execuções sugere um pipeline altamente automatizado, com ampla cobertura de testes e integração contínua em larga escala.

Em segundo plano, destacam-se os projetos RIOT e rt-thread, que mantêm volumes mensais entre 2 mil e 5 mil execuções. Ambos são desenvolvidos por comunidades abertas, com participação de instituições acadêmicas e empresas, o que contribui para uma atividade constante e significativa.

Projetos como nuttx e FreeRTOS apresentam variações mensais mais acentuadas, com picos e quedas que podem estar associados a ciclos de desenvolvimento específicos ou à execução de testes em massa em determinados períodos. Já os projetos Contiki-NG e mbed-os apresentam os menores volumes de execução, com médias mensais inferiores a 100

execuções. No caso do mbed-os, esse comportamento está alinhado ao seu status de fim de vida[1], uma vez que a Arm anunciou em 2024 que ele deixará de receber manutenção e suporte, com descontinuação definitiva prevista para julho de 2026. Desde então, observa-se uma redução significativa na atividade da comunidade e na execução dos pipelines, refletindo a migração gradual para outras soluções mais ativas. Já o Contiki-NG possui um escopo mais específico, tendo como foco dispositivos de baixo consumo e redes de sensores, áreas de aplicação bastante específicas dentro do universo de sistemas embarcados. Diferente de projetos como Zephyr ou FreeRTOS, que contam com ampla adoção industrial e comercial, o Contiki-NG não possui uma comunidade de desenvolvedores tão extensa. Como resultado, sua atividade se concentra em nichos de pesquisa e experimentação, refletindo menor intensidade de contribuições e execução de testes automatizados.

4.2.2 Taxa de sucesso

A taxa de sucesso representa a proporção de execuções de pipelines concluídas com êxito em relação ao total de execuções mensais. Esta métrica é um dos principais indicadores de confiabilidade e estabilidade dos workflows. A Figura **??** apresenta a evolução mensal dessa métrica para os sete projetos analisados.

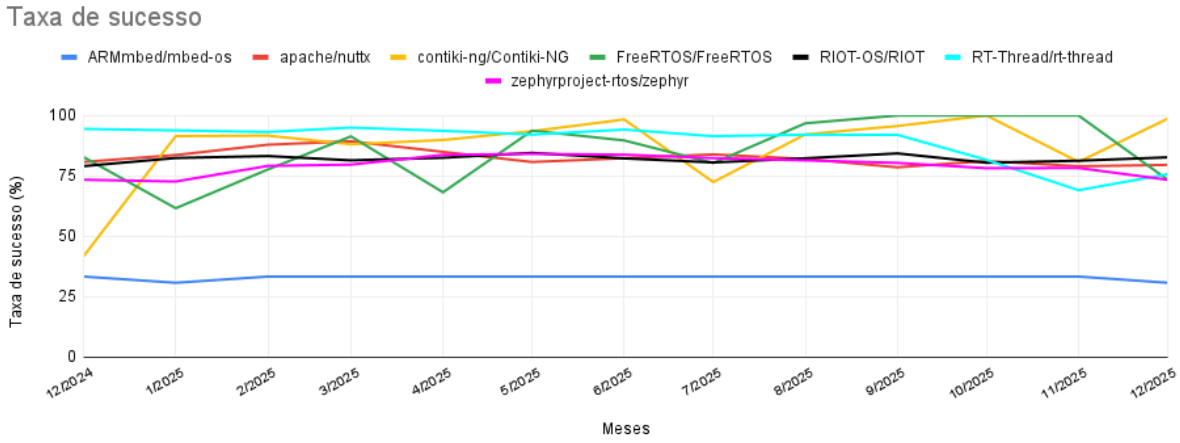

Figura 3 – Taxa de sucesso

Observa-se que a maioria dos projetos mantém taxas de sucesso superiores a 75% ao longo do período, com destaque para rt-thread, que apresenta desempenho superior a 90% em diversos meses, especialmente entre dezembro de 2024 e agosto de 2025. Esse resultado evidencia a maturidade do pipeline e a consistência dos testes automatizados, refletindo um ambiente de integração contínua bem estruturado.

RIOT também apresenta estabilidade, com taxas variando entre 79% e 84%, o que indica um bom equilíbrio entre confiabilidade e volume de execuções. FreeRTOS, embora

[1] https://forums.mbed.com/t/important-update-on-mbed-end-of-life/23644

apresente flutuações mais acentuadas, mantém taxas superiores a 80% na maior parte do período, com picos de 100% em três meses consecutivos (setembro a novembro de 2025), sugerindo melhorias pontuais na infraestrutura ou nos testes.

O projeto zephyr, apesar de seu elevado volume de execuções, mantém taxas de sucesso entre 73% e 84%. Já o projeto nuttx apresenta desempenho estável, com taxas entre 79% e 89%, reforçando sua confiabilidade como projeto mantido pela Apache Software Foundation.

Contiki-NG apresenta oscilações mais evidentes, com taxas que variam de 41,82% (dezembro de 2024) a 100% (outubro de 2025), o que pode indicar instabilidade pontual ou variações na cobertura de testes. Por fim, mbed-os apresenta taxa de sucesso fixada em 33,33% na maior parte do período, refletindo falhas recorrentes e possível descontinuidade de manutenção, dado seu status de fim ciclo de vida.

4.2.3 Taxa de falha

Complementar à taxa de sucesso, a taxa de falha representa a proporção de execuções que não foram concluídas com êxito. A Figura 4 apresenta a evolução mensal dessa métrica para os projetos analisados, no mesmo intervalo temporal.

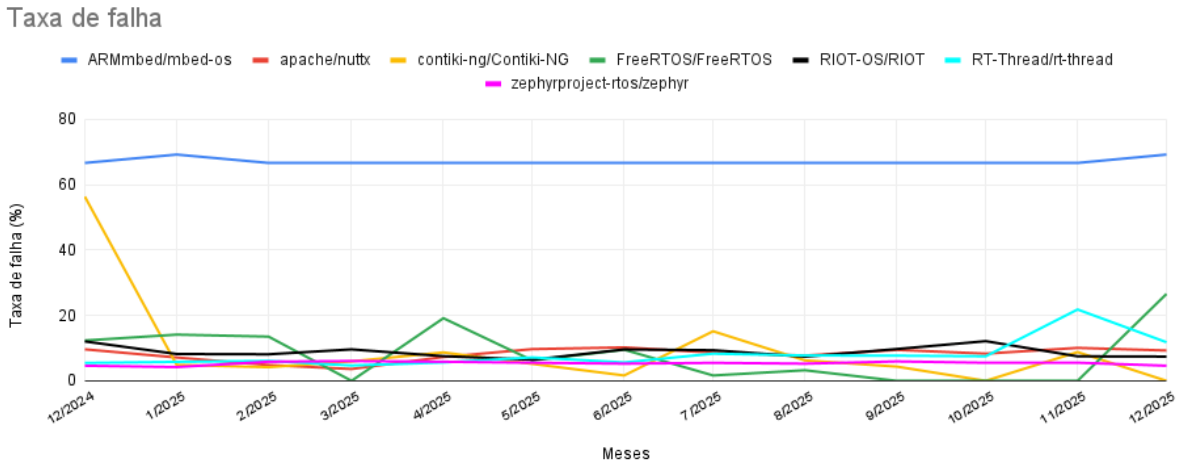

Figura 4 – Taxa de falha

O projeto mbed-os apresenta, de forma consistente, as maiores taxas de falha, oscilando entre 66,67% e 69,23% ao longo de todo o período. Este resultado é preocupante e reforça a hipótese de abandono do projeto, com falhas sistemáticas nos pipelines e ausência de correções estruturais.

Em contraste, rt-thread mantém taxas de falha inferiores a 10% na maior parte do período, com exceção de novembro de 2025, quando atinge 21,83%. Essa elevação pontual pode estar associada a mudanças significativas no código ou a instabilidades temporárias

na infraestrutura. Zephyr também apresenta desempenho positivo, com taxas entre 4,62% e 6,05%, o que é notável considerando seu elevado volume de execuções.

RIOT mantém taxas entre 6% e 12%, com variações moderadas e sem picos críticos, o que indica estabilidade operacional. Nuttx apresenta comportamento semelhante, com taxas entre 3,62% e 10,08%, reforçando sua confiabilidade como projeto institucional.

O Contiki-NG apresenta oscilações mais acentuadas, com taxa de falha de 56,36% em dezembro de 2024, seguida por uma queda significativa nos meses subsequentes. Esse aumento esteve associado a falhas em um workflow específico, cuja correção resultou em redução consistente da taxa de falha. A partir desse ponto, observa-se um comportamento mais estável e controlado nos processos de execução.

FreeRTOS apresenta comportamento irregular, com taxas que variam de 0% (março, setembro, outubro e novembro de 2025) a 26,6% (dezembro de 2025). Essa instabilidade pode indicar problemas intermitentes na infraestrutura ou na configuração dos *pipelines*.

4.2.4 Reexecuções

A Figura 5 apresenta o número absoluto de *commits* que resultaram em múltiplas execuções de pipeline. Essa métrica pode refletir diferentes situações: por um lado, pode indicar instabilidade, presença de testes intermitentes (*flaky tests*), falhas temporárias de infraestrutura ou necessidade de reprocessamento manual; por outro, também pode estar associada a estratégias deliberadas de validação intensiva, em que múltiplas tentativas são realizadas para assegurar a confiabilidade dos *builds*.

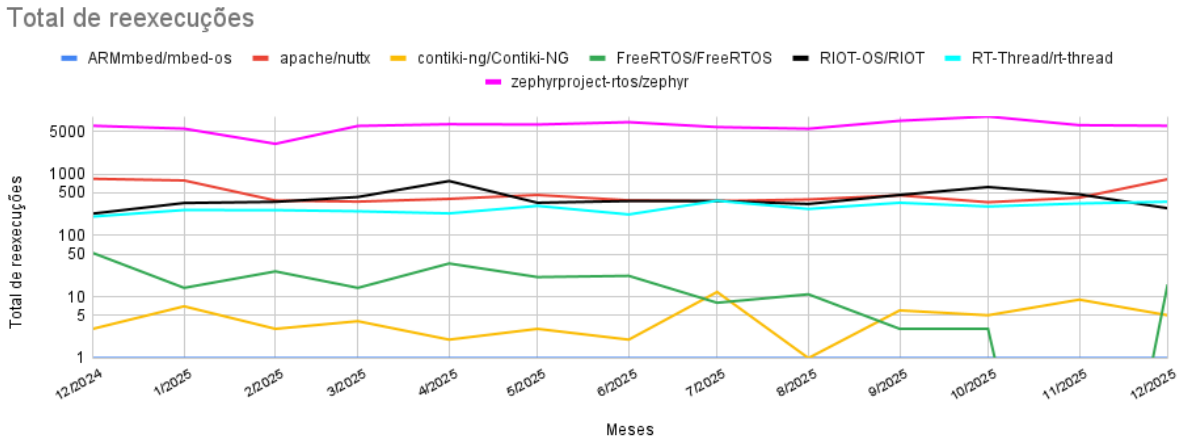

Figura 5 – Total de reexecuções

Embora o gráfico absoluto seja útil para evidenciar o volume bruto de atividade, ele pode gerar interpretações enviesadas, já que projetos com maior número de execuções tendem naturalmente a apresentar mais reexecuções. Para mitigar esse efeito, foi calculada a **taxa de reexecuções**, definida como a proporção de reexecuções em relação ao total de

execuções mensais. A Figura 6 apresenta essa métrica em termos percentuais, permitindo comparações mais equilibradas entre projetos de diferentes tamanhos.

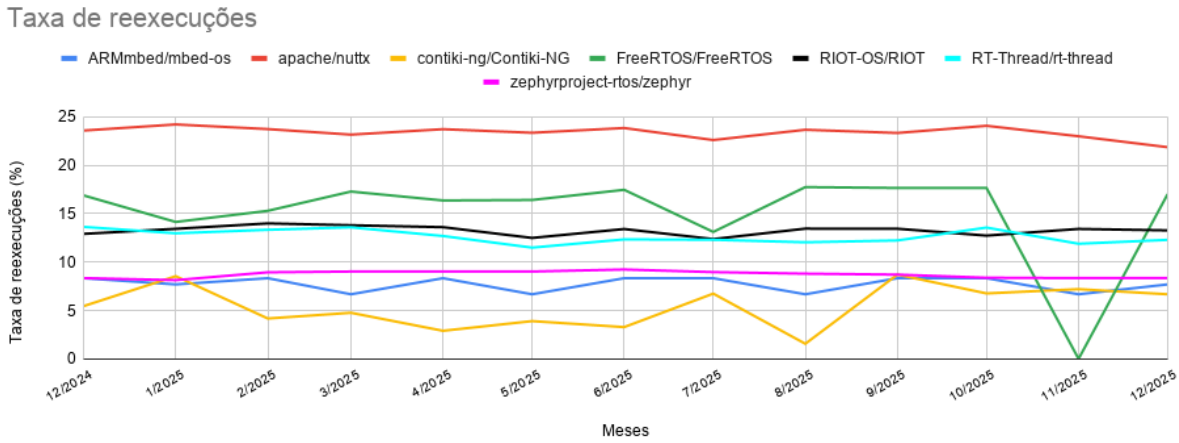

Figura 6 – Taxa de reexecuções (%)

Os resultados mostram que o nuttx apresenta as maiores taxas proporcionais de reexecução, variando entre 21% e 24% ao longo do período, o que sugere maior incidência relativa de instabilidades ou necessidade de reprocessamento. O zephyr, apesar dos altos valores absolutos, mantém taxas baixas e estáveis (entre 8% e 9%), indicando que seu elevado número de reexecuções está mais associado ao grande volume de atividade do que à falta de confiabilidade. Projetos como RIOT e rt-thread apresentam taxas intermediárias, geralmente entre 12% e 14%, com oscilações pontuais. O FreeRTOS mostra taxas variando entre 13% e 17%, com uma queda abrupta em novembro de 2025 (0%), possivelmente relacionada a ausência de execuções registradas no período. O Contiki-NG apresenta taxas mais baixas e irregulares, entre 2% e 8%, refletindo menor atividade. Por fim, o mbed-os mantém taxas relativamente altas (entre 6% e 8%), o que, considerando seu baixo volume de execuções, amplifica o impacto relativo de cada reexecução.

Essa análise proporcional complementa a visão absoluta, permitindo identificar não apenas quais projetos possuem maior volume de reexecuções, mas também quais enfrentam maiores desafios relativos de estabilidade em seus pipelines.

4.2.5 Tempo médio de execução

A Figura 7 apresenta a média mensal de duração, em segundos, das execuções bem-sucedidas dos *pipelines*. Esta métrica é fundamental para avaliar a eficiência dos processos automatizados e o tempo de retorno de feedback para os desenvolvedores.

O projeto nuttx destaca-se com os maiores tempos médios de execução, oscilando entre 16 e 22 segundos ao longo do período analisado. Este resultado pode estar associado à complexidade dos testes, à quantidade de etapas nos workflows e à infraestrutura utilizada.

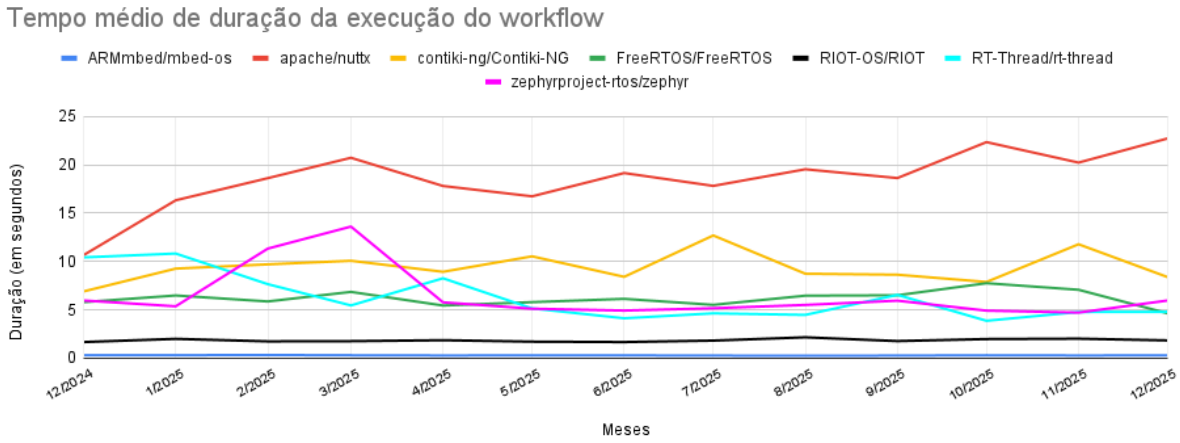

Figura 7 – Tempo médio de duração da execução do workflow

Essa duração pode impactar negativamente a agilidade dos ciclos de desenvolvimento, especialmente em ambientes que demandam respostas rápidas, como o GitHub Actions, onde o tempo de retorno imediato é essencial para que os desenvolvedores recebam feedback sobre falhas ou sucessos de forma contínua. Nesse contexto, pipelines mais longos podem comprometer a eficiência do processo de integração, mesmo quando oferecem maior rigor e segurança.

Por outro lado, projetos como rt-thread e FreeRTOS apresentam tempos médios significativamente menores, variando entre 4 e 7 segundos. Essa performance sugere pipelines mais enxutos, com menor carga de testes, ou então pipelines que, mesmo contendo diversas etapas, são executados de forma mais eficiente graças a otimizações na infraestrutura ou na configuração dos workflows. O projeto RIOT também se destaca positivamente, mantendo tempos médios entre 1,6 e 2 segundos, o que favorece ciclos de desenvolvimento rápidos e iterativos, com feedback quase imediato para os desenvolvedores.

O projeto mbed-os apresenta os menores tempos médios (cerca de 0,28 segundos), embora esse valor deva ser interpretado com cautela, dado o baixo volume de execuções e a alta taxa de falhas. Já o projeto zephyr mantém uma média estável entre 4,7 e 5,9 segundos, o que é notável considerando o elevado volume de execuções mensais.

A análise dessa métrica revela que projetos com maior apoio institucional tendem a apresentar tempos de execução otimizados, enquanto projetos com menor suporte ou em descontinuidade apresentam variações mais acentuadas ou tempos reduzidos, porém com baixa confiabilidade.

4.2.6 Média anual das métricas

As médias anuais apresentadas correspondem ao valor médio obtido a partir de todos os registros mensais coletados para cada métrica. Esse procedimento permite suavizar

variações pontuais e oferecer uma visão consolidada do comportamento dos projetos ao longo do período analisado. Ao considerar a média dos valores, é possível identificar padrões mais estáveis de desempenho e comparar de forma equilibrada aspectos como volume de execuções, frequência de reexecuções, duração dos workflows e taxas de sucesso e falha. Dessa maneira, as métricas anuais não apenas sintetizam os resultados observados, mas também fornecem uma base mais confiável para avaliar a maturidade e a consistência operacional dos pipelines de integração contínua entre os diferentes projetos.

No que se refere ao total de execuções anuais, o projeto Zephyr apresenta uma dominância absoluta, com média de 72.057 execuções, evidenciado na Figura 8. Esse volume é mais de 20 vezes superior ao segundo colocado, RIOT, que apresenta média de 3.108 execuções anuais. Essa diferença expressiva reflete o alto grau de automação, a ampla cobertura de testes e o suporte institucional robusto que o Zephyr recebe da Linux Foundation e de empresas parceiras.

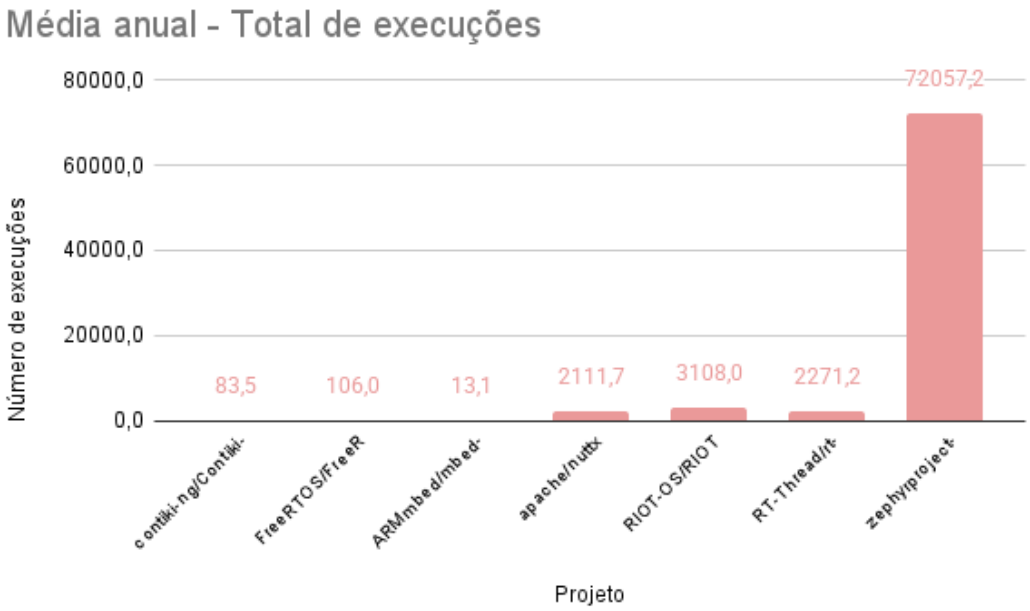

Figura 8 – Total de execuções

Em seguida, aparecem RT-Thread (2.271) e Nuttx (2.111), ambos com volumes significativos que indicam pipelines ativos e bem estruturados. FreeRTOS apresenta uma média mais modesta (106), enquanto Contiki-NG (83) e mbed-os (13) registram os menores volumes, refletindo menor atividade comunitária ou, no caso do mbed-os, o processo de descontinuidade anunciado pela Arm.

A métrica de reexecuções reforça a liderança do Zephyr, conforme a Figura 9, que apresenta uma média anual de 6.260 reexecuções. Esse número elevado, embora possa sugerir instabilidade, deve ser interpretado à luz do volume total de execuções, que também é extremamente alto. A alta taxa de reexecuções pode estar associada à execução de testes

em ambientes variados, à presença de testes intermitentes (*flaky tests*), que falham de forma não determinística e exigem repetição para confirmar resultados, ou ainda à estratégia de validação intensiva adotada pelo projeto, em que múltiplas rodadas de testes são realizadas deliberadamente para assegurar maior confiabilidade dos *builds*.

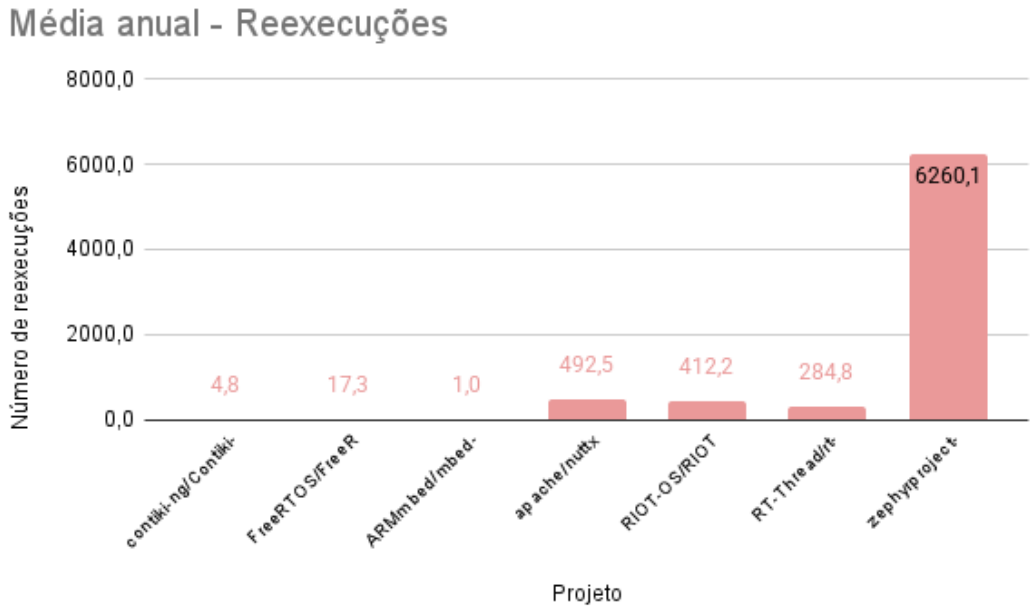

Figura 9 – Total de reexecuções

Nuttx (492), RIOT (412) e RT-Thread (285) também apresentam volumes relevantes, sugerindo pipelines ativos, porém com desafios de estabilidade. FreeRTOS (17), Contiki-NG (5) e mbed-os (1) mantêm valores baixos, o que pode estar relacionado à simplicidade dos workflows ou à limitação na cobertura de testes.

Quanto à duração média de execução dos workflows, apresentada na Figura 10, observa-se que o Nuttx apresenta o maior tempo anual, com 18,6 minutos, resultado que reflete a complexidade de seus processos e a quantidade de etapas configuradas. No caso do Nuttx, há arquivos YAML bastante extensos, como o *build.yaml* (392 linhas) e o *arch.yaml* (261 linhas), que indicam workflows com múltiplas fases de compilação e validação. Em seguida, aparece o Contiki-NG, com média de 9,4 minutos, que também possui arquivos relativamente grandes, como o *build.yaml* (236 linhas) e o *codeql.yaml* (146 linhas), sugerindo pipelines robustos e detalhados. O Zephyr mantém duração média de 6,5 minutos, compatível com seu elevado volume de execuções e a complexidade de sua infraestrutura. FreeRTOS e RT-Thread apresentam valores equilibrados, em torno de 6,2 minutos, resultado de workflows que combinam arquivos menores com outros bastante extensos, como o *freertos_demos.yaml* (637 linhas) e o *freertos_plus_demo.yaml* (1371 linhas), o que explica a duração intermediária observada. RIOT (1,8 minutos) e mbed-os (0,3 minutos) registram os menores tempos, favorecendo ciclos de desenvolvimento mais

ágeis, embora, no caso do ARMmbed, esse valor esteja associado à baixa confiabilidade e ao reduzido volume de execuções.

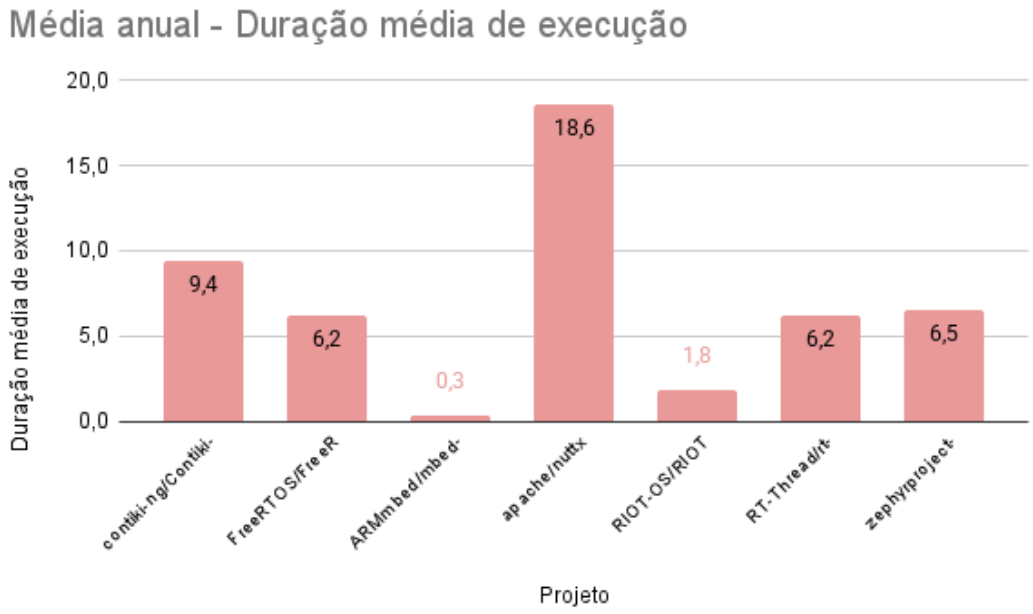

Figura 10 – Média anual - Duração média de execução do workflow

A média da taxa de sucesso revela que RT-Thread lidera com 89,1%, seguido por Contiki-NG (87,3%) e FreeRTOS (85,8%), conforme apresentado na Figura 11. Esses resultados indicam alta confiabilidade dos pipelines. RIOT (82,1%) e Nuttx (82,6%) também apresentam desempenho sólido. Já o Zephyr apresenta uma taxa média de sucesso de 79,3%, resultado que, embora inferior aos índices observados em outros projetos, deve ser interpretado à luz de seu elevado volume de execuções. A grande quantidade de pipelines processados amplia a probabilidade de ocorrência de falhas, o que naturalmente impacta o percentual de sucesso. Nesse sentido, o valor registrado não indica fragilidade estrutural, mas reflete os desafios inerentes à integração contínua em larga escala, em que a diversidade de arquiteturas e configurações testadas exige maior rigor e complexidade nos processos de validação. Assim, a taxa de sucesso do Zephyr evidencia tanto a robustez de sua infraestrutura quanto o esforço da comunidade em sustentar um ambiente de automação intensivo e altamente ativo. O mbed-os, por outro lado, apresenta a menor taxa de sucesso (32,9%), evidenciando falhas recorrentes e possível abandono de manutenção.

Complementarmente, a taxa de falha anual confirma os desafios enfrentados pelo ARMmbed, com 67,1% de falhas, contrastando com os demais projetos que mantêm índices entre 5% e 9%, conforme ilustrado na Figura 12. Zephyr (5,4%), RT-Thread (8,1%), FreeRTOS (8,2%) e Contiki-NG (9,3%) apresentam os menores valores, reforçando sua estabilidade operacional. Nuttx (8,2%) e RIOT (8,9%) mantêm taxas moderadas, compatíveis com seus volumes de execução e complexidade dos workflows.

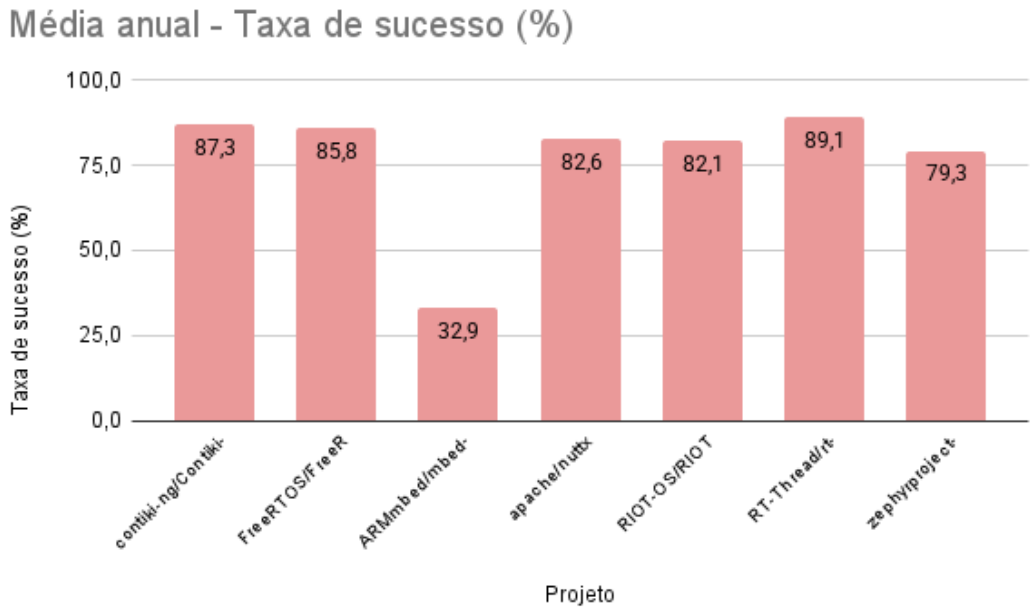

Figura 11 – Média anual - Taxa de sucesso (%)

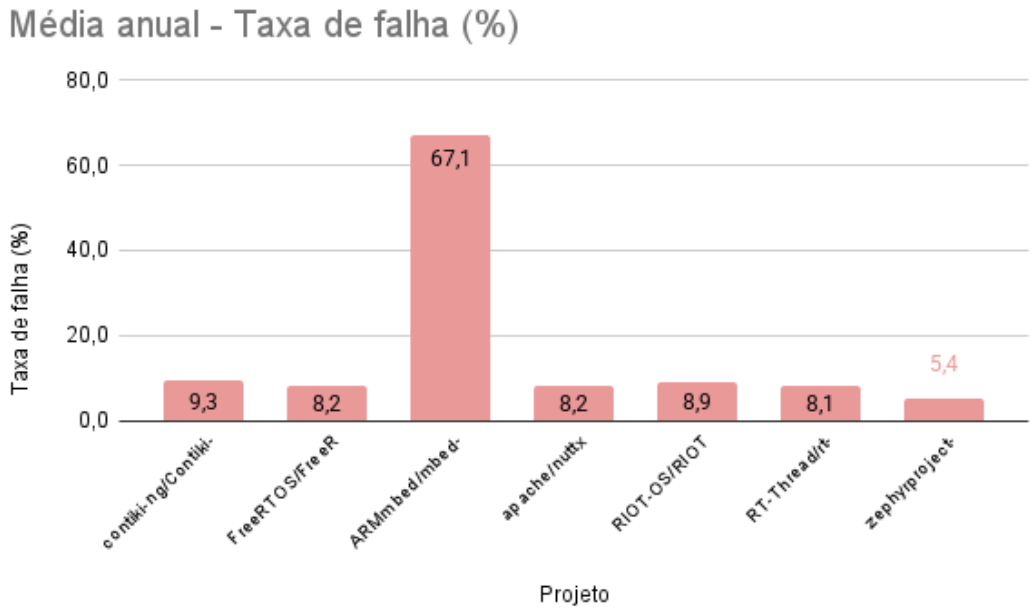

Figura 12 – Média anual - Taxa de falha (%)

A análise das métricas de eficácia operacional dos pipelines evidencia diferentes níveis de maturidade e confiabilidade entre os projetos avaliados. Projetos com maior suporte institucional, como Zephyr e Nuttx, apresentam elevado volume de execuções e workflows complexos, ainda que enfrentem desafios relacionados à estabilidade e ao tempo de processamento. Iniciativas conduzidas por comunidades abertas, como RIOT e RT-Thread, demonstram equilíbrio entre desempenho e consistência, mantendo taxas de sucesso elevadas e tempos de execução moderados. Em contraste, projetos com menor

suporte ou em processo de descontinuidade, como ARMmbed e Contiki-NG, revelam limitações operacionais, seja pela baixa atividade, pela elevada taxa de falhas ou pela restrição de uso em contextos mais específicos. Dessa forma, a resposta à RQ2 indica que a eficácia operacional dos pipelines está condicionada ao grau de automação, à robustez da infraestrutura e ao suporte institucional, fatores determinantes para a confiabilidade e a eficiência da integração contínua.

5 Discussão

A análise dos resultados obtidos permite compreender de forma mais ampla como os projetos de RTOS de código aberto estruturam e mantêm seus pipelines de integração e entrega contínua por meio do GitHub Actions. Os dados apresentados anteriormente evidenciam diferenças significativas entre iniciativas com maior suporte institucional e aquelas conduzidas por comunidades abertas, o que reforça a importância de relacionar os achados com o referencial teórico discutido na fundamentação.

Projetos como zephyr e mbed-os, que contam com apoio de grandes organizações, demonstraram elevado volume de execuções e workflows mais complexos, confirmando a literatura que aponta a automação como elemento central para a escalabilidade e a rastreabilidade em sistemas críticos Decan et al. (2022). Contudo, os desafios de estabilidade e tempo de processamento observados nesses casos indicam que a robustez da infraestrutura não elimina problemas de confiabilidade, aspecto já destacado por Sommerville (2011) ao tratar da previsibilidade em sistemas de tempo real.

Por outro lado, iniciativas comunitárias como RIOT e Nuttx apresentaram equilíbrio entre desempenho e consistência, ainda que com menor volume de execuções. Esse resultado dialoga com Wessel et al. (2023), que ressaltam como práticas automatizadas podem favorecer a colaboração e a qualidade do código em ambientes de software livre. A menor complexidade dos workflows nesses projetos pode ser interpretada como uma estratégia de adaptação às limitações de recursos, sem comprometer a estabilidade operacional.

As limitações identificadas em projetos com baixo suporte institucional, como baixa atividade ou elevada taxa de falhas, reforçam os desafios apontados por Calefato, Lanubile e Quaranta (2022) em relação à adoção de pipelines em domínios específicos. Esses achados sugerem que a eficácia operacional dos pipelines não depende apenas da adoção de ferramentas de automação, mas também da capacidade da comunidade em sustentar práticas contínuas de manutenção e evolução.

Os tempos médios de execução observados nos projetos RTOS FLOSS podem ser comparados com resultados de outros domínios. Bernardo et al. (2024), ao analisar 185 projetos open source, identificaram que projetos de aprendizado de máquina apresentam maior duração de builds e tendência de aumento ao longo do tempo, especialmente em projetos pequenos e médios. No caso dos RTOS FLOSS, os tempos médios de execução mostraram-se mais estáveis, sugerindo que a complexidade dos pipelines está mais associada à diversidade de arquiteturas embarcadas do que ao crescimento contínuo da duração dos builds. Essa comparação evidencia que, embora ambos os domínios enfrentem desafios de automação, os fatores que impactam a duração das execuções diferem substancialmente.

Do ponto de vista das implicações, os resultados indicam que a confiabilidade dos pipelines em RTOS FLOSS está condicionada a três fatores principais: grau de automação, robustez da infraestrutura e suporte institucional. O grau de automação refere-se ao nível em que os processos de integração contínua são executados sem intervenção manual, abrangendo desde a compilação até a validação, o que reduz a ocorrência de erros humanos e aumenta a consistência dos resultados. A robustez da infraestrutura diz respeito à capacidade técnica dos ambientes de execução, incluindo servidores, ferramentas de orquestração e mecanismos de monitoramento, que asseguram estabilidade mesmo diante de altos volumes de processamento ou de falhas pontuais. Já o suporte institucional envolve o engajamento de fundações, empresas e comunidades que sustentam o desenvolvimento, fornecendo recursos, manutenção contínua e governança, fatores que ampliam a confiabilidade e a longevidade dos projetos. Essa constatação contribui para a engenharia de software em sistemas críticos, oferecendo evidências empíricas que podem orientar tanto profissionais quanto pesquisadores na adaptação de práticas de IC/EC às especificidades dos RTOS.

Em síntese, a discussão evidencia que a automação de workflows em RTOS FLOSS não apenas reflete práticas consolidadas de integração e entrega contínua, mas também revela desafios e oportunidades específicos desse domínio. O significado central da análise é mostrar que a eficácia operacional dos pipelines está diretamente ligada à capacidade de equilibrar complexidade técnica, recursos disponíveis e exigências de confiabilidade, constituindo-se em um campo fértil para avanços futuros na engenharia de software aplicada a sistemas de tempo real.

6 Considerações finais

Este trabalho teve como objetivo investigar a estrutura e a eficácia dos workflows automatizados em projetos RTOS de código aberto hospedados no GitHub. A análise concentrou-se nos eventos de acionamento dos *workflows*, nas categorias de Actions utilizadas e nas métricas de eficácia operacional extraídas por meio da API da plataforma. Os resultados demonstraram que os projetos analisados utilizam práticas alinhadas à integração e entrega contínua, ainda que em diferentes níveis de maturidade. A predominância de eventos como *push* e *pull request* confirma a relevância da integração contínua como prática cotidiana, enquanto o uso recorrente de Actions voltadas para testes, compilação e análise de código evidencia a preocupação das comunidades com a qualidade e a confiabilidade do software.

A análise das métricas de eficácia operacional dos *pipelines* evidencia diferentes níveis de maturidade e confiabilidade entre os projetos avaliados. Projetos com maior suporte institucional, como Zephyr e Nuttx, apresentaram elevado volume de execuções e workflows complexos, ainda que enfrentem desafios relacionados à estabilidade e ao tempo de processamento. Iniciativas conduzidas por comunidades abertas, como RIOT e RT-Thread, demonstraram equilíbrio entre desempenho e consistência, mantendo taxas de sucesso elevadas e tempos de execução moderados. Em contraste, projetos com menor suporte ou em processo de descontinuidade, como ARMmbed e Contiki-NG, revelaram limitações operacionais, seja pela baixa atividade, pela elevada taxa de falhas ou pela restrição de uso em contextos predominantemente acadêmicos.

Dessa forma, a pesquisa conclui que a eficácia operacional dos pipelines está diretamente condicionada ao grau de automação, à robustez da infraestrutura e ao suporte institucional. Projetos com maior estrutura e apoio apresentam maior volume de execuções e workflows mais sofisticados, mas também enfrentam desafios de estabilidade. Já iniciativas comunitárias demonstram consistência e eficiência, mesmo com recursos mais limitados. Por outro lado, projetos com menor suporte revelam fragilidades que comprometem sua confiabilidade e continuidade.

Em síntese, este trabalho contribui para o avanço das discussões sobre engenharia de software em sistemas críticos, evidenciando que a automação desempenha papel fundamental para a confiabilidade e a previsibilidade em ambientes de tempo real. Nesse contexto, o GitHub Actions mostra-se como uma excelente ferramenta que pode ser utilizada para implementar práticas de integração e entrega contínua, oferecendo recursos relevantes para projetos de código aberto e servindo como apoio às comunidades que buscam consolidar processos de desenvolvimento mais eficientes e rastreáveis. Além disso, os resultados obtidos

abrem caminho para novas pesquisas que aprofundem a relação entre IC/EC e sistemas críticos, considerando diferentes ferramentas e abordagens de automação.

6.1 Limitações

Entre as principais limitações, destaca-se a representatividade da amostra, restrita a um conjunto específico de projetos selecionados, o que pode limitar a generalização dos resultados para todo o ecossistema de RTOS FLOSS. Além disso, a ausência de dados qualitativos sobre a experiência dos desenvolvedores impede uma compreensão mais profunda das práticas cotidianas de manutenção e colaboração.

Outro aspecto relevante é a existência de Actions personalizadas, que não são oficialmente categorizadas pelo GitHub Actions. Essa limitação dificulta a análise comparativa entre projetos, uma vez que tais componentes podem desempenhar papel significativo na automação, mas permanecem invisíveis em classificações padronizadas.

6.2 Trabalhos futuros

Como continuidade desta pesquisa, sugere-se ampliar o escopo da análise para incluir projetos menos populares, de modo a aumentar a representatividade da amostra e oferecer uma visão mais abrangente do ecossistema de sistemas operacionais de tempo real. Também se recomenda a realização de estudos qualitativos, por meio de entrevistas ou questionários, que permitam compreender a percepção dos desenvolvedores acerca da eficácia dos pipelines e identificar fatores humanos e organizacionais que não são capturados pelas métricas quantitativas. Outra possibilidade consiste em desenvolver técnicas de categorização para lidar com Actions personalizadas que não estão disponíveis no GitHub Marketplace. Esse tipo de recurso, criado especificamente para atender às necessidades particulares de determinados projetos, não possui tags oficiais associadas, o que dificulta sua classificação e análise sistemática. Além disso, torna-se relevante investigar a integração de pipelines de IC/EC com práticas específicas de sistemas embarcados, como compilação cruzada e testes em hardware restrito, considerando as particularidades dos ambientes de tempo real. Por fim, estudos comparativos entre RTOS e outros domínios de software, como projetos de aprendizado de máquina ou sistemas tradicionais, podem contribuir para identificar padrões e diferenças na adoção de automação, ampliando o entendimento sobre a aplicabilidade e os limites das práticas de IC/EC em diferentes contextos.

Referências

BERNARDO, J. a. H. et al. How do machine learning projects use continuous integration practices? an empirical study on github actions. In: *Proceedings of the 21st International Conference on Mining Software Repositories*. New York, NY, USA: Association for Computing Machinery, 2024. (MSR '24), p. 665–676. ISBN 9798400705878. Disponível em: <https://doi.org/10.1145/3643991.3644915>. Citado 3 vezes nas páginas 23, 24 e 48.

BORGES, H. et al. What's in a github star? understanding repository starring practices in a social coding platform. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.]: IEEE, 2016. p. 223–233. Citado na página 25.

CALEFATO, F.; LANUBILE, F.; QUARANTA, L. A preliminary investigation of mlops practices in github. In: *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. New York, NY, USA: Association for Computing Machinery, 2022. (ESEM '22), p. 283–288. ISBN 9781450394277. Disponível em: <https://doi.org/10.1145/3544902.3546636>. Citado 4 vezes nas páginas 23, 24, 33 e 48.

CLOUD, G. *DORA Metrics: Accelerate DevOps Performance*. 2024. <https://dora.dev/guides/dora-metrics/>. Accessed: 05 Mar. 2026. Citado na página 21.

DECAN, A. et al. On the Use of GitHub Actions in Software Development Repositories . In: *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Los Alamitos, CA, USA: IEEE Computer Society, 2022. p. 235–245. Disponível em: <https://doi.ieeecomputersociety.org/10.1109/ICSME55016.2022.00029>. Citado 3 vezes nas páginas 23, 24 e 48.

FOUNDATION, F. S. *The Free Software Definition*. 2021. <https://www.gnu.org/philosophy/free-sw.html>. Acesso em: 10 fev. 2026. Citado na página 17.

FOWLER, M. *Continuous Integration*. 2006. <https://martinfowler.com/articles/continuousIntegration.html>. Acesso em: 10 fev. 2026. Citado na página 19.

FOWLER, M. *Continuous Delivery*. 2010. <https://martinfowler.com/bliki/ContinuousDelivery.html>. Acesso em: 10 fev. 2026. Citado na página 20.

GHALEB, T. A.; ABDULJALIL, O.; HASSAN, S. Ci/cd configuration practices in open source android apps: An empirical study. *ACM Trans. Softw. Eng. Methodol.*, Association for Computing Machinery, New York, NY, USA, v. 35, n. 2, jan. 2026. ISSN 1049-331X. Disponível em: <https://doi.org/10.1145/3736758>. Citado na página 24.

SOMMERVILLE, I. *Engenharia de Software*. 10. ed. Boston: Addison-Wesley, 2011. Citado 3 vezes nas páginas 14, 17 e 48.

VALENTE, M. T. *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. [S.l.]: Editora: Independente, 2020. Citado 2 vezes nas páginas 19 e 20.

WESSEL, M. et al. Github actions: The impact on the pull request process. In: *Proceedings of the 2023 ACM/IEEE International Conference on Software Engineering*. [S.l.]: ACM, 2023. Citado 3 vezes nas páginas 23, 24 e 48.