Impacto da Adoção de Continuous Integration (CI) na Qualidade do Código: Uma Análise Baseada em Métricas Estruturais e Estáticas

Bruno P. Duarte[1], **Leticia A. Fraga**[1]

[1]Pontifícia Universidade Católica de Minas Gerais (PUC-MG)
Caixa Postal 15.064 – 91.501-970 – Belo Horizonte – MG – Brazil

Abstract. *Continuous Integration (CI) has become an essential practice in modern software development. Although there are studies that associate CI with improvements in code quality and test suites, there remains a lack of analyses examining its effects on source code quality, particularly in object-oriented projects. Thus, this work investigates the impacts of adopting CI on the structural quality of source code in open-source Java projects. The analysis is based on CK metrics, as well as fan-in and fan-out. The study mines the most popular Java repositories on GitHub and evaluates the historical evolution of these metrics before and after the adoption of CI. The results indicate that CI positively influences the progression of the CBO, WMC, RFC, LCOM and fan-out metrics, while showing no significant effects on the others.*

Resumo. *A Integração Contínua (CI, do inglês Continuous Integration) consolidou-se como uma prática essencial no desenvolvimento moderno de software. Ainda que existam estudos que a relacionam à melhoria da qualidade do código e da suíte de testes, há uma escassez de análises que associem os efeitos da CI sobre a qualidade do código-fonte, sobretudo em projetos orientados a objetos. Assim, este trabalho investiga os impactos da adoção da CI na qualidade estrutural do código-fonte de projetos open-source desenvolvidos em Java. A análise é conduzida a partir de métricas CK, fan-in e fan-out. O estudo minera os repositórios em Java mais populares do GitHub e avalia a evolução histórica das métricas antes e depois da adoção da CI. Os resultados indicam que a CI influencia positivamente a progressão das métricas CBO, WMC, RFC, LCOM e fan-out, enquanto não produz efeitos significativos sobre as demais.*

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

Orientador de conteúdo (TCC I): João Pedro Batisteli - joao.batisteli@sga.pucminas.br
Orientador de conteúdo (TCC I): Joana Souza - joanasouza@pucminas.br
Orientador de conteúdo (TCC I): Leonardo Vilela - leonardocardoso@pucminas.br
Orientador acadêmico (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientador do TCC II: José Laerte Pires Xavier Júnior - laertexavier@pucminas.br

Belo Horizonte, 23 de novembro de 2025.

1. Introdução

A Integração Contínua (CI, do inglês *Continuous Integration*) é uma prática de desenvolvimento de software baseada na automação de verificações de compilação e testes, com o

objetivo de facilitar o desenvolvimento de software [Cassee et al. 2020]. Essa automação reduz riscos de atrasos, erros de integração e facilita a extensão do sistema para acomodar novas funcionalidades [Fowler 2024]. O funcionamento da CI baseia-se na execução de uma *pipeline* automatizada que é acionada a cada alteração de código, sendo composta por uma sequência de passos denominados *steps*, como compilação, testes e análise estática, executadas em ordem. Projetos *open-source* fazem uso da CI por meio de ferramentas como o GitHub Actions[1], que permitem automatizar fluxos de trabalho no processo de desenvolvimento.

Embora trabalhos anteriores correlacionem a prática de CI à qualidade de código e abordem métricas relacionadas ao produto [Vasilescu et al. 2015, Rahman et al. 2018] e ao fluxo de desenvolvimento [Cassee et al. 2020], ou ainda à qualidade da suíte de testes [Silva et al. 2024, Saraiva et al. 2023], **ainda são escassas as análises que buscam compreender os impactos da adoção da CI sobre a qualidade estrutural do código, mais especificamente, sobre métricas que refletem características de projeto orientado a objetos**. Com isso, este trabalho propõe uma investigação baseada nas seis métricas CK [Chidamber and Kemerer 1994]: WMC (*Weighted Methods per Class*), DIT (*Depth of Inheritance Tree*), NOC (*Number of Children*), CBO (*Coupling Between Object classes*), RFC (*Response For a Class*) e LCOM (*Lack of Cohesion in Methods*), além das métricas *fan-in* e *fan-out* [Sutoyo et al. 2025], buscando uma análise mais detalhada dos efeitos da CI a nível estrutural do código.

Ainda que a cultura de integração contínua de mudanças exista desde a década de 1970, as práticas de CI ganharam destaque apenas após os anos 2010, tornando-se mais discutidas, aplicadas e estudadas no âmbito da engenharia de software [Felidré et al. 2019]. Além disso, a melhor colaboração entre desenvolvedores e qualidade de código é reportada como um dos principais benefícios com a adoção do processo, além da detecção antecipada de possíveis problemas acarretados por alterações no código [Sghaier and Sahraoui 2023]. Tendo em mente a crescente popularidade do processo de CI atrelada à complexidade de sistemas modernos, compreender os impactos e benefícios dessa prática diretamente sobre o código-fonte torna-se relevante para equipes de desenvolvimento que buscam aprimorar seus processos internos e a qualidade intrínseca do produto de software final.

Dessa maneira, **o objetivo geral deste trabalho é investigar os impactos da Integração Contínua na qualidade do código-fonte de projetos *open-source* em Java hospedados no GitHub, com base em métricas estruturais e de análise estática**. A escolha dessa linguagem justifica-se pela sua popularidade e por suas características de orientação a objetos, enquanto a escolha por projetos *open-source* deve-se ao acesso ao código-fonte, o que viabiliza a coleta e mensuração das métricas. Para tanto, são investigados os seguintes objetivos específicos: quantificar a variação da qualidade da estrutura interna do código antes e apgithós a implementação de CI; correlacionar a qualidade da estrutura interna do código com a frequência de uso de CI no projeto; e, por fim, avaliar a correlação entre a quantidade de *steps* em *pipelines* e a qualidade do código e a qualidade do código inferida a partir das métricas coletadas.

Como resultados, observa-se que a adoção da Integração Contínua exerce in-

[1] `https://docs.github.com/pt/actions`

fluência positiva sobre a progressão de algumas métricas estruturais de qualidade. Em particular, verificou-se uma melhora nas métricas CBO, RFC e *fan-out* após a introdução da CI, enquanto as demais métricas analisadas não apresentaram variações estatisticamente significativas. Dessa forma, o trabalho oferece informações que podem apoiar desenvolvedores e líderes técnicos acerca da adoção do processo de CI em projetos.

No contexto deste artigo, a Seção 2 apresenta conceitos essenciais sobre Integração Contínua e análise estática de código. A Seção 3 sintetiza estudos anteriores que serviram como base para esta pesquisa, analisando suas contribuições e identificando lacunas que justificam a realização deste estudo. Já a Seção 4 descreve em detalhes os procedimentos adotados, incluindo as estratégias de coleta e análise de dados, as métricas analisadas e os critérios empregados na seleção dos repositórios analisados. Na Seção 5, são exibidos os resultados obtidos, enquanto a Seção 6 discute suas implicações. A Seção 7 aborda as ameaças à validade do estudo. Por fim, a Seção 8 apresenta a conclusão e indica possíveis pesquisas futuras.

2. Fundamentação Teórica

Esta seção tem como objetivo apresentar os principais conceitos que embasam este estudo. Na Seção 2.1, é apresentado o conceito de Integração Contínua. A Seção 2.2 contempla as métricas CK, adotadas neste trabalho como base para a análise.

2.1. Integração Contínua

A prática de Integração Contínua consiste em automatizar a compilação, construção e execução de testes ou verificações personalizáveis a cada mesclagem do código, permitindo ciclos de lançamento menores, mais rápidos e mais confiáveis [Shahin et al. 2017]. A adoção da prática a nível organizacional é acompanhada de diversos benefícios, como o *feedback* mais rápido durante o processo de desenvolvimento, detecção de falhas de forma antecipada e melhora nos âmbitos de extensibilidade e manutenibilidade do sistema [Fowler 2024].

O núcleo dessa automação é a *pipeline* de CI, um fluxo de trabalho que define uma sequência de *steps* (passos) que devem ser executados pelo sistema. Cada *step* é uma unidade de trabalho atômica e sequencial que especifica uma ação a ser realizada. Tipicamente, incluem ações como o *checkout* do código do repositório, a instalação de dependências, a compilação do projeto, a execução de suítes de testes e, até mesmo, a execução de análises estáticas de código para verificar padrões de estilo e segurança, atuando desde o preparo do ambiente até a verificação final da qualidade.

Alguns exemplos de ferramentas de CI populares são *GitHub Actions*, *TravisCI*, *CircleCI* e *Jenkins* [Golzadeh et al. 2022]. Esses softwares permitem automatizar fluxos de trabalho no processo de desenvolvimento por meio de arquivos de configuração, que geralmente são armazenados no sistema de controle de versão do projeto, como o *GitHub*.

2.2. Métricas CK

As métricas CK, propostas por Chidamber e Kemerer originalmente em 1994, formam um conjunto de indicadores voltados à avaliação de características estruturais em projetos orientados a objetos, apresentados na Tabela 1. Esse conjunto abrange aspectos de complexidade, acoplamento, coesão e herança, permitindo uma análise quanti-

tativa da qualidade do código sob o ponto de vista do *design* e implementação de classes [Chidamber and Kemerer 1994].

Tabela 1. Descrição das métricas CK propostas [Chidamber and Kemerer 1994]

Métrica	Descrição
WMC	Soma dos pesos dos métodos que indica complexidade da classe.
DIT	Nível da classe na hierarquia, evidenciando herança e reutilização.
NOC	Quantidade de subclasses diretas que herdam da classe.
CBO	Quantidade de outras classes às quais a classe está conectada. Associada com acoplamento.
RFC	Total de métodos que podem ser acionados em resposta a uma chamada.
LCOM	Mede quão relacionados estão os métodos dentro da classe. Indica falta de coesão.

Dessa maneira, a escolha do conjunto para o presente estudo é baseada em sua capacidade de capturar propriedades relevantes do software de forma automatizada e comparável entre projetos estudados. No contexto deste trabalho, as métricas CK desempenham papel central na avaliação da qualidade estrutural do código-fonte antes e após a adoção da prática de *CI*, permitindo identificar possíveis efeitos da análise automatizada contínua sobre o *design* interno dos sistemas analisados.

3. Trabalhos Relacionados

Nesta seção, são apresentados e discutidos trabalhos que investigam a adoção da CI em diferentes contextos da Engenharia de Software. Os estudos selecionados exploram os impactos da CI em aspectos como cobertura de testes, segurança, ocorrência de *bugs*, gerenciamento de *pull requests* e dinâmicas de *code review*.

Rahman et al. (2018) investigam os impactos da adoção de ferramentas de CI em 150 projetos de código aberto e 123 projetos privados, com o objetivo de entender como a adoção da CI influencia o processo de desenvolvimento de software. Os repositórios que utilizam a CI foram identificados por meio da presença de arquivos de configuração. Nos projetos *open-source*, observou-se um aumento significativo na resolução de *bugs* (aumento de 5 vezes) e *issues* (aumento de aproximadamente 2,4 vezes) após a adoção da CI. No entanto, para os projetos privados, não foram observadas alterações significativas, o que é atribuído ao uso ineficiente do mecanismo de *feedback* da CI, destacando que, nesses projetos, os desenvolvedores tendem a realizar *commits* com menor frequência. A pesquisa se relaciona ao presente trabalho ao analisar os impactos da CI sobre atributos de qualidade do produto, mensurados por meio da resolução de *bugs* e *issues*, complementando a proposta deste artigo ao focar na avaliação de métricas estruturais de código em projetos *open-source*.

Saraiva et al. (2023) investigam a relação entre a adoção de CI e a cobertura de código em projetos de software. O estudo se baseia na análise de 60 projetos *open-source* do *GitHub*, sendo 30 que adotaram CI e 30 que nunca adotaram. Os autores aplicaram o método estatístico *Regression Discontinuity Design* (RDD), utilizado para avaliar os impactos de uma intervenção, permitindo estimar o impacto da introdução da CI na cobertura de código ao longo do tempo. Por fim, os resultados indicam que 50%

dos repositórios que utilizam CI tiveram um aumento na cobertura de código, contra 10% dos repositórios que não a utilizam. Assim, os autores indicam a adoção de CI para maior consistência e estabilidade de projetos. Este artigo se relaciona ao presente por também investigar os impactos da CI na qualidade do software, embora sob a ótica da cobertura de testes, além de utilizar o método RDD, que é aplicada de maneira similar pelo presente estudo.

Santos et al. (2022) investigam como cinco subpráticas da CI afetam a produtividade e qualidade em projetos *open-source* no *GitHub*. O artigo reuniu 90 projetos a partir dos repositórios mais estrelados no GitHub em 15 linguagens e de projetos indexados no *SonarQube*. Os repositórios foram selecionados com base nos seguintes filtros: mais de 100 *stars*, 100 *PRs*, 100 *issues*, tamanho mínimo de 10 MB, exclusão de *forks*, uso do *Travis-CI*, ao menos dois anos de atividade contínua e presença de *builds* e *commits* em 80% dos meses analisados. Foram identificadas fortes correlações a partir do teste de *Spearman* de *Commit Activity* (ρ = 0,388) e *Build Activity* (ρ = 0,274) com o aumento de produtividade. Além disso, maior *Build Activity* e melhor *Build Health* estão associados a menos *issues* rotuladas como *bugs*. A pesquisa complementa este trabalho por apresentar uma metodologia semelhante de mineração de repositórios no *GitHub* e por avaliar métricas de qualidade de código, embora com enfoque diferente. O artigo associa qualidade ao número de *bugs* reportados, enquanto este trabalho considera métricas estruturais do código.

Cassee et al. (2020) investigam o impacto da adoção da CI no processo de *code review*, com foco em como a automação influencia a comunicação e o esforço dos revisores. Foram analisados 685 repositórios de código aberto hospedados no *GitHub* que adotaram *Travis CI*, totalizando 1.214.826 *pull requests*. A metodologia emprega o método RDD, mencionado anteriormente, para comparar a atividade de revisão antes e depois da introdução da CI. Os resultados indicam uma redução de aproximadamente um comentário por *pull request* após a adoção da CI, sugerindo menor necessidade de comunicação entre os revisores. No entanto, essa diminuição não está associada a uma redução nas mudanças feitas no código durante o processo de revisão, o que reforça a ideia de que a CI funciona como um "auxiliador silencioso", capaz de aliviar a carga dos revisores sem comprometer a qualidade do código. Este trabalho se aproxima da presente pesquisa ao explorar os efeitos da CI sobre a dinâmica de desenvolvimento, porém voltado para as revisões de código, além de também utilizar o método RDD.

Wikanty et al. (2023) investigam a correlação entre más práticas arquiteturais (AS, do inglês *Architecture Smells*) e métricas de qualidade de software orientado a objetos, com base no conjunto de métricas CK. O estudo analisa três projetos *open-source* desenvolvidos em *C#* e hospedados no *GitHub*, com variação no número de linhas de código e classes. Os resultados indicam correlação forte entre certas ASs e métricas CK: "Dependência Cíclica" teve correlação de 1.0 e "Dependência Instável" de 0.8. No sentido inverso, "Funcionalidade Dispersa" apresentou correlação de -0.8 com as métricas CK. Entre as métricas analisadas, *DIT* e *WMC* foram as mais correlacionadas com AS (valores de 1.0 e 0.74, respectivamente). Por fim, o estudo conclui que há evidências claras de que estruturas arquiteturais problemáticas afetam negativamente aspectos estruturais do código, como complexidade e coesão. A pesquisa se relaciona ao presente trabalho por analisar métricas CK em projetos *open-source* no *GitHub* com o objetivo de avaliar

a qualidade do código desenvolvido. Entretanto, este trabalho realiza uma análise que considera não apenas a arquitetura do software, mas também o fator temporal associado ao evento causal da introdução da CI no projeto.

4. Materiais e Métodos

Nesta seção, são apresentados os materiais e métodos utilizados para realizar este estudo. Este trabalho é classificado como uma pesquisa quantitativa, utilizando de estatística descritiva, pois busca identificar a relação de causa e efeito entre a implementação de CI e a qualidade do código, avaliada por meio de comparações e de análise estatística [Wohlin et al. 2012].

A presente pesquisa é conduzida a partir da mineração de repositórios *open-source* hospedados no *GitHub*, com código-fonte desenvolvido em *Java* e que adotaram a CI após o início de seu ciclo de vida. As etapas do estudo estão organizadas conforme o fluxo apresentado na Figura 1, que também orienta a estrutura da seção atual. A definição do escopo na linguagem *Java* se justifica por sua forte orientação a objetos e por sua consolidação como tecnologia predominante no desenvolvimento de aplicações corporativas, sendo amplamente adotada em pesquisas modernas de análise estática [Antoniadis et al. 2020]. Além disso, o estudo foca exclusivamente na ferramenta *GitHub Actions*, selecionada por sua ampla popularidade e adoção na comunidade de desenvolvimento *open-source* [Golzadeh et al. 2022]. A mineração de dados é realizada por meio de *scripts* desenvolvidos em *Python*.

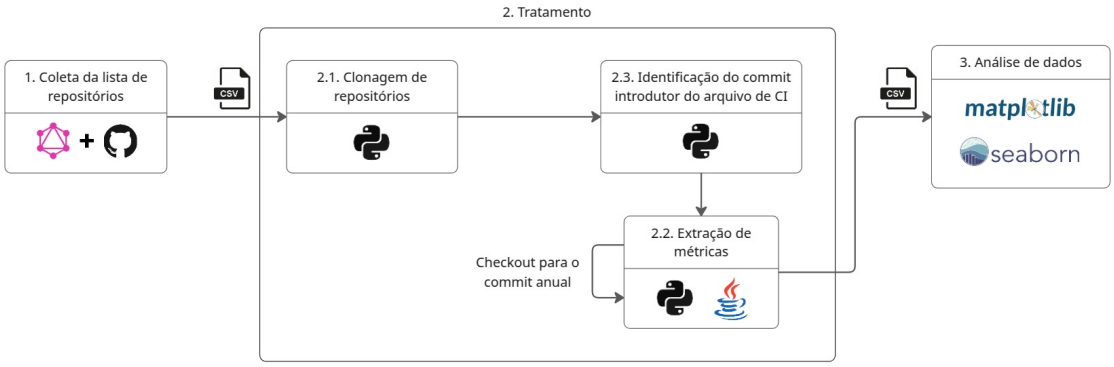

Figura 1. Etapas da metodologia

4.1. Coleta de Lista de repositórios

A primeira etapa do método consiste na coleta de dados, realizada por meio de uma ferramenta que consome a *API* (do inglês, *Application Programming Interface*) *GraphQL* do *GitHub* para recuperar os repositórios a serem analisados. Para isso, são aplicados filtros com o objetivo de excluir projetos de pequeno porte ou experimentais [Santos et al. 2022]. Nesse contexto, são considerados apenas repositórios que atendam às seguintes condições:

- quantidade de estrelas maior que 100
- quantidade de *pull requests* maior que 100

- tamanho maior que 10 MB
- não sejam *forks*

Ademais, por definição de escopo do presente estudo, foi adicionada a condição de que os repositórios devem utilizar o *GitHub Actions* para fins de Integração Contínua. Tal condição é identificada por meio da verificação de um arquivo *YAML* correspondente na estrutura do repositório. Além disso, o repositório deve ter no mínimo seis anos de idade, para que seja considerado estável. O arquivo de CI deve ter sido criado ao menos três anos após a criação do repositório e três anos antes do último *commit*, garantindo tempo suficiente para a análise anual.

As *URLs* dos repositórios selecionados são armazenadas para serem utilizadas na etapa subsequente. Essa modularização permite a execução em etapas da coleta de dados do trabalho, evitando complexidade de código e *overhead* de execução, além de criar *checkpoints* entre as etapas.

4.2. Tratamento

A segunda etapa de análise do ciclo de vida recupera os repositórios coletados e realiza sua clonagem. Em seguida, conforme ilustrado na Figura 2, para cada projeto, é identificado o *commit* de introdução do arquivo de configuração da *pipeline* de Integração Contínua. Esta identificação visa permitir a análise separada do repositório antes e depois da adoção da CI.

A partir da data de adoção da CI, são definidos seis momentos de medição: três pontos anuais anteriores e três pontos anuais posteriores a essa data. Em cada um desses seis períodos, são extraídas as métricas correspondentes ao estado do projeto. Este método permite comparar a evolução do sistema com e sem a CI. Assim, a análise consiste em:

1. Clonar o repositório.
2. Identificar a data de adoção da CI.
3. Extrair as métricas em seis momentos distintos ao longo da linha do tempo do projeto.

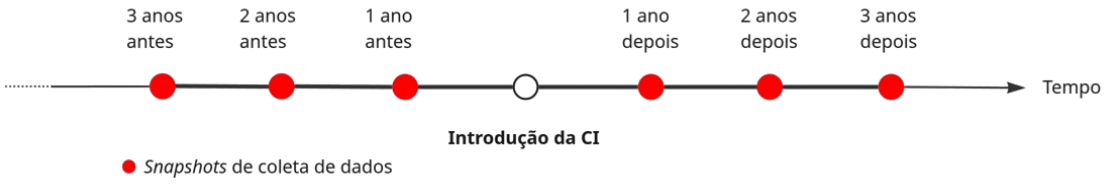

Figura 2. Análise individual do ciclo de vida do repositório

Para a extração de dados, além das métricas CK levantadas na na Seção 2.2 para análise da qualidade estrutural do código, também são coletadas duas variáveis auxiliares e duas métricas estruturais adicionais. Dentre as auxiliares, a primeira, frequência anual de *pull requests*, é utilizada como estimativa da intensidade de uso da CI, visto que a criação de *pull requests* é um gatilho comum para a execução de *pipelines*. A segunda variável corresponde ao número de *steps* da *pipeline* do projeto e complementa a análise de correlação com as métricas CK. Já as métricas estruturais, *fan-in* e *fan-out*, fornecem

uma visão complementar do código, indicando relações de dependência e acoplamento entre classes do projeto [Sutoyo et al. 2025].

A coleta das métricas CK, NCLOC, *fan-in* e *fan-out* é realizada utilizando a biblioteca *CK Tools* [Aniche 2015]. A coleta da frequência de *pull requests* no último ano é realizada a partir da *API GraphQL* do *GitHub*. Já para a coleta do número de *steps*, é realizada uma análise direta nos arquivos de configuração de CI. Os dados coletados são armazenados para posterior análise dos resultados.

A última etapa do processo de análise é a validação da integridade dos dados coletados. Inicialmente, é verificada a consistência das extrações, buscando repositórios que não possuam exatamente três entradas em cada período de tempo (anterior e posterior à CI). Para aqueles repositórios em que a extração inicial de métricas falha, um reprocessamento é executado. Nesse reprocessamento, é implementado um mecanismo de *fallback*: se a extração falhar em um período específico, o *checkout* do código é feito na data de uma semana após o período da falha, e uma nova tentativa de execução do *CK Tools* é realizada. Caso o erro persista, o *checkout* é adiantado em mais uma semana. Essa medida lida com possíveis erros de compilação momentâneos do código, que podem impedir a execução da biblioteca externa *CK Tools*.

4.3. Análise de dados

As métricas coletadas são analisadas por meio da comparação entre as métricas coletadas anterior e posteriormente à adoção da CI. A avaliação dos resultados é conduzida com o auxílio da técnica estatística RDD (*Regression Discontinuity Design*), que permite estimar o impacto causal da introdução da CI. Essa abordagem explora a descontinuidade natural presente na linha do tempo dos projetos, marcada pelo momento em que a CI passa a ser adotada. Dessa maneira, é possível identificar mudanças nas métricas estruturais levadas em consideração entre os dois períodos coletados.

Por fim, são utilizados recursos gráficos para apoiar a visualização dos resultados. Diagramas de dispersão com linhas de regressão são utilizados para apresentar correlações entre os dados. Complementarmente, *boxplots* são empregados para comparar a distribuição das métricas antes e depois da adoção da CI, permitindo observar variações na mediana, dispersão e presença de valores atípicos.

5. Resultados

Nesta seção, são apresentados os resultados obtidos através da coleta de dados. Ao todo, foram selecionados 222 repositórios que atendem aos critérios estabelecidos por este estudo. O conjunto analisado apresenta um volume significativo de popularidade e atividade, com média de 5386,83 estrelas e mediana de 2354, sendo o *spring-framework* o mais popular, com 59232 estrelas; além de uma média de 1844,66 *forks* e mediana de 763,5, indicando repositórios relevantes e frequentemente reutilizados. Observa-se também um fluxo contínuo de contribuições externas, refletido pela média de 3333,15 *pull requests* e mediana de 1192,5. Em relação à maturidade dos projetos, ambas a idade média e mediana dos repositórios é de 12 anos, o que evidencia o longo histórico de evolução. De forma geral, o *dataset* é majoritariamente composto por projetos voltados à comunidade de desenvolvimento, reunindo bibliotecas, ferramentas e *frameworks* de uso geral, que somam 185 repositórios, enquanto o restante distribui-se entre categorias mais

específicas, como bancos de dados, sistemas de processamento de dados, ferramentas de teste, segurança e CI.

A Figura 3 apresenta a evolução temporal das médias por repositório das métricas WMC, DIT, NOC, CBO, RFC, LCOM, *fan-in* e *fan-out* ao longo de três anos antes (indicados por A1 a A3) e três anos depois da adoção da CI (indicados por D1 a D3). Observa-se que, para a maioria das métricas de complexidade e acoplamento (CBO, WMC, DIT, NOC, RFC, *fan-in* e *fan-out*), o aumento percentual anual do valor médio (μ) se torna menos acelerado ou, em alguns casos, reduz após a introdução da CI.

Em primeiro lugar, a métrica *CBO* teve uma variação acumulada de $+5.5\%$ no período pré-CI (A1-A3), e passou a apresentar uma variação de de $+1.0\%$ no período pós-CI (D1-D3), indicando uma redução em seu valor médio por repositório. De maneira similar, *WMC* e *RFC* também mostram uma tendência de desaceleração, com *WMC* caindo de $+3.3\%$ para $+2.6\%$, e *RFC* passando de $+6.8\%$ para $+3.5\%$. A métrica *fan-out* também ilustra essa tendência de desaceleração, registrando $+5.5\%$ antes e $+1.0\%$ depois da introdução da CI Essas observações sugerem que a automação pode estar contendo a expansão do acoplamento e da complexidade para a maioria das métricas analisadas.

Já as métricas *NOC* e *fan-in* apresentam uma redução em sua média, com *NOC* passando de um crescimento de $+4.0\%$ para uma redução de -1.9% e *fan-in* reduzindo de $+0.2\%$ para -0.8%. Por outro lado, *LCOM* apresentou uma aceleração na variação acumulada, de $+3.1\%$ para $+10.2\%$. Por outro lado, a métrica *DIT* (com -0.4% e -0.3%) apresentou uma tendência de estabilidade antes e depois do evento causal, sugerindo que a CI não causou um impacto significativo sobre a profundidade da hierarquia de herança.

Para avaliar se os resultados são estatisticamente significativos, foi aplicado o método estatístico RDD, comparando a taxa de crescimento percentual médio das métricas, do primeiro ao terceiro ano, antes e após a adoção da CI. Os resultados da análise, apresentados na Tabela 2, revelam que a adoção da CI teve um efeito significativo na desaceleração do crescimento das métricas *LCOM* e *RFC*, apresentando redução estatisticamente significativa nas taxas de crescimento após a introdução da CI. A magnitude desse efeito foi quantificada e interpretada por meio do *d* de *Cohen*, índice que mede o tamanho do efeito de uma intervenção, padronizando a diferença entre as médias pela variação dos dados. Os resultados indicaram efeitos médios ($d > 0.2$) para as métricas analisadas.

Tabela 2. Análise RDD: Impacto da CI no crescimento das métricas

Métrica (média)	n	Δ% antes	Δ% após	Efeito RDD	d de Cohen	Força	p-valor
CBO	444	3.55	2.96	-0.59	-0.048	Trivial	0.611
WMC	444	3.36	1.60	-1.76	-0.123	Trivial	0.196
DIT	444	0.25	-0.02	-0.27	-0.040	Trivial	0.672
LCOM	**444**	**31.33**	**10.89**	**-20.44**	**-0.218**	**Pequena**	**0.022***
RFC	**444**	**5.34**	**2.06**	**-3.27**	**-0.249**	**Pequena**	**0.009***
Fan-out	444	3.55	2.97	-0.57	-0.046	Trivial	0.626
Fan-in	444	4.79	-0.95	-5.74	-0.151	Trivial	0.113
NOC	444	5.01	1.54	-3.47	-0.095	Trivial	0.319

*p $< 0,05$; Δ% = variação percentual média anual

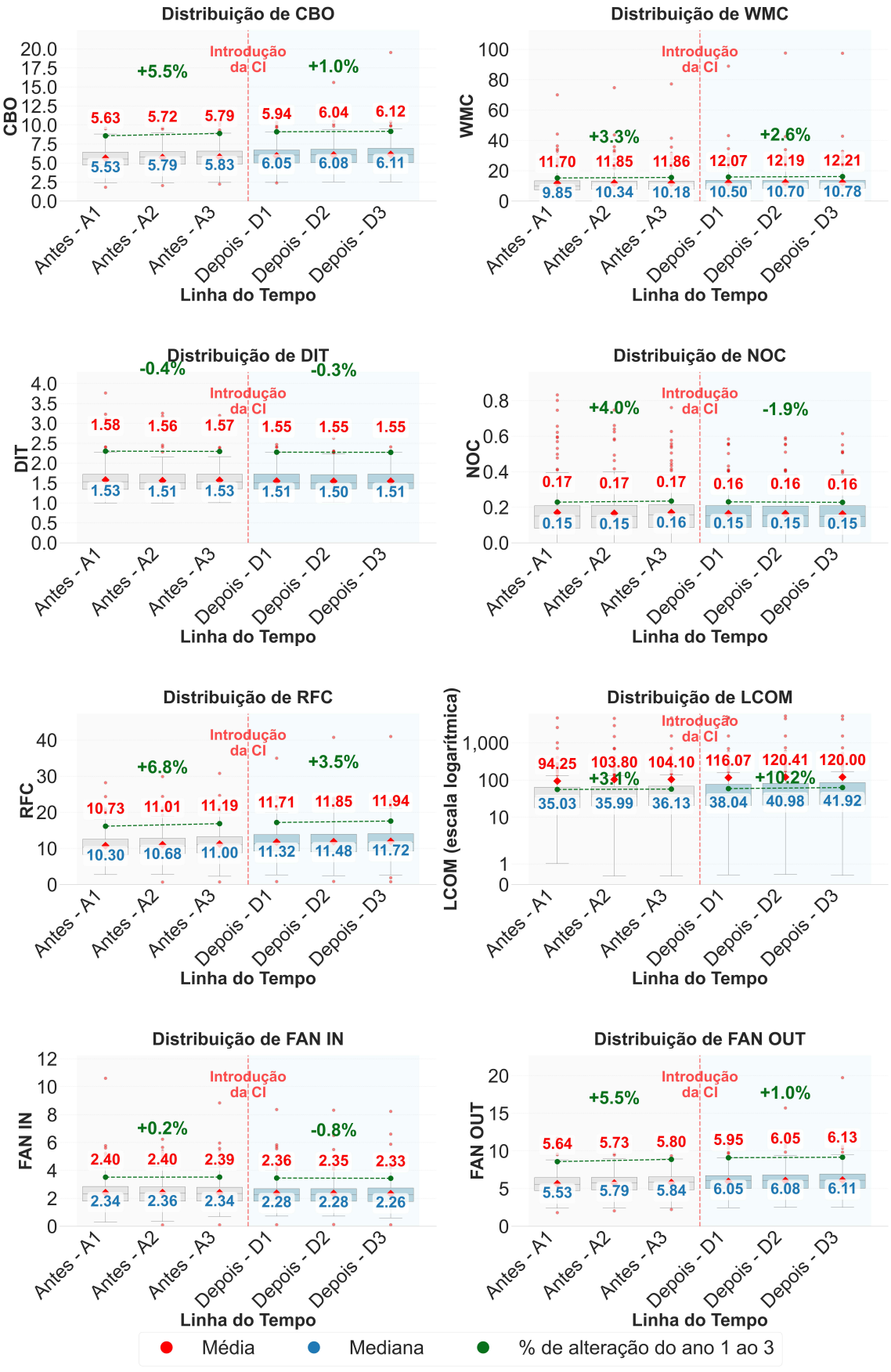

Figura 3. Distribuição das métricas ao longo do tempo - Antes e após a CI

A Tabela 3 apresenta os resultados detalhados da análise de correlação entre a quantidade de *pull requests* criados no terceiro ano após a adoção da CI e as métricas de qualidade de código. Foram analisados 219 repositórios dos 222 selecionados, visto que não foi possível coletar os dados dos repositórios restantes por meio da API do *GitHub*.

Entre as métricas analisadas, três apresentaram correlações classificadas como pequenas: CBO com $r = 0,149$ ($p = 0,027$), RFC com $r = 0,110$ ($p = 0,105$) e *fan-out* com $r = 0,148$ ($p = 0,029$). Essas correlações positivas, embora fracas, sugerem uma tendência de aumento nessas métricas de complexidade e acoplamento conforme aumenta o número de *steps*. A significância estatística de 95% observada em CBO e *fan-out*, mesmo com correlações pequenas, reforça a existência de uma associação fraca entre a quantidade de *steps* e o aumento da complexidade código. Por outro lado, a correlação com a métrica RFC não apresenta significância estatística. As demais métricas analisadas (WMC, DIT, NOC, LCOM e *fan-in*) apresentaram correlações triviais e não significativas. Esses resultados indicam ausência de relação consistente entre essas métricas e a frequência de criação de *pull requests*.

Tabela 3. Correlação entre PRs por ano (Ano 3) e métricas de qualidade

Métrica	n	Pearson r	p-valor	Força
CBO	**219**	**0,149**	**0,027***	**Pequena**
WMC	219	-0,032	0,633	Trivial
DIT	219	0,072	0,292	Trivial
NOC	219	0,076	0,262	Trivial
RFC	219	0,110	0,105	Pequena
LCOM	219	-0,041	0,546	Trivial
FAN IN	219	-0,063	0,354	Trivial
FAN OUT	**219**	**0,148**	**0,029***	**Pequena**

*p $< 0,05$

A Tabela 4 apresenta os resultados detalhados da análise de correlação entre o número de *steps* nas *pipelines* de CI no terceiro ano após a adoção e as métricas de qualidade de código. Assim como a análise voltada para *pull-requests*, três métricas apresentaram correlações classificadas como pequenas: CBO com $r = 0,161$ ($p = 0,017$), RFC com $r = 0,125$ ($p = 0,064$) e *fan-out* com $r = 0,159$ ($p = 0,018$). Essas correlações fracas sugerem uma tendência de aumento nessas métricas de complexidade e acoplamento associada ao aumento do número de *steps*. A significância estatística de 95% observada em CBO e *fan-out*, mesmo com correlações pequenas, reforça a existência de uma associação fraca entre a quantidade de *steps* e o aumento da complexidade código. Por outro lado, a correlação com a métrica RFC não apresenta significância estatística. As demais métricas analisadas (WMC, DIT, NOC, LCOM e *fan-in*) apresentaram correlações triviais e não significativas.

Tabela 4. Correlação entre *steps* de CI (Ano 3) e métricas de qualidade

Métrica	n	Pearson r	p-valor	Força
CBO	**220**	**0.161**	**0.017***	**Pequena**
WMC	220	-0.047	0.485	Trivial
DIT	220	0.079	0.244	Trivial
NOC	220	0.074	0.272	Trivial
RFC	220	0.125	0.064	Pequena
LCOM	220	-0.047	0.493	Trivial
FAN IN	220	-0.033	0.625	Trivial
FAN OUT	**220**	**0.159**	**0.018***	**Pequena**

*p $< 0,05$

6. Discussão

A comparação entre os períodos antes e após a CI mostra que embora sua adoção não impeça o crescimento natural da complexidade em projetos, ela desacelera significativamente esse processo. Destacam-se as métricas *LCOM* e *RFC*, que apresentaram reduções de 65,24% e 61,42% no crescimento médio, respectivamente, em comparação ao período pré-CI. Esses resultados evidenciam que o uso da CI pode auxiliar no desenvolvimento de software, contribuindo para a manutenibilidade dos projetos ao longo do tempo. A contenção do crescimento da coesão entre métodos (*LCOM*) e da complexidade de resposta (*RFC*) sugere que práticas de CI promovem código mais modular e testável, facilitando futuras modificações e reduzindo o débito técnico.

Em relação à correlação da frequência de uso da CI com a qualidade de código no período após a CI, os resultados obtidos indicam que a frequência de *builds* mensurada pela quantidade de *PRs* criados no período de um ano apresentaram correlação baixa ou inexistente com as métricas estruturais. Conforme apresentado anteriormente, apenas duas das oito métricas analisadas, CBO e *fan-out*, apresentaram correlações positivas estatisticamente significativas, enquanto as demais métricas (WMC, DIT, NOC, RFC, LCOM e FAN IN) não obtiveram significância estatística. Essa constatação contraria a hipótese inicial deste estudo, que previa a existência de uma relação positiva entre a intensidade de uso da CI e a melhoria da qualidade do código, mensurada por meio das métricas CK e das métricas complementares *fan-in* e *fan-out*.

Já em relação à correlação do número de *steps* de CI com a qualidade de código no período após a CI, os dados obtidos também indicam que o número de *steps* das *pipelines* de CI apresentaram correlação baixa ou inexistente com as métricas estruturais. De forma semelhante à análise de frequência de *builds*, apenas duas das oito métricas analisadas, CBO e *fan-out*, apresentaram correlações positivas estatisticamente significativas, enquanto as demais métricas (WMC, DIT, NOC, RFC, LCOM e FAN IN) não obtiveram significância estatística. Essa constatação também contraria a hipótese inicial deste estudo, que previa a existência de uma relação positiva entre a quantidade de verificações em uma *pipeline* de CI e a melhoria da qualidade do código, mensurada por meio das métricas selecionadas. Entretanto, é importante destacar que o número de *steps* em uma *pipeline* não necessariamente reflete a complexidade ou o rigor dos testes automatizados executados.

Esses resultados sugerem que a frequência de *builds* e número de *steps* de um

projeto são insuficientes para prever as características estruturais do código. A ausência de correlações relevantes indica que a qualidade do código é influenciada por um conjunto mais amplo de fatores, incluindo práticas de engenharia de software, experiência e cultura da equipe e arquitetura inicial do sistema.

7. Ameaças à validade

Esta seção apresenta as potenciais ameaças à validade do estudo e as estratégias de mitigação utilizadas.

Primeiramente, quanto à validade de construção, a adoção da CI pelos repositórios foi identificada pelo *commit* de introdução do arquivo de configuração do *GitHub Actions*, o que não garante que a CI foi imediatamente utilizada de forma eficaz. Além disso, a frequência de utilização da CI foi estimada a partir da quantidade de *pull requests* criados, o que fornece apenas uma aproximação da intensidade real de uso da CI.

Em relação à validade interna, o emprego da CI pode ter ocorrido simultaneamente a outros eventos no ciclo de vida do projeto, como inclusão de novas funcionalidades, refatorações e manutenções. Esses fatores podem impactar as métricas de qualidade e, consequentemente, afetar a correlação observada entre a adoção da CI e as métricas analisadas. Foi aplicada a análise de descontinuidade na regressão (RDD) para isolar o efeito da CI, no entanto, outros eventos que ocorram no mesmo período da adoção podem influenciar os resultados.

Por fim, em relação à validade externa, os filtros aplicados aos repositórios favorecem a análise de projetos *open-source* maduros em Java, que utilizam *GitHub Actions*, em sua maioria bibliotecas, *frameworks* e ferramentas voltadas à comunidade de desenvolvimento. Portanto, os resultados podem não ser generalizáveis para projetos com outras linguagens de programação, outras ferramentas de CI ou de outros perfis funcionais, como aplicações corporativas fechadas ou softwares orientados a domínios específicos.

8. Conclusão

Neste trabalho, foram analisados os impactos da adoção da CI na qualidade estrutural do código-fonte de projetos *open-source* desenvolvidos em *Java*. Para tanto, foram coletadas métricas CK, *fan-in* e *fan-out*, considerando períodos anteriores e posteriores à introdução da CI. Como resultados, observou-se que a CI influencia positivamente a progressão das métricas CBO, WMC, RFC, LCOM e *fan-out*, enquanto não produz efeitos significativos sobre as demais métricas avaliadas.

Como proposta para trabalhos futuros, sugere-se a ampliação da análise para incluir uma maior variedade de repositórios, como projetos desenvolvidos em outras linguagens de programação ou que utilizem diferentes ferramentas de CI; assim como que sejam fechados em um escopo funcional predefinido. Recomenda-se também investigar outros fatores que possam influenciar o impacto da CI na qualidade do código, como os tipos de testes executados nas *pipelines*.

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-bruno-duarte-e-leticia-fraga`

Referências

Aniche, M. (2015). *Java code metrics calculator (CK)*. Available in https://github.com/mauricioaniche/ck/.

Antoniadis, A., Filippakis, N., Krishnan, P., Ramesh, R., Allen, N., and Smaragdakis, Y. (2020). Static analysis of java enterprise applications: frameworks and caches, the elephants in the room. PLDI 2020, page 794–807, New York, NY, USA. Association for Computing Machinery.

Cassee, N., Vasilescu, B., and Serebrenik, A. (2020). The silent helper: The impact of continuous integration on code reviews. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 423–434.

Chidamber, S. and Kemerer, C. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493.

Felidré, W., Furtado, L., Costa, D. A. d., Cartaxo, B., and Pinto, G. (2019). Continuous integration theater. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10.

Fowler, M. (2024). Continuous integration. `https://martinfowler.com/articles/continuousIntegration.html`. Acessado em: 25 de março de 2025.

Golzadeh, M., Decan, A., and Mens, T. (2022). On the rise and fall of ci services in github. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 662–672.

Rahman, A., Agrawal, A., Krishna, R., and Sobran, A. (2018). Characterizing the influence of continuous integration: empirical results from 250+ open source and proprietary projects. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Software Analytics*, SWAN 2018, page 8–14, New York, NY, USA. Association for Computing Machinery.

Santos, J., Alencar da Costa, D., and Kulesza, U. (2022). Investigating the impact of continuous integration practices on the productivity and quality of open-source projects. In *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '22, page 137–147, New York, NY, USA. Association for Computing Machinery.

Saraiva, D., Da Costa, D. A., Kulesza, U., Sizílio, G., Neto, J. G., Coelho, R., and Nagappan, M. (2023). Unveiling the relationship between continuous integration and code coverage. In *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, pages 247–259.

Sghaier, O. B. and Sahraoui, H. (2023). A multi-step learning approach to assist code review. In *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 450–460.

Shahin, M., Babar, M. A., and Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943.

Silva, E., Rodrigues, R., Oliveira, J., Boechat, D., and Tavares, C. (2024). Evaluating test quality in github repositories: A comparative analysis of ci/cd practices using github actions. In *Anais do XII Workshop de Visualização, Evolução e Manutenção de Software*, pages 45–55, Porto Alegre, RS, Brasil. SBC.

Sutoyo, E., Avgeriou, P., and Capiluppi, A. (2025). Tracing the lifecycle of architecture technical debt in software systems: A dependency approach. In *2025 IEEE 22nd International Conference on Software Architecture (ICSA)*, pages 199–209.

Vasilescu, B., Yu, Y., Wang, H., Devanbu, P., and Filkov, V. (2015). Quality and productivity outcomes relating to continuous integration in github. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, page 805–816, New York, NY, USA. Association for Computing Machinery.

Wikantyasa, I. M. A., Kurniawan, A. P., and Rochimah, S. (2023). C k metric and architecture smells relations: Towards software quality assurance. In *2023 14th International Conference on Information Communication Technology and System (ICTS)*, pages 13–17.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering*. Computer Science. Springer Berlin Heidelberg.