Análise do Uso de Práticas de DevOps em Projetos Acadêmicos de Graduação na Área de Engenharia de Software

Amanda M. Souza

[1]Instituto de Ciências Exatas e Informática
Pontifícia Universidade Católica de Minas Gerais (PUC Minas)
Belo Horizonte – MG – Brasil

amanda.moura.1381861@sga.pucminas.br

Abstract. *DevOps practices have shown significant growth in the software industry, however, those are not significantly adopted in academic environments. This study aims to evaluate the viability and effectiveness of adopting DevOps practices and tools in undergraduate academic projects, considering the cost-benefit relationship between implementation effort and potential gains. The methodology combines a quantitative survey with 62 Software Engineering students, followed by an analysis of 1526 GitHub repositories classified as academic projects. The analysis focuses on comparing repositories classified as using or not using DevOps practices, examining metrics such as commit distribution patterns and time to establish functional deployment environments. Finally, a qualitative analysis through interviews with 7 students is conducted to evaluate the implementation cost of the most effective tools according to data obtained in the previous stages. The main results indicate a more regular commit history in projects with DevOps, and identifies GitHub Actions, Maven, and Docker as the most relevant tools in the context of the study. It is concluded that a simple and early adoption of CI/CD pipelines triggered by pull requests and standardized build processes can improve the organization and predictability of projects with low-cost. Prioritizing these basic activities from the beginning of the courses can be beneficial for students of Software Engineering.*

Resumo. *As práticas DevOps têm apresentado crescimento significativo na indústria de software, mas sua adoção em projetos acadêmicos ainda é pouco caracterizada. Este estudo tem como objetivo avaliar a viabilidade e eficácia da adoção de práticas e ferramentas de DevOps em projetos acadêmicos de graduação, considerando a relação custo-benefício entre o esforço de implementação e os ganhos potenciais. A metodologia combina um levantamento quantitativo com 62 estudantes do curso de Engenharia de Software, seguido por uma análise de 1526 repositórios do GitHub classificados como projetos acadêmicos. A análise concentra-se na comparação entre repositórios classificados como com ou sem utilização de práticas DevOps, examinando métricas como padrões de distribuição de commits e tempo para estabelecer ambientes de deploy funcionais. Por fim, é feita uma análise qualitativa em forma de entrevista com 7 estudantes para avaliar o custo de implementação das ferramentas mais eficazes de acordo com os dados obtidos nas etapas anteriores. Os principais resultados indicam uma cadência mais regular de commits*

em projetos com DevOps, além de identificar as ferramentas GitHub Actions, Maven e Docker como mais relevantes para o contexto do estudo. Conclui-se que uma adoção simples e antecipada de pipelines CI/CD acionado por pull request e build padronizado pode melhorar organização e previsibilidade de projetos curtos com baixo custo. Priorizar essas atividades básicas desde o início das disciplinas, pode ser benéfico para alunos da área de Engenharia de software.

Bacharelado em Engenharia de *Software* - PUC Minas
Trabalho de Conclusão de Curso (TCC)

Orientadora de conteúdo (TCC I): Joana Souza - joanasouza@pucminas.br
Orientador de conteúdo (TCC I): João Pedro Batisteli - jp.batisteli@hotmail.com
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientadora do TCC II: Michelle Nery - michellenery@sga.pucminas.br

Belo Horizonte, 16 de Dezembro de 2025.

1. Introdução

A área de *DevOps*, ou Desenvolvimento e Operações, tem apresentado crescimento significativo na indústria de software, com previsão de expansão de 21,20% na Taxa Anual Composta de Crescimento entre 2024 e 2032 [Polaris 2024]. Este crescimento é impulsionado pela natureza de sua abordagem, que visa unificar o desenvolvimento de software com as práticas de operações de projeto, responsáveis pela infraestrutura, publicação e manutenção dos sistemas em produção. Estudos demonstram que práticas como CI/CD (das siglas em inglês *Continuous Integration e Continuous Delivery*) ajudam a detectar erros de implementação rapidamente através de *pipelines* automatizadas com passos de teste, além de diminuir problemas de *deploy* [Fluri et al. 2023].

Em cursos de graduação na área de software, alunos desenvolvem projetos em pequenos times, com duração de 4 a 6 meses [Koolmanojwong and Boehm 2013]. O problema central desses projetos reside na **ocorrência sistemática de falhas técnicas e gerenciais nas etapas finais dos trabalhos acadêmicos, comprometendo sua conclusão e qualidade.** Isso é agravado pela limitação de tempo dos estudantes que é dividido entre múltiplas disciplinas, resultando em atrasos e código de baixa qualidade. Estudos como o de Behutiye et al. (2024) examinaram a implementação de métodos ágeis em projetos universitários, identificando benefícios e desafios. Chang and Shirazi (2022) propuseram um processo de tomada de decisão para adaptar o *Scrum* aos cursos acadêmicos, considerando as diferenças entre os contextos profissional e acadêmico.

Apesar das contribuições desses estudos para a compreensão dos benefícios de metodologias ágeis no ambiente acadêmico, existe uma lacuna quanto à aplicação de práticas *DevOps* para auxiliar alunos em projetos de desenvolvimento. Trabalhos anteriores concentram-se em métodos ágeis baseados no *Scrum*, ou na adoção de *DevOps* em contextos empresariais [Mumbarkar and Prasad 2022]. Este trabalho busca preencher essa lacuna por meio de um estudo em três etapas: um levantamento quantitativo com estudantes para identificar percepções dos alunos; uma análise quantitativa de repositórios

de projetos universitários no *GitHub*, investigando como a presença de práticas *DevOps* se relaciona com características de um projeto; e entrevistas com estudantes a fim de avaliar qualitativamente por experiências pessoais o custo de implementação das ferramentas identificadas como mais relevantes.

Portanto, este estudo tem como objetivo geral **avaliar a viabilidade da adoção de práticas e ferramentas de *DevOps* em projetos acadêmicos de graduação, considerando a relação custo-benefício entre o esforço de implementação e os ganhos obtidos sob a perspectiva do estudante de Engenharia de Software.** Para atingir esse objetivo, a pesquisa se desdobra em 3 objetivos específicos: (1) Identificar os principais problemas técnicos e de gestão em projetos acadêmicos por meio de um levantamento quantitativo em forma de questionário; (2) Avaliar a eficácia de ferramentas de *DevOps* na prevenção de falhas técnicas e gerenciais analisando métricas de repositórios públicos do *GitHub* marcados como projetos acadêmicos; (3) Estabelecer recomendações práticas para a implementação de ferramentas *DevOps* adaptadas ao contexto de projetos acadêmicos, considerando o tempo limitado de implementação.

Em suma, os resultados alcançados deste estudo incluem uma compreensão das principais dificuldades técnicas e organizacionais nos projetos caracterizados. Espera-se uma análise da relação entre a utilização de ferramentas de *DevOps* específicas e a mitigação desses problemas. Por fim, o desenvolvimento de recomendações práticas para adoção de *DevOps* em trabalhos universitários, incluindo recomendações de ferramentas que ofereçam o melhor custo-benefício considerando o contexto de recursos limitados presente nos mesmos.

O restante deste artigo está organizado da seguinte maneira: a Seção 2 fornece um arcabouço sobre o campo de *DevOps* e sua relevância para o desenvolvimento de software. A Seção 3 revisa pesquisas anteriores que exploraram aspectos semelhantes ao presente estudo. A Seção 4 descreve a metodologia utilizada para coletar e analisar os dados. A Seção 5 apresenta e interpreta os resultados encontrados pela pesquisa. A Seção 6 apresenta as ameaças a validade do estudo. Por fim a Seção 7 apresenta a conclusão do trabalho e sugere direções para pesquisas futuras.

2. Fundamentação Teórica

Para o entendimento do potencial impacto da adoção de *DevOps* em projetos acadêmicos, é fundamental estabelecer uma base conceitual sólida sobre o tema. Esta seção apresenta os conceitos essenciais que fundamentam a pesquisa e contextualiza a relevância do *DevOps* no cenário atual de desenvolvimento de software.

Segundo Jabbari et al. (2016), *DevOps* pode ser definido como um movimento que visa unificar práticas de implementação e operações de software, com foco principal na comunicação, colaboração e integração entre desenvolvedores e profissionais de operações de TI. A abordagem surgiu como resposta às limitações do modelo em cascata, que se mostrava lento para atender às necessidades das empresas modernas.

Kim et al. (2016) propõem os "Três Caminhos" do *DevOps* como princípios fundamentais que orientam a implementação bem-sucedida das práticas. O Primeiro Caminho enfatiza o fluxo de trabalho da esquerda para a direita, do desenvolvimento para as operações e, finalmente, para o cliente. Ele foca na otimização do fluxo de valor, minimizando o tempo de ciclo e eliminando desperdícios. Práticas como integração contínua,

automação de testes e publicação automatizada são manifestações deste princípio. O Segundo Caminho trata do *feedback* rápido da direita para a esquerda, criando loops de *feedback* que permitem correções rápidas e aprendizado contínuo. Isso inclui monitoramento em tempo real, alertas automatizados e análise de métricas de performance. O Terceiro Caminho promove uma cultura de experimentação contínua e aprendizado, encorajando a tomada de riscos calculados e o aprendizado com falhas.

Para materializar estes princípios em implementações práticas, o *DevOps* aborda um conjunto de práticas e ferramentas que facilitam a colaboração e automação ao longo do ciclo de vida do desenvolvimento de software. Dentre estas, três práticas fundamentais se destacam pela sua capacidade de transformar o processo de desenvolvimento e entrega de software positivamente: Integração Contínua (CI), onde membros de uma equipe integram seu trabalho frequentemente com a parte do produto existente [Fowler 2024]; Entrega Contínua (CD), capacidade de colocar mudanças de todos os tipos nas mãos dos usuários, de forma segura e rápida de maneira sustentável [Humble and Farley 2010]; a Conteinerização fornece uma forma leve de virtualização permitindo empacotar uma aplicação com todas as suas dependências em um ambiente isolado [Merkel 2014].

Para a Engenharia de Software, integrações frequentes e *builds* automatizados em CI materializam a gerência de *builds* e a integração de componentes descritas no SWEBOK (4ª edição, 2024), reduzindo inconsistências e "*drifts*" de dependências. *Pipelines* com *suites* de testes automatizados ampliam cobertura e frequência de verificação, alinhando-se à recomendação de testes contínuos e regressão sistemática. Versionamento, definição de *baselines* e automação de empacotamento são reforçados por CI/CD e pela infraestrutura como código. A entrega Contínua e ambientes reprodutíveis apoia correções rápidas, facilitando *releases* frequentes e diminuindo risco de regressões em manutenção evolutiva e corretiva.

3. Trabalhos Relacionados

Esta seção apresenta uma revisão crítica de estudos anteriores que abordam temas relacionados à implementação de práticas *DevOps* e metodologias ágeis. Os trabalhos selecionados oferecem perspectivas complementares sobre desafios, soluções e métricas relevantes para a presente pesquisa, estabelecendo uma base comparativa para os resultados obtidos.

Fluri et al. (2023) conduziram um estudo sobre a implementação de práticas de CI/CD em projetos de banco de dados, com foco em verificar se a automação melhora o desenvolvimento e implantação dessas aplicações. A pesquisa foi realizada através de dois estudos de caso, ambos projetos de grande porte utilizando *Oracle Database*. Foram realizadas medições quantitativas dos processos de desenvolvimento e entrevistas com desenvolvedores antes e depois da implementação do CI/CD. Os resultados demonstraram uma redução significativa no número de implantações falhas (até 90% em ambiente de teste) e um aumento na frequência de implantações bem-sucedidas. Além disso, os desenvolvedores reportaram uma diminuição na carga cognitiva de trabalho e maior confiança na entrega de novas implementações. A metodologia de avaliação antes/depois e as métricas utilizadas para avaliar o sucesso da implementação podem ser relevantes para a avaliação da eficácia de práticas *DevOps* em projetos acadêmicos.

Alves e Rocha (2021) descrevem um curso orientado a projetos para ensinar conceitos técnicos e não-técnicos de *DevOps*. Os autores mineraram 148 repositórios de 72

projetos diferentes desenvolvidos por estudantes ao longo de quatro anos. Observou-se aumento da produtividade após a adoção de DevOps em relação ao enfoque ágil anterior do curso. Em um questionário sobre percepção pessoal e aprendizado de *DevOps*, os estudantes atribuíram maior relevância a práticas técnicas como integração contínua, implantação contínua, versionamento e entrega contínua, em comparação a aspectos não-técnicos. Particularmente relevante para esta pesquisa é a conclusão dos autores de que, embora a automação de uma *pipeline* de *deployment* seja importante, ela não garante por si só uma entrega contínua efetiva, sendo necessária uma combinação adequada de práticas e ferramentas considerando as limitações de recursos disponíveis.

Smith et al. (2024) investigaram o problema de distribuição desigual de trabalho em projetos acadêmicos de conclusão de curso, propondo o uso de *templates GitHub* e métricas de produtividade individual como solução. O estudo analisou 150 alunos divididos em 29 grupos de 4-5 membros, comparando dados de *commits* entre os anos acadêmicos de 2022/23 e 2023/24. Os autores desenvolveram um índice de equidade para medir a distribuição do trabalho entre membros da equipe, onde 0 indica trabalho igualmente distribuído e 1 indica concentração em um único membro. Os resultados demonstraram que mais de um terço dos *commits* eram realizados nos três dias anteriores aos prazos de entrega, e o índice médio de equidade ficou em torno de 0,6, indicando distribuição significativamente desigual. Após a implementação das intervenções propostas, observou-se uma redução de 24% para 18% nos *commits* realizados nas datas de entrega. Este trabalho é particularmente relevante para esse estudo por evidenciar quantitativamente os problemas de gestão em projetos acadêmicos e demonstrar como métricas do *GitHub* podem ser utilizadas para análise de padrões de desenvolvimento.

4. Materiais e Métodos

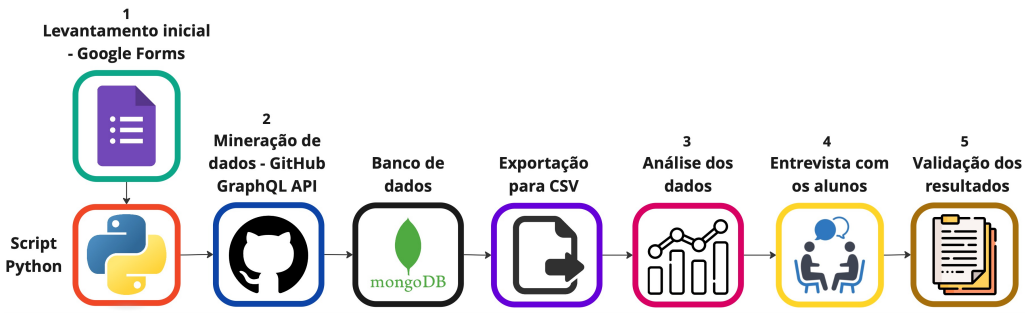

Figura 1. Diagrama de fluxo da metodologia

Esta seção apresenta a metodologia adotada para investigar a eficácia de práticas *DevOps* na mitigação de problemas em projetos acadêmicos. A pesquisa caracteriza-se como quali-quantitativa, de natureza aplicada e objetivo exploratório, visando gerar conhecimentos práticos sobre a implementação de *DevOps* em projetos de graduação. O estudo combina um levantamento por questionário online com uma análise quantitativa de repositórios do *GitHub* e uma entrevista qualitativa, permitindo tanto a identificação dos problemas mais frequentes quanto a avaliação objetiva do impacto das práticas *DevOps* na qualidade e eficiência dos projetos bem como o seu custo de implementação. O desenho adotado foi baseado em métodos mistos [Creswell and Clark 2013], essa

combinação é recomendada em Engenharia de Software para fortalecer inferências através de triangulação de dados [Wohlin and Runeson 2021].

4.1. Levantamento Inicial

A primeira etapa da metodologia, como demonstrado na Figura 1, consiste da aplicação de um questionário online via *Google Forms* para identificar as principais dificuldades técnicas e gerenciais em projetos acadêmicos de desenvolvimento de software. Esta abordagem foi selecionada por sua eficiência na coleta de dados de uma amostra significativa de participantes, além de capturar experiências qualitativas dos envolvidos.

O público-alvo do questionário foi selecionado para garantir respostas baseadas em uma experiência prática relevante, sendo composto por estudantes do curso de Engenharia de Software do 5º ao 8º período, contemplando diferentes níveis de familiaridade com práticas de *DevOps*. A estrutura do questionário (Apêndice A) organiza-se em três seções: (1) perfil do respondente, (2) problemas enfrentados e (3) práticas *DevOps*. Os dados coletados foram analisados por meio de estatística descritiva para identificar tendências e padrões, com ênfase na frequência e no impacto dos problemas relatados. Os resultados dessa análise fundamentaram a definição dos critérios de investigação na etapa de mineração de dados.

4.2. Mineração de Dados

A segunda etapa da metodologia consiste na mineração de repositórios do *GitHub* identificados como projetos acadêmicos, sem exclusão por características de tecnologia. Esta abordagem foi escolhida por permitir uma análise objetiva e quantitativa de dados reais de desenvolvimento, complementando as percepções subjetivas obtidas através do questionário. Para a coleta de dados, foi desenvolvido um sistema em Python 3.9+, utilizando a API *GraphQL* do *GitHub* (v4). O sistema emprega bibliotecas especializadas: *python-dotenv* para gerenciamento de variáveis de ambiente, *requests* para comunicação HTTP com a API, *pymongo* para persistência de dados em MongoDB, *pandas* para análise de dados, *numpy* para operações matemáticas, *scipy* para testes estatísticos e *matplotlib* para visualização dos resultados. O Código em Python (Apêndice B) exemplifica a implementação do minerador de repositórios.

Os repositórios foram selecionados com base nos critérios de: presença de marcadores como "*school-project*", "*university-project*", "*academic-project*" ou "*course-project*"; criação nos últimos cinco anos; e mínimo de três colaboradores. Assim, garantindo que a amostra represente adequadamente projetos acadêmicos contemporâneos. Para cada repositório selecionado, são coletados dados sobre: presença de arquivos de configuração de CI/CD, Dockerfiles e *scripts* de automação; padrões temporais de *commits* e sua distribuição ao longo do ciclo de vida do projeto; existência e histórico de execução de *pipelines*; e tempo de resolução de *issues* e *pull requests*.

Os dados coletados foram armazenados em um banco de dados MongoDB, escolhido por sua flexibilidade para lidar com dados semi-estruturados e sua capacidade de escalar conforme necessário. Esta abordagem permite a persistência dos dados para análises posteriores e facilita a integração com ferramentas de visualização. Posteriormente, as informações armazenadas no banco de dados foram exportadas para um arquivo de formato CSV.

4.3. Análise dos Dados

A terceira etapa da metodologia consiste na análise comparativa dos dados coletados, com foco na identificação de correlações entre a adoção de práticas *DevOps* e os indicadores de qualidade e eficiência dos projetos. Esta análise é estruturada em torno de três categorias principais de métricas:

Métricas de processo, que avaliam a eficiência do processo de desenvolvimento e identificam problemas de gestão e coordenação: (1) frequência de *commits* ao longo do tempo, permitindo identificar padrões como "*commit* em massa" próximo aos prazos de entrega; (2) quantidade média de *commits* realizados entre *releases*, indicando a constância do desenvolvimento para entregas formais; (3) taxa de sucesso de *builds* em *pipelines* CI/CD, refletindo a estabilidade do código; (4) tempo médio de resolução de *issues*, indicando a eficiência da equipe na resolução de problemas.

Métricas de qualidade, que avaliam aspectos relacionados à qualidade técnica do projeto: (1) tempo entre o início do projeto e primeiro *deploy*, refletindo a capacidade da equipe em estabelecer um ambiente funcional; (2) quantidade de *issues* criadas no último mês do projeto, indicando problemas de última hora.

Métricas de adoção, para caracterizar o nível da implementação de práticas *DevOps*: (1) utilização de ferramentas de CI/CD, conteinerização e automação; (2) diversidade das ferramentas *DevOps* empregadas, indicando a amplitude da adoção; (3) tempo dedicado ao *setup* inicial, estimado pela análise dos primeiros commits relacionados à infraestrutura.

A análise comparativa divide os repositórios em dois grupos principais: aqueles que adotam práticas *DevOps*, identificados pela presença de arquivos e pastas relevantes; e aqueles que não adotam. Esta divisão permite avaliar o impacto das práticas *DevOps* nos indicadores de qualidade e eficiência do projeto. Para cada métrica, calculam-se estatísticas descritivas, como mediana e desvio padrão, e realizam-se testes apropriados para verificar a significância das diferenças observadas entre os grupos.

4.4. Entrevistas Qualitativas

A quarta etapa da metodologia consiste na realização de entrevistas qualitativas semiestruturadas com o objetivo de complementar os dados quantitativos obtidos e aprofundar em aspectos práticos da adoção de *DevOps* [Kvale and Brinkmann 2009]. Foram selecionados sete estudantes de Engenharia de Software, priorizando aqueles que participaram do levantamento inicial e demonstraram familiaridade com pelo menos uma ferramenta *DevOps*. As entrevistas foram transcritas automaticamente pelo *Microsoft Teams*, revisadas e analisadas posteriormente identificando padrões recorrentes.

4.5. Validação dos Resultados

A quinta e última etapa da metodologia consiste na validação dos resultados obtidos e na elaboração de recomendações práticas para adoção de *DevOps* em projetos acadêmicos. A validação foi realizada por meio de abordagens complementares: uma análise estatística apropriada para verificar as correlações identificadas; e uma triangulação dos dados, comparando os resultados da mineração de dados com as percepções obtidas por meio do questionário e das entrevistas.

5. Resultados

Esta seção apresenta os dados obtidos a partir da aplicação da metodologia proposta. Nela são identificados padrões e tendências nas respostas do formulário, resultados da mineração de dados e entrevistas realizadas.

5.1. Questionário

O questionário online foi enviado para turmas do quinto ao oitavo período do curso de Engenharia de Software, recebendo ao todo 62 respostas válidas. E analisando as respostas para a pergunta "Quais contratempos você já encontrou em projetos acadêmicos?", os itens mais selecionados pelos alunos (Tabela 1) indicam que ferramentas voltadas para padronização de ambientes, integração contínua e automação de testes têm potencial direto para mitigar os principais desafios enfrentados.

Tabela 1. Principais dificuldades relatadas pelos estudantes

Muita implementação acumulada nas entregas finais	77,4%
Erros de configuração de ambiente	66,1%
Problemas de integração entre componentes ou funcionalidades	58,1%
Falhas em casos de uso sem teste	19,4%
Problemas de performance	9,7%

Quanto aos resultados da questão "Selecione as ferramentas que já tenha utilizado", o ***GitHub Actions*** foi a ferramenta que mais apareceu, presente em 80,6% das respostas, indicando ampla familiaridade dos estudantes com automação de fluxos de integração e entrega contínua. O ***Docker*** ficou em segundo lugar, utilizado por 77,4% dos participantes, confirmando a experiência com containerização. Ferramentas de build como ***Maven*** (64,5%) e ***Gradle*** (33,9%) também apresentaram adoção significativa. Plataformas de nuvem como ***Azure*** (46,8%) e orquestradores como ***Kubernetes*** (30,6%) mostraram uso moderado. Ferramentas de automação de testes como ***Selenium*** (30,6%) e ***Cucumber*** (14,5%) foram as menos utilizadas. Apenas 4,8% dos respondentes declararam não ter utilizado nenhuma das ferramentas listadas.

Com base nesses dados, as ferramentas mais relevantes para serem utilizadas na próxima fase de mineração de dados do *GitHub* são o ***GitHub Actions***, pela alta adoção e relação direta com integração contínua, podendo reduzir problemas de integração; ***Docker***, pela ampla utilização e capacidade de resolver problemas de configuração de ambiente; ***Maven/Gradle***, que padronizam o processo de build e reduzem falhas de integração; **Selenium**, que auxilia na automação de testes, mitigando falhas não detectadas.

Experiência com projetos e práticas *DevOps*: 54,8% dos respondentes possuem mais de três anos de experiência com projetos acadêmicos de desenvolvimento, caracterizando uma amostra com vivência significativa no contexto universitário. Quanto ao nível de familiaridade com práticas *DevOps*, 45,2% se classificaram com nível médio e 27,5% com níveis altos, indicando que há conhecimento consolidado que pode facilitar a adoção de ferramentas como facilitadoras, mas também mostra que há espaço para ampliação.

Problemas na publicação e tempo gasto com correções: 40,3% dos estudantes afirmam encontrar problemas frequentes ao publicar sistemas. O tempo gasto com correções de última hora é elevado: 40,3% gastam entre 3,1 e 6 horas, 16,4% entre 6,1 e 9

horas e 22,2% entre 1 e 3 horas. Apenas 1,6% resolvem problemas em menos de 1 hora e 9,7% relataram que gastam mais de 9 horas para essa atividade. Esses dados evidenciam a necessidade de automação e confiabilidade no processo de *deploy*.

Dificuldades e benefícios das práticas *DevOps*: As principais dificuldades relatadas são o tempo necessário para configuração (53,2%) e a curva de aprendizado elevada (48,4%). Quanto aos benefícios observados, destacam-se a maior facilidade de deploy selecionada por 79% dos respondentes, seguida pela maior produtividade da equipe com 54,8%, maior eficácia de testes com 53,2% e redução de erros no produto com 48,4%.

Esses resultados foram utilizados para identificar as ferramentas mais relevantes na perspectiva de estudantes de Engenharia de Software, que foram utilizadas para guiar a coleta de dados de projetos do GitHub. Além disso o levantamento demonstrou que, apesar das dificuldades iniciais, os benefícios da adoção de práticas *DevOps* podem ser observados pelos participantes com o seu uso em projetos acadêmicos.

5.2. Coleta de repositórios

Com base nos dados consolidados, pode-se evidenciar que práticas *DevOps* relacionadas à integração contínua e containerização, podem ter impacto positivo na redução dos problemas mais comuns em projetos acadêmicos. A mineração de dados do *GitHub* é direcionada para comparar repositórios que possuem zero, com os que têm no mínimo uma das características: pasta do ***GitHub Actions***; histórico de *pipeline* CI/CD antes do meio de vida do projeto; presença de pasta "*tests*"; *Dockerfile* dentre os arquivos; configuração em *docker-compose.yml*; arquivo do *Maven*; pasta "*Selenium*"; presença de arquivos *build.gradle*. A comparação é feita com relação a métricas como padrões temporais de *commits*, histórico de execução de *pipelines*, e tempo de resolução de *issues* e *pull requests*. A avaliação estatística do resultado de cada métrica está descrita no Apêndice C.

Métricas de Processo: Para avaliar o impacto da utilização de práticas de *DevOps* na eficiência do processo de desenvolvimento, foram analisadas as diferenças entre projetos que possuem algum indicativo do uso de *DevOps* e os que não possuem. Para tal, foram verificadas a relação atividade de *commit vs* tempo; atividade de *commit vs releases* formais; taxa de sucesso dos *builds* em *pipelines* CI/CD; e atividade de criação e fechamento de *issues*.

O histórico de *commits* de cada repositório coletado foi separado em dez grupos que representam partições da duração total do projeto, desde o início do projeto (Grupo 1), até a conclusão (Grupo 10). A partir desse agrupamento, foi gerado um *boxplot* (Figura 3), que demonstra que os projetos com *DevOps* tiveram uma mediana de 15,90 por grupo. Em contraste, os projetos sem *DevOps* têm uma mediana de 7,81. Esse perfil descritivo de *commits* mais frequentes e regulares evidencia que os projetos com *DevOps* podem possuir um nível mais alto de atividade de contribuições, possivelmente influenciadas por integração contínua e automação de fluxos de trabalho.

Dos repositórios analisados, 85% dos projetos com *DevOps* e 92% dos projetos sem *DevOps* não possuem qualquer *release* registrada, o que pode indicar que a amostra utilizada não caracteriza de forma integral a população dos projetos acadêmicos. Ainda assim, como pode ser observado na Figura 4, entre os repositórios que possuem *releases*, o perfil descritivo mostra que os projetos com *DevOps* têm uma média de 43,94 e mediana de 15 *commits* entre *releases*. Já os projetos sem *DevOps* apresentam uma média de 28,46

Figura 2. Visualização das métricas de processo

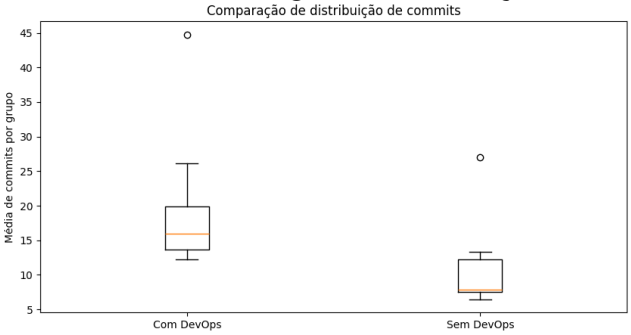

Figura 3. Boxplot de distribuição de commits

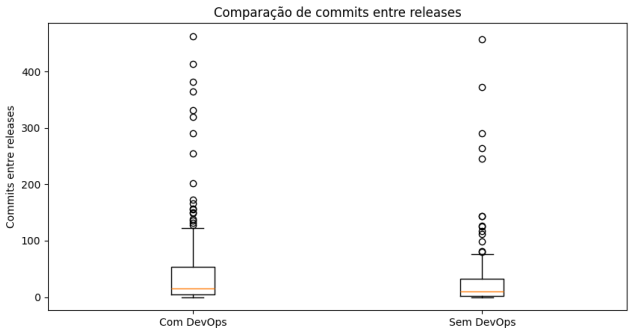

Figura 4. Boxplot de número de commits entre releases

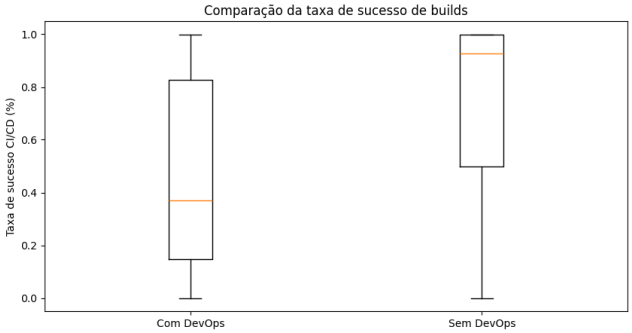

Figura 5. Taxa de sucesso de *pipelines* CI/CD

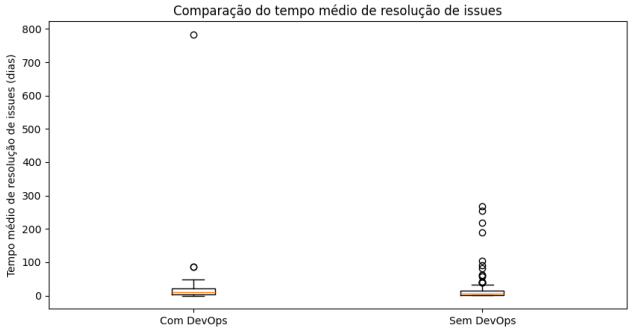

Figura 6. Tempo médio de resolução de *issues* (em dias)

e mediana de 10 *commits* entre *releases*. Os desvios padrão obtidos são de 73,47 e 55,67 respectivamente, sugerindo maior dispersão na categoria com *DevOps*. Esses resultados sugerem que, a adoção de práticas *DevOps* está associada a uma maior quantidade de *commits* entre entregas formais.

Entre os repositórios analisados, 81% dos projetos com *DevOps* e 96% dos projetos sem *DevOps* não possuem identificação de *pipelines*. Nos repositórios com dados disponíveis, como pode ser observado no *boxplot* da Figura 5, projetos com *DevOps* apresentam uma mediana de 0,37 (37% de sucesso). Já os projetos sem *DevOps* apresentam uma mediana de 0,93 (93% de sucesso). As médias reforçam essa diferença: 0,45 para projetos com *DevOps* contra 0,72 para projetos sem *DevOps*, ambos com desvio padrão de 0,36. Esses resultados podem indicar que, nos projetos acadêmicos analisados, a presença de práticas *DevOps* pode estar associada a uma maior taxa de exposição de *bugs* devido à falha das *pipelines* CI/CD, potencialmente contribuindo para a resolução de problemas e melhoria contínua do código. Além disso, pode ser indicativo de que nos projetos classificados como sem *DevOps*, a pipeline não foi integrada ao processo de desenvolvimento.

Dos repositórios analisados, 84,4% dos projetos com práticas *DevOps* e 87,8% dos projetos sem práticas *DevOps* não possuem *issues* fechadas, sendo descartados na avaliação do tempo de resolução de *issues*. Nos projetos com *DevOps*, a mediana foi de 10,06 dias, a média observada foi de 25,31 dias, com desvio padrão elevado (92,82 dias), indicando grande dispersão e presença de casos extremos em que a resolução de problemas levou vários meses. Já nos projetos sem *DevOps*, a mediana foi de 6,70 dias, a média

foi de 18,65 dias, com desvio padrão de 42,19 dias, também sugerindo dispersão significativa, embora menor que na categoria com *DevOps*. Os resultados podem indicar que, em alguns projetos a adoção de práticas *DevOps* pode estar associada a um tempo médio de resolução ligeiramente maior, possivelmente relacionado à formalização do processo de registro e resolução de problemas.

Métricas de Qualidade: Para avaliar o impacto da utilização de práticas de *DevOps* em aspectos de qualidade, foram analisadas as diferenças entre projetos que possuem algum indicativo do uso de *DevOps* e os que não possuem. Para tal, foram verificadas as métricas de tempo até o primeiro *deploy* e criação de *issues* ao final do projeto.

Figura 7. Visualização das métricas de qualidade

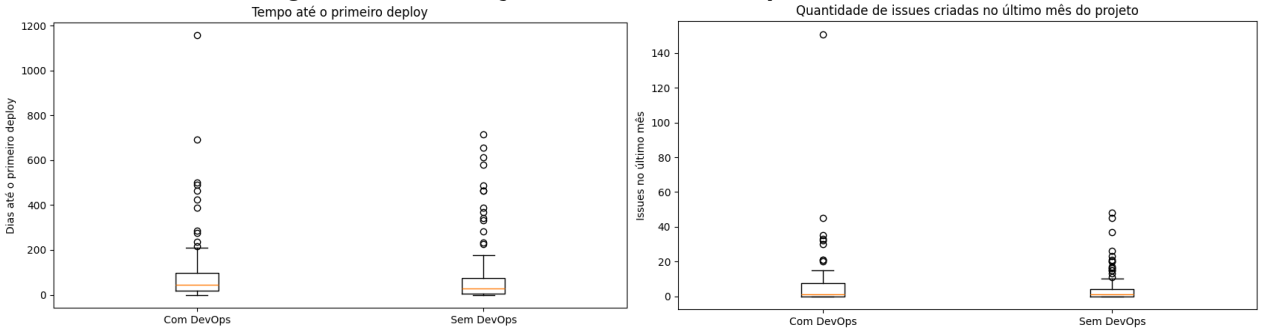

Figura 8. Tempo até o primeiro *deploy* (em dias)

Figura 9. Quantidade de *issues* criadas no último mês do projeto

Nos projetos com *DevOps*, a mediana de tempo até o primeiro deploy foi de 43 dias, com média de 94,51 dias e desvio padrão de 156,78 dias. Já nos projetos sem *DevOps*, a mediana foi de 27 dias com média de 72,56 dias e desvio padrão de 131,19 dias. Esses resultados podem indicar que, de forma geral, projetos sem *DevOps* tendem a realizar o primeiro *deploy* em menos tempo, possivelmente por optarem por configurações mais simples ou menos dependentes de automação inicial. Por outro lado, projetos com *DevOps* podem demandar mais tempo no início para configurar *pipelines*, ambientes de conteinerização e outras práticas, o que pode retardar o primeiro *deploy*, mas potencialmente trazer benefícios nas etapas subsequentes do desenvolvimento.

Dos repositórios analisados, 84% dos projetos com práticas *DevOps* e 88% dos projetos sem práticas *DevOps* não possuem qualquer *issue* registrada, indicando que a maioria dos projetos acadêmicos não mantém histórico formal de rastreamento de tarefas por esse método. Como pode ser observado na Figura 9, entre os repositórios que possuem *issues*, o perfil descritivo mostra que os projetos com *DevOps* apresentam uma média de 7,87 *issues* no último mês e mediana de 1,00. Já os projetos sem *DevOps* apresentam uma média de 3,89 *issues* e mediana de 1,00. Os desvios padrão obtidos foram de 19,87 e 8,11 respectivamente, sugerindo maior dispersão na categoria com *DevOps*. Esses resultados podem indicar que, entre projetos acadêmicos que registram *issues*, a adoção de práticas *DevOps* está associada a uma quantidade ligeiramente maior de problemas reportados no último mês do projeto. A maior dispersão na categoria com *DevOps* pode estar relacionado à maior visibilidade e formalização do processo de registro de problemas.

Métricas de Adoção: Para avaliar a adoção prática de ferramentas *DevOps* nos

projetos classificados como “Com *DevOps*”, foram realizadas: a contagem de repositórios que utilizam diferentes tecnologias; uma avaliação da diversidade de ferramentas por repositório foi analisada; e uma observação do intervalo entre a criação do repositório e o primeiro *commit* relacionado à configuração de infraestrutura via *Docker*.

Figura 10. Visualização das métricas de adoção

Figura 11. Utilização de Ferramentas *DevOps* nos projetos classificados como “Com *DevOps*”

Ferramenta	Uso Absoluto	Uso Relativo (%)
GitHub Actions (CI/CD)	204	44,84
Maven	99	21,76
Tests Folder	74	16,26
Dockerfile	66	14,51
Docker Compose	58	12,75
Gradle	43	9,45
Gradle com Kotlin	17	3,74
Selenium	0	0,00

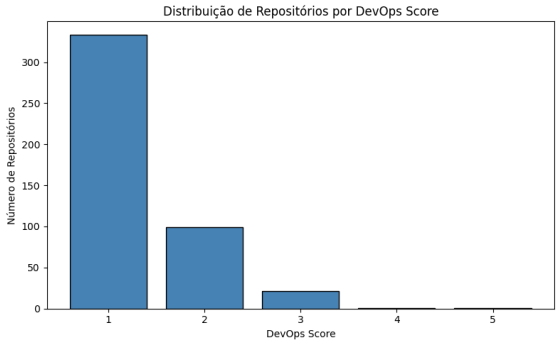

Figura 12. Diversidade de ferramentas

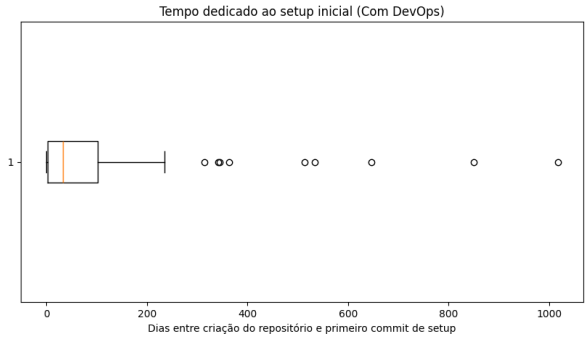

Figura 13. Tempo de configuração do ambiente inicial

Os resultados da Figura 11 mostram que a ferramenta mais utilizada é o *GitHub Actions*, presente em 44,84% dos repositórios com *DevOps*, evidenciando uma ampla adoção de *pipelines* de integração e entrega contínua. Em seguida, destaca-se o *Maven* (21,76%), indicando uso relevante de automação de *build*. A presença de pastas de testes (*testsFolder*) aparece em 16,26% dos repositórios, sugerindo que a automação de testes ainda não é predominante. Ferramentas de conteinerização como *Dockerfile* (14,51%) e *Docker Compose* (12,75%) apresentam adoção moderada, enquanto *Gradle* e *Gradle* com *Kotlin* têm participação menor (9,45% e 3,74%, respectivamente). O uso de Selenium para automação de testes de interface é inexistente na amostra.

O histograma da Figura 12 mostra que a maioria dos projetos concentra-se em baixa diversidade de ferramentas, com poucos casos de uso combinando três ou mais tecnologias. Dos 455 repositórios analisados, 73,19% possuem *devops score* igual a 1, indicando adoção mínima de práticas *DevOps*. Em seguida, 21,76% dos repositórios apresentam *devops score* igual a 2, enquanto apenas 4,62% atingem o valor 3. Os níveis mais altos são raros: apenas 1 repositório possui *devops score* igual a 4 e outro atinge o maior valor de 5.

Dos repositórios classificados como “Com *DevOps*”, 117 apresentaram evidências de setup inicial com *Docker*. O *boxplot* da Figura 13 mostra uma média de 89,79 dias e

mediana de 33 dias até o primeiro *commit* relacionado a configuração do *Docker*, indicando que metade dos projetos realiza o *setup* em até pouco mais de um mês após a criação do repositório. Esses resultados sugerem que, mesmo entre projetos que adotam práticas *DevOps*, a configuração de infraestrutura com *Docker* nem sempre é imediata. Em alguns casos, o *setup* é feito logo no início, possivelmente como parte de uma abordagem estruturada de desenvolvimento; em outros, é adiado, o que pode indicar que o uso de containerização foi incorporado apenas em fases posteriores do projeto, talvez para atender requisitos específicos ou facilitar a implantação.

5.3. Entrevistas

Foram realizadas sete entrevistas semiestruturadas com estudantes de Engenharia de Software ("Entrevistado n" - En). Os entrevistados foram selecionados por terem experiências prévias em projetos acadêmicos e exposição variada às ferramentas *GitHub Actions*, *Maven* e *Docker*.

No caso do *GitHub Actions*, a percepção predominante é de que a curva de aprendizado é baixa para configurar fluxos simples de *build*, teste e *deploy*. Os entrevistados relatam que exemplos e *templates* públicos facilitam o início, tornando a configuração "rápida" e "intuitiva" quando o objetivo é um *pipeline* padrão. O E1, que configurou *pipelines* mais complexas, destaca que a dificuldade cresce à medida que se introduzem ações customizadas, concorrência entre *jobs*, integração com certificados e serviços em nuvem. Benefícios práticos foram recorrentes: o E3 usou múltiplas *pipelines* em paralelo para reduzir drasticamente o tempo de processamento de tarefas pesadas; outros (E4, E5) ressaltam a automação de *deploy* "com um clique" e a redução de trabalho manual; também comentam sobre um ganho de produtividade e visibilidade contínua do estado do sistema. O uso posterior à configuração foi percebido como simples quando a *pipeline* está bem estruturada. Estratégias de aprendizagem incluíram documentação, exemplos públicos, apoio de colegas e uso de IA. Em síntese, a ferramenta oferece alto valor com custo inicial baixo em cenários de CI/CD básicos, com crescimento de complexidade proporcional à personalização.

Em relação ao *Maven*, há consenso quanto à facilidade de início em projetos Java, sobretudo via *Spring Initializr*, que gera um esqueleto funcional rapidamente. Seu valor é associado à padronização do *build* e das dependências, à ampla disponibilidade de conteúdo e à integração fluida com *pipelines* de CI. O E7 relata o uso da ferramenta para controle de dependências transitivas e mitigação de vulnerabilidades, e destaca como principal benefício a centralização e organização das dependências. Entretanto, emergem barreiras conceituais nas primeiras experiências: compreensão do *lifecycle*, da estrutura do POM e da gestão de artefatos em XML. O E5 ressalta a dificuldade prática de diagnosticar erros genéricos, que tornam o *troubleshooting* mais demorado. Apesar disso, a recomendação é majoritariamente favorável, tanto por utilidade técnica quanto por alinhamento ao mercado. Há, porém, preferência explícita pelo *Gradle* por parte de alguns (E1 e E5), que o consideram menos verboso e mais extensível, sobretudo quando a equipe busca simplicidade de configuração.

O *Docker* é a ferramenta com percepção de maior custo de entrada. Os entrevistados convergem que a curva de aprendizado é elevada quando se vai além de casos básicos, exigindo entendimento de conceitos de containerização, escolha de ima-

gens, redes e, eventualmente, orquestração. Ainda assim, reconhecem ganhos significativos: padronização de ambientes, portabilidade entre máquinas e nuvens, aceleração na configuração de ambiente pessoal e facilidade para subir múltiplos serviços. Para atividades simples e imagens prontas, o esforço é reduzido, mas a personalização e o ajuste fino aumentam a complexidade. O E6 estimou que cerca de um terço do tempo de um de seus projetos foi utilizado na configuração da ferramenta; o E2 relata barreira adicional com o uso por linha de comando e menciona o Podman como facilitador. Existem relatos contrastantes quanto ao retorno no contexto acadêmico: em um projeto específico, o E4 não percebeu benefícios práticos proporcionais ao esforço; por outro lado, E1 e E5 enfatizam ganhos claros de previsibilidade, reprodutibilidade e velocidade de desenvolvimento, bem como a utilidade formativa para o mercado. No balanço, a decisão de adotar *Docker* em trabalhos curtos e simples parece sensível ao escopo e à experiência da equipe.

6. Ameaças a Validade

Esta seção explicita as principais limitações do estudo e descreve as estratégias adotadas para mitigá-las, com o objetivo de delimitar claramente o escopo de inferência dos resultados, em especial dos achados quantitativos, e de evidenciar como a triangulação com as evidências qualitativas confere maior robustez ao conjunto. Os resultados da mineração mostraram uma baixa representatividade de dados formais, com a ausência de *releases*, *issues* e indicação de *pipelines* em boa parte da amostra, além das características do desenho e das fontes de dados utilizadas: heterogeneidade de *stacks* e linguagens sem controle específico; critério de inclusão de repositórios pelos tópicos "*school-project*", "*university-project*", "*academic-project*" e "*course-project*", janela temporal 2020–2025 e mínimo de três colaboradores. Para mitigar essas ameaças, os resultados numéricos foram interpretados como associações e não como relações causais, e foram acompanhados de testes estatísticos e medidas de efeito para evitar conclusões além do suporte empírico disponível.

Ameaças à validade interna: a heurística de detecção de CI/CD considerou apenas a primeira execução de *pipeline* bem-sucedida antes da metade do projeto, não refletindo o histórico completo de execuções nem mudanças de configuração ao longo do tempo; e a classificação de adoção de *DevOps* por presença de arquivos, como *Dockerfile* e pasta *tests*, pode gerar falsos positivos quando tais artefatos estão inativos ou desatualizados. Para mitigar esses riscos, adotou-se um conjunto de artefatos para serem utilizados na identificação do *DevOps*. Adicionalmente, a triangulação com entrevistas buscou validar a plausibilidade dos padrões observados, reduzindo o risco de interpretações enviesadas.

Ameaças à validade externa: o levantamento concentrou-se em turmas do curso de Engenharia de Software e períodos avançados, podendo não representar contextos com menor exposição a ferramentas de mercado; e práticas de gestão fora do GitHub, como rastreamento de tarefas em outras plataformas, podem ter reduzido a visibilidade de *issues* e *releases*. Para mitigar a generalização indevida, manteve-se a interpretação como associação, reconhecendo que esses resultados são mais representativos de contextos similares, de projetos de graduação com colaboração em GitHub, do que de todo o universo de projetos acadêmicos.

Uma ameaça adicional à validade externa decorre da ausência de uma análise sistemática de restrições institucionais, tais como limitações de infraestrutura, licencia-

mento e custos de ferramentas, e a adequação a conteúdos e diretrizes curriculares do MEC. Essas condições podem facilitar ou dificultar a adoção de práticas e ferramentas de DevOps, modulando os efeitos práticos observáveis nos cursos e projetos. Como forma de mitigação parcial, as recomendações propostas foram desenhadas com foco em baixo custo e baixa complexidade, e sugerem adoção incremental e seletiva.

Há, ainda, fatores de confusão não controlados (tamanho e experiência da equipe, complexidade e tipo de projeto, disciplina e calendário acadêmico, linguagem e ecossistema), que podem afetar a relação entre adoção de *DevOps* e métricas observadas. Nas entrevistas, a autoavaliação de familiaridade com *DevOps* tende a concentrar-se em níveis médios, o que pode distorcer o custo percebido de implementação, além de possíveis imprecisões de transcrição e viés de memória. Embora a triangulação entre questionário, mineração e entrevistas, bem como o uso de testes estatísticos apropriados, mitigem parte desses riscos, reconhece-se que eles limitam a generalização e a interpretação causal dos resultados.

7. Conclusão e Trabalhos Futuros

Este estudo investigou a viabilidade da adoção de práticas de *DevOps* em projetos acadêmicos de graduação, combinando levantamento com estudantes, mineração de repositórios do GitHub e entrevistas semiestruturadas. Do ponto de vista quantitativo, o achado mais significativo foi a distribuição mais regular de *commits* ao longo do tempo em projetos que adotam práticas de *DevOps*. Essa cadência pode reduzir a concentração de implementação nas fases finais e endereçar diretamente o principal problema relatado pelos respondentes do questionário online. Demais métricas apresentaram diferenças pequenas e alta sobreposição, refletindo dados esparsos e heterogeneidade da amostra.

Do ponto de vista qualitativo, *GitHub Actions*, *Maven* e *Docker* foram bem avaliados pelos estudantes por ganhos práticos, como facilidade de *deploy*, padronização de *build*, reprodutibilidade de ambientes e por alinhamento ao mercado. Apesar de "tempo de configuração" e "curva de aprendizado" surgirem como barreiras, a familiaridade prévia com CI/CD em mais da metade da amostra sugere que iniciar com *pipelines* básicas e *templates* reduz atrito no contexto acadêmico. Visto isso, propõe-se uma recomendação prática para a adoção de *DevOps* focado nas três ferramentas mais relevantes identificadas (*GitHub Actions*, *Maven* e *Docker*) e em facilitadores mencionados nas entrevistas, mantendo baixo custo e complexidade:

Iniciar o projeto com uma *pipeline* simples do *GitHub Actions* disparada por *pull request*, contendo passos de *build* e testes. A ferramenta possui curva de aprendizado baixa para fluxos básicos e ampla disponibilidade de *templates* e exemplos. Padronizar o *build* desde o início com *Maven* ou *Gradle* considerando a familiaridade do time e *stack* do projeto, oferecendo centralização de dependências. Para o *Docker*, adotar quando houver ganho claro de reprodutibilidade ou necessidade de múltiplos serviços. Começar com *Dockerfile* mínimo e, se necessário, *docker-compose* para desenvolvimento; fixar versões de imagem e evitar personalizações profundas, a fim de reduzir o custo inicial de tempo. Como facilitadores, considerar alternativas mais leves mencionadas nas entrevistas, como *Podman*.

Para trabalhos futuros, propõe-se expandir para análises por linguagem, *stack* e tipo de projeto; revisar a heurística de detecção de CI/CD incorporando todo o histórico

de execuções e eventos; conduzir estudos experimentais comparando turmas com e sem adoção das recomendações propostas, controlando fatores de confusão como tamanho e experiência do time e complexidade do projeto; ampliar a instrumentação de qualidade (testes, cobertura e vulnerabilidades) e de entrega nos repositórios analisados. Essas extensões podem refinar a compreensão dos impactos de *DevOps* no contexto acadêmico, fortalecer a base empírica e apoiar recomendações mais prescritivas para adoção eficiente em cenários educacionais.

Pacote de Replicação

O pacote de replicação deste trabalho encontra-se disponível em:

`https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2025-1-tcci-0393100-pes-amanda-moura/tree/main`

Referências

Alves, I. and Rocha, C. (2021). Qualifying software engineers undergraduates in devops - challenges of introducing technical and non-technical concepts in a project-oriented course. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, pages 144–153.

Behutiye, W., Tripathi, N., and Isomursu, M. (2024). Adopting scrum in hybrid settings, in a university course project: Reflections and recommendations. *IEEE Access*, 12:105633–105650.

Chang, H.-F. and Shirazi, M. S. (2022). Adapting scrum for software capstone courses. *Informatics in Education*, 21(4):605–634.

Creswell, J. W. and Clark, V. L. P. (2013). *Pesquisa de métodos mistos*. Penso.

Fluri, J., Fornari, F., and Pustulka, E. (2023). Measuring the benefits of ci/cd practices for database application development. In *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*, pages 46–57.

Fowler, M. (2024). Continuous integration. Disponível em `https://martinfowler.com/articles/continuousIntegration.html`. Acesso em: 13 jun. 2025.

Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Signature Series (Fowler). Pearson Education.

Jabbari, R., bin Ali, N., Petersen, K., and Tanveer, B. (2016). What is devops? a systematic mapping study on definitions and practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, XP '16 Workshops, New York, NY, USA. Association for Computing Machinery.

Kim, G., Humble, J., Debois, P., and Willis, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations*. ITpro collection. IT Revolution.

Koolmanojwong, S. and Boehm, B. (2013). A look at software engineering risks in a team project course. In *2013 26th International Conference on Software Engineering Education and Training (CSEET)*, pages 21–30.

Kvale, S. and Brinkmann, S. (2009). *InterViews: Learning the craft of qualitative research interviewing*. Sage Publications, Inc.

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239).

Mumbarkar, P. and Prasad, S. (2022). Adopting devops: Capabilities, practices, and challenges faced by organizations. *AIP Conference Proceedings*, 2519.

Polaris (2024). Devops market share, size, trends, industry analysis report, by organizational size (small and medium enterprises, large enterprises); by industry; by deployment; by offering; by region; segment forecast, 2024 - 2032. Technical report, Polaris Market Research.

Smith, S., Schankula, C. W., Dutton, L., and Anand, C. K. (2024). A software engineering capstone course facilitated by github templates.

Washizaki, H. (2024). *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide)*. IEEE Computer Society.

Wohlin, C. and Runeson, P. (2021). Guiding the selection of research methodology in industry–academia collaboration in software engineering. *Information and Software Technology*, 140:106678.

Apêndice

A. Estrutura do Questionário

1. Perfil do Respondente:

- Tempo de experiência com projetos acadêmicos de desenvolvimento [Seleção única]
  - Menos de 1 ano
  - Entre 1 e 2 anos
  - Entre 2 e 3 anos
  - Mais de 3 anos
- Familiaridade com práticas DevOps [Escala Likert 1-5]
  - 1 - Nenhuma familiaridade
  - 2 - Pouca familiaridade
  - 3 - Familiaridade moderada
  - 4 - Boa familiaridade
  - 5 - Excelente familiaridade
- Experiência com ferramentas de CI/CD [Seleção única]
  - Nunca ouvi falar
  - Já ouvi falar mas nunca utilizei
  - Já aprendi um pouco
  - Já utilizei em um ou dois projetos
  - Já utilizei em vários projetos

2. Problemas Enfrentados:

- Tipos de contratempos mais comuns encontrados [Múltipla escolha]
  - Muita implementação acumulada nas entregas finais

  - Problemas de integração entre componentes ou funcionalidades
  - Erros de configuração de ambiente
  - Falhas em casos de uso não testados
  - Problemas de performance
  - Outro(s) (campo de texto)
- Frequência de problemas no deploy final [Escala Likert 1-5]
  - 1 - Nunca ocorre
  - 2 - Raramente ocorre
  - 3 - Às vezes ocorre
  - 4 - Frequentemente ocorre
  - 5 - Sempre ocorre
- Tempo médio gasto com correções de última hora [Seleção única]
  - Menos de 1 hora
  - Entre 1 e 3 horas
  - Entre 3 e 6 horas
  - Entre 6 e 9 horas
  - Mais de 9 horas

3. Práticas DevOps:

- Ferramentas DevOps já utilizadas [Múltipla escolha]
  - GitHub Actions
  - Jenkins
  - Docker
  - Maven
  - Gradle
  - Kubernetes
  - Azure
  - Travis CI
  - CircleCI
  - Selenium
  - Cucumber
  - Nenhuma
  - Outro(s) (campo de texto)
- Dificuldades na implementação de práticas DevOps [Múltipla escolha]
  - Curva de aprendizado elevada
  - Falta de documentação clara
  - Tempo necessário para configuração
  - Problemas de infraestrutura
  - Custo das ferramentas
  - Nenhuma
  - Outro(s) (campo de texto)
- Benefícios observados com uso de ferramentas DevOps [Múltipla escolha]
  - Maior facilidade de deploy
  - Maior eficácia de testes
  - Maior produtividade da equipe
  - Redução de erros no produto
  - Nenhum
  - Outro(s) (campo de texto)

Algorithm 1: Minerador de Repositórios

```
class GitHubRepositoryMiner:
  def handle_query_run(self, query, variables={}, retries=5,
      wait_time=10):
    for attempt in range(retries):
      try:
        request = requests.post(GITHUB_API_URL, json={'query':
            query, 'variables': variables}, headers=self.headers,
             timeout=30)

        if "X-RateLimit-Remaining" in request.headers:
          remaining = int(request.headers["X-RateLimit-Remaining
              "])
          limit = int(request.headers["X-RateLimit-Limit"])
          reset_time = int(request.headers["X-RateLimit-Reset"])
          used = int(request.headers.get("X-RateLimit-Used", 0))
          self.graphql_remaining = remaining

          if remaining < 100:  # proactive threshold
            wait_seconds = reset_time - int(time.time()) + 1
            print(f"[GraphQL] Low rate limit ({remaining}/{limit
                }, used {used}). Waiting {wait_seconds}s...")
            time.sleep(wait_seconds)

        if request.status_code == 200:
          return request.json()

        elif request.status_code == 403:
          remaining = int(request.headers.get("X-RateLimit-
              Remaining", 0))
          reset_time = int(request.headers.get("X-RateLimit-
              Reset", time.time()))
          if remaining == 0:
            wait_seconds = reset_time - int(time.time()) + 1
            print(f"[GraphQL] Rate limit exceeded. Waiting {
                wait_seconds}s...")
            time.sleep(wait_seconds)
          else:
            print(f"[GraphQL] 403 error but {remaining} requests
                 remaining.")

        else:
          print(f"Attempt {attempt + 1} failed ({request.
              status_code}): {request.text}")
          if attempt < retries - 1:
            time.sleep(wait_time)

      except requests.exceptions.RequestException as e:
        if attempt < retries - 1:
          time.sleep(wait_time)

    raise Exception(f"Failed to execute query after {retries}
        attempts.")
```

Tabela 2. Comparação estatística para a métrica de distribuição de commits por fase do projeto

Grupo	t (Welch)	p-valor (Welch)	Cohen's d	U (Mann–Whitney)	p-valor (MW)	RBC
1 - 9	2.8551 => 4.1886	3.279×10^{-5} => 0.004488	0.191 => 0.300	227 096.0 => 244 296.0	0.006931 => 4.053×10^{-7}	-0.164 => -0.082
10	3.3721	0.0007962	0.232	223 395.5	0.05348	-0.064
Geral	9.6724	6.009×10^{-22}	–	23 331 455.0	6.014×10^{-28}	–

B. Código em Python

C. Detalhes estatísticos

A comparação estatística (Tabela 2) indica diferença significativa entre as categorias em praticamente todas as fases, com exceção da fase final (Grupo 10), onde a diferença não foi estatisticamente relevante pelo teste de *Mann–Whitney*.

Tabela 3. Comparação estatística para a métrica de commits entre releases consecutivas

t (Welch)	p-valor (Welch)	Cohen's d	U (Mann–Whitney)	p-valor (MW)	RBC
2.4626	0.01421	0.237	26 344.5	0.006796	-0.151

A comparação estatística (Tabela 3) indica diferença significativa entre as categorias, com valores de tamanho de efeito pequenos (*Cohen's d* = 0,237; RBC = -0,151), sugerindo que, embora haja mais *commits* entre *releases* nos projetos com *DevOps*, existe considerável sobreposição entre as distribuições.

Tabela 4. Comparação estatística para a métrica de tempo médio de resolução de *issues*

t (Welch)	p-valor (Welch)	Cohen's d	U (Mann–Whitney)	p-valor (MW)	RBC
0.5738	0.5676	0.103	5 276.0	0.1151	-0.135

A comparação estatística (Tabela 4) não indicou diferença significativa entre as categorias, com tamanho de efeito muito pequeno, sugerindo pouca influência da adoção de práticas *DevOps* sobre o tempo médio de resolução de *issues*.

Tabela 5. Comparação estatística para a métrica de tempo até o primeiro *deploy*

t (Welch)	p-valor (Welch)	Cohen's d	U (Mann–Whitney)	p-valor (MW)	RBC
1.1978	0.2324	0.154	9 956.0	0.01128	-0.183

A comparação estatística (Tabela 5) mostra que não houve diferença significativa entre as médias pelo teste de Welch, mas o teste de *Mann–Whitney* indicou diferença na distribuição dos tempos. O tamanho de efeito foi pequeno (*Cohen's d* = 0,154; RBC = -0,183), sugerindo que, embora projetos sem *DevOps* tendam a realizar o primeiro *deploy* mais rapidamente, há considerável sobreposição entre as distribuições.

Tabela 6. Comparação estatística para a métrica de quantidade de *issues* criadas no último mês do projeto

t (Welch)	p-valor (Welch)	Cohen's d	U (Mann–Whitney)	p-valor (MW)	RBC
1.6195	0.1091	0.296	5 024.0	0.3178	-0.080

A comparação estatística (Tabela 6) não indicou diferença significativa entre as categorias, com tamanho de efeito pequeno (*Cohen's d* = 0,296; RBC = -0,080), sugerindo alta sobreposição entre as distribuições e pouca influência da adoção de práticas *DevOps* sobre a quantidade de *issues* registradas no último mês do projeto (mês da última atualização).