

Análise Comparativa de Repositórios com Releases Rápidos e Lentos: Implicações para a Qualidade do Código

Alfredo L. Vieira¹, Bruno E. Gomes de Azevedo²
Estevão de F. Rodrigues³, Vinícius S. de Oliveira⁴

¹¹Departamento de Engenharia de Software e Sistemas da Informação (DES)
Pontifícia Universidade Católica de Minas Gerais (PUC-Minas)
Belo Horizonte – MG – Brasil

Abstract. *Release strategies are strongly connected to the software development process and impact the final product quality. Project managers often doubt which strategy to adopt, evaluating the tradeoffs of each. This study compares the influence of rapid release (RR) and slow release (SR) strategies on code quality in open-source projects. We analyze three quality aspects: vulnerabilities, errors (bugs and code smells), and rework (technical debt) using the SonarQube tool. The objective is to provide an organized view of the impacts of each strategy, aiming to assist project managers in their decision-making.*

Resumo. *Estratégias de lançamento de releases estão fortemente conectadas ao processo de desenvolvimento e impactam a qualidade final do produto. Gerentes de projeto frequentemente têm dúvidas sobre qual estratégia adotar, avaliando os tradeoffs de cada uma. Este estudo compara a influência de estratégias de rapid release (RR) e slow release (SR) na qualidade do código de projetos open-source. Analisamos três aspectos de qualidade: vulnerabilidades, erros (bugs e code smells) e retrabalho (dívida técnica), utilizando a ferramenta SonarQube. O objetivo é fornecer uma visão organizada dos impactos de cada estratégia, visando auxiliar gestores de projeto em suas tomadas de decisão.*

1. Introdução

Atualmente, no contexto de desenvolvimento de software, existem diversas estratégias de lançamento de releases, estas estratégias estão fortemente conectadas ao processo de desenvolvimento do software em si, e como consequência, impactam na qualidade do produto final, com essa pluralidade de estratégias, gerentes/responsáveis de projetos de software facilmente se encontram na dúvida sobre qual estratégia de lançamento/desenvolvimento abordar, sendo obrigados a avaliarem os *tradeoffs* oferecidos por cada estratégia, onde uma decisão equivocada pode levar a custos elevados de manutenção, insatisfação e perda de competitividade no mercado.. Este estudo tem como comparar a influência que estratégias de release possuem sobre a qualidade do código (mais precisamente rapid releases e slow releases), podemos identificar rapid releases, como uma organização de lançamento de releases, onde um projeto possui releases lançadas com uma mediana de tempo inferior a 35 dias [Joshi and Chimalakonda 2019]. Estas influências, mais especificamente na qualidade do código, analisando vulnerabilidades, erros e retrabalho do projeto. Após a obtenção destes resultados, será possível possuir uma visão

mais organizada dos impactos das rapid e das slow releases, a fim de auxiliar gestores de projetos a tomarem suas decisões de lançamentos de seus produtos.

O objetivo deste trabalho é comparar as influências das rapid releases com as influências das slow releases em projetos *open-source*, mais especificamente, em questão de qualidade, mais precisamente, abordaremos 3 aspectos de qualidade, sendo eles, vulnerabilidade, quantidade de erros, e retrabalho nos sistemas open-source, assim, espera-se responder as seguintes questões de pesquisa ao final do trabalho.

O estudo se estrutura da seguinte forma: Na sessão 2, será abordada os trabalhos relacionados, estes trabalhos serão brevemente explicados, e sua relação com o contexto deste estudo será definida, na próxima sessão, a sessão 3, será apresentada a metodologia, nesta sessão será explicado como o estudo foi executado, passo a passo. Na sessão 4, será apresentado os resultados deste estudo, onde serão apresentados o produto da pesquisa realizada.

- **RQ1:** O RRC torna as releases mais vulneráveis?
- **RQ2:** Erros são mais comuns em releases de RRC?
- **RQ3:** O retrabalho é maior em sistemas que utilizam RRC, do que em sistemas que utilizam releases lentas?

Neste trabalho, utilizaremos a ferramenta SonarQube, uma ferramenta de análise estática de código, a fim de obter algumas métricas relacionadas aos repositórios, métricas estas que serão úteis para o trabalho desenvolvido..

2. Trabalhos relacionados

Este estudo foi realizado se baseando em uma literatura pré-existente, segue abaixo os trabalhos acadêmicos que foram utilizados como base:

1. **RapidRelease - A Dataset of Projects and Issues on Github with Rapid Releases** Este trabalho nos apresenta o dataset RapidRelease, criado para facilitar estudos empíricos sobre engenharia de release e desenvolvimento ágil em projetos open-source. O dataset foi construído a partir de mineração de dados no GitHub e filtrando projetos com um ciclo médio de release, que ocorre entre 5 e 35 dias, os autores chamam esse ciclo de *rapid release*. O conjunto final contém um dataset muito rico, que pode servir de base para uma gama diversa de estudos na área. Este artigo se relaciona com o trabalho atual se baseando no fato que utilizamos uma parte de sua metodologia, ou seja, serviu de base para nos entregar o conceito de *rapid release*, que foi utilizado, e considerado nas filtragens realizadas ao longo deste estudo.
2. **Modern Release Engineering in a Nutshell** **Why Researchers should Care** Este artigo argumenta que, embora a engenharia de release moderna tenha evoluído rapidamente impulsionada pela indústria (com práticas como Continuous Delivery), a pesquisa acadêmica da área ainda é considerada "atrasada". O trabalho introduz as seis principais fases do pipeline de engenharia de release

(Integração, Integração Contínua, Especificação do Sistema de Build, Infraestrutura como Código, Deployment e Release). O principal objetivo é fornecer para os pesquisadores um guia para pesquisas futuras na área. Este artigo se correlaciona com o trabalho atual, a partir do estudo sobre releases que ele fornece, ele foi utilizado como base para entendimentos iniciais acerca dessa etapa do desenvolvimento de software.

3. Systematic literature review on the impacts of agile release engineering practices

Esta revisão sistemática de literatura teve como objetivo analisar os impactos diretos e indiretos das práticas de Agile Release Engineering (ARE), bem como a forma como foram investigadas empiricamente. Utilizando termos como "rapid release" e "continuous deployment", foram analisados 71 estudos primários. Os resultados indicaram que as práticas de releases rápidas promovem menores lead times e melhor comunicação entre as equipes, mas também revelaram desafios em questões como gerência de mudanças e de qualidade. O resultado mostra que práticas de releases rápidas melhoram a eficiência do processo, mas que é necessário outros estudos mais abrangentes, para identificar efeitos em outras áreas do processo. Este trabalho se correlaciona com o estudo atual servindo como uma "base de conhecimento acadêmico" acerca de *releases* rápidas, onde podemos embasar nossas hipóteses.

4. An Empirical Evaluation of the Relationship between Technical Debt and Software Security

Este artigo tem como objetivo identificar se a dívida técnica pode possuir impactos na segurança do produto, não apenas em questões apenas de qualidade. O estudo foi realizado através de análises estáticas em repositórios java, utilizando ferramentas como FindBugs e FindSecurityBug. Como resultado, foi identificado que dívida técnica pode sim estar relacionada diretamente a questões de segurança, e que pode ser usada como um indicador de segurança, mostrando que produtos com alto nível de dívida técnica estão propensos a possuir falhas de segurança. Este estudo se correlaciona com o trabalho atual partindo do ponto que dívida técnica e segurança são dois pontos em que o estudo apresentado possui seu foco, ou seja, ele nos dá uma base sobre questões de segurança, a partir da dívida técnica.

5. Impact of Continuous Integration on Software Quality and Productivity

Este artigo busca fornecer evidências empíricas para os benefícios da Integração Contínua (CI). O estudo de caso, realizado em uma grande empresa, correlaciona a maturidade da CI com a melhoria na qualidade e produtividade. A pesquisa testa a hipótese de que a CI detecta defeitos mais cedo no ciclo de desenvolvimento e melhora a produtividade, utilizando métricas internas. Os resultados visam validar que a CI é uma prática eficaz para aprimorar a qualidade e a eficiência no desenvolvimento de software. O resultado do artigo confirma a hipótese de que a Integração Contínua aumenta o nível de qualidade do produto, pois ela garante que erros são detectados o mais cedo possível, adiantando a correção dos mesmos, e aumentando a eficiência do processo como um todo. Este artigo se correlaciona ao trabalho apresentado neste estudo, a partir do ponto de vista da qualidade, quando olhada por pontos de vista de Integração contínua (CI), pois o estudo atual tem seu

grande foco na entrega de releases, e suas influências na qualidade do produto.

Em resumo, a literatura aborda a definição de rapid releases [Joshi et al.], a necessidade de estudos empíricos na área [Siavvas et al.], e o impacto da CI na qualidade [Bhattacharya]. Nosso trabalho se diferencia e complementa esta literatura ao verificar diretamente os impactos desse padrão de releases na qualidade do código, a partir de verificações estáticas de segurança, bugs e retrabalho, e comparando com repositórios que utilizando as *slow releases*.

3. Metodologia

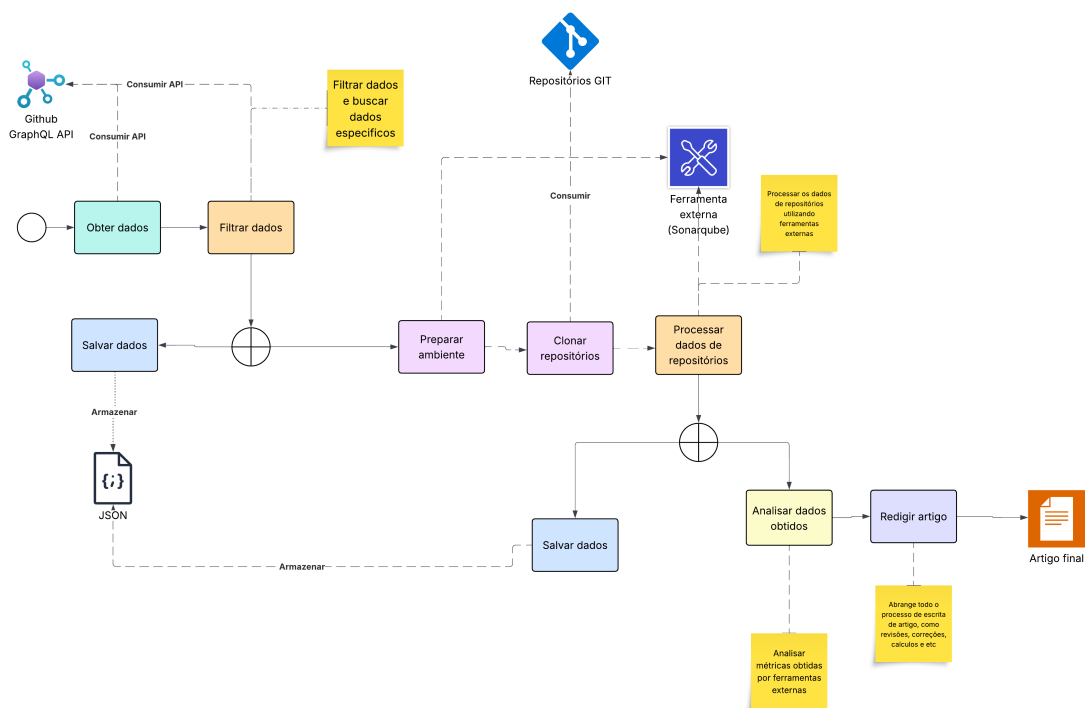


Figura 1. Visão Geral da Metodologia do Estudo

A metodologia adotada para responder às questões de pesquisa propostas é detalhada a seguir e resumida na figura 1, primeiramente, foram minerados 1400 repositórios do github, utilizando sua API GraphQL, onde os mesmos serão alvo das análises. Para a execução da mineração de dados, foi considerada a definição realizada por, onde são considerados repositórios que aplicam *rapid release*, repositórios que possuem um intervalo médio de 5 a 35 dias entre cada release [Joshi and Chimalakonda 2019] (para a nossa execução, consideramos a mediana deste intervalo possuindo este valor), e para que possamos comparar estes valores, estipulamos um valor mediano de 50 ou superior, para o intervalo entre as releases de repositórios que aplicam a *slow release*.

Fora a caracterização entre *slow* e *rapid* dos repositórios, aplicamos os seguintes filtros na mineração dos repositórios:

- Foram buscados os top 1400 repositórios em quesitos de *stars*
- Cada repositório deverá possuir 50+ forks e 50+ stars no github
- Cada repositório deverá possuir no mínimo 19 colaboradores e 19 repositórios

Com essa caracterização de dataset em mente, foi desenvolvido um script em python para fazer a mineração inicial destes repositórios, nessa etapa foram buscados um total dos top 5000 repositórios do github em questão de *stars*. A partir deste dataset inicial obtido, foi desenvolvido outro script, agora, para realizar a filtragem, nessa filtragem, o script coletou informações das releases, e informações sobre número de colaboradores nos projetos. Esse script também foi responsável por fazer a filtragem dos repositórios coletados previamente, agora gerando o dataset definitivo em formato JSON.

Análise SonarQube: Para realizar a análise do SonarQube, análise esta que realizará a verificação estática do código, fornecendo métricas necessárias para responder as questões propostas neste estudo, inicialmente foi necessário a preparação do ambiente para a execução da ferramenta. Segue abaixo a lista de métricas que foram obtidas através da análise do SonarQube:

- bugs
- vulnerabilities
- code_smells
- sqale_index (Dívida técnica)
- coverage
- duplicated_lines_density
- ncloc (Linhas de código não comentadas)
- complexity (Complexidade Ciclométrica)
- cognitive_complexity
- reliability_rating (Nota de Confiabilidade)
- security_rating (Nota de Segurança)
- sqale_rating (Nota de Manutenibilidade)

Inicialmente, foi implementado um script em python, que itera sobre o dataset obtido na etapa anterior, e primeiramente, realiza um clone do respectivo repositório utilizando a ferramenta GIT, em seguida, o SonarQube é executado, analisando estaticamente este repositório. buscando as métricas setadas anteriormente. Após a análise de todos os 1400 repositórios, os resultados são salvos em formato .CSV, e armazenados, para que sejam consumidos por uma ferramenta de visualização de dados (BI).

Para o desenvolvimento do estudo, foram identificadas as seguintes hipóteses acerca das RQs, e as variáveis:

- **Hipóteses sobre Vulnerabilidade (Q1)**

$H_{0,1}$: Não há diferença significativa na vulnerabilidade entre projetos com *rapid release* (RRC) e projetos com *slow release* (SL).

$H_{1,1}$: Projetos com *rapid release* (RRC) são significativamente mais vulneráveis que projetos com *slow release* (SL).

- **Hipóteses sobre Erros e Confiabilidade (Q2)**

$H_{0,2}$: Não há diferença significativa na ocorrência de erros (bugs e code smells) entre projetos com *rapid release* (RRC) e projetos com *slow release* (SL).

$H_{1,2}$: Erros são significativamente mais comuns em projetos com *rapid release* (RRC) que em projetos com *slow release* (SL).

- **Hipóteses sobre Retrabalho e Dívida Técnica (Q3)**

$H_{0,3}$: Não há diferença significativa no retrabalho (Dívida Técnica) entre projetos com *rapid release* (RRC) e projetos com *slow release* (SL).

$H_{1,3}$: O retrabalho é significativamente maior em projetos com *rapid release* (RRC) que em projetos com *slow release* (SL).

Tabela 1. Experimental Setup: Variáveis do Estudo

Tipo de Variável	Variável	Métricas de Medição
Independente	Tipo de Ciclo de Release (<i>rapid release</i> vs. <i>slow release</i>)	Não se aplica (Variável Categórica)
Dependente	Qualidade (Segurança)	<i>vulnerabilities</i> , <i>security_rating</i>
Dependente	Qualidade (Erros/Confiabilidade)	<i>bugs</i> , <i>reliability_rating</i> , <i>code_smells</i>
Dependente	Qualidade (Retrabalho/Manutenibilidade)	<i>sqale_index</i> , <i>duplicate_lines_density</i> , <i>sqale_rating</i>

4. Resultados

1. Q1: O RRC torna as releases mais vulneráveis?

Ao analisarmos os resultados relativos a segurança, conseguimos identificar primeiramente, no ranking de segurança, que é uma nota dada pelo SonarQube, baseando-se na taxa de tempo gasto analisando pontos de segurança no código, em relação ao resto do código. Podemos identificar que repositórios que seguiram o modelo de *slow release* possuíram uma taxa superior a 80% com notas 1, que é a nota mais alta possível, esta porcentagem é um pouco menor que 80% nos repositórios *rapid*. Podemos encontrar uma taxa mínima, de repositórios com nota 2, beirando 1%.

Já sobre o quesito vulnerabilidade, podemos encontrar uma média superior em repositórios que seguem *slow release* sendo superior a 8, enquanto repositórios que seguem *rapid release*, este valor é inferior a 4.0

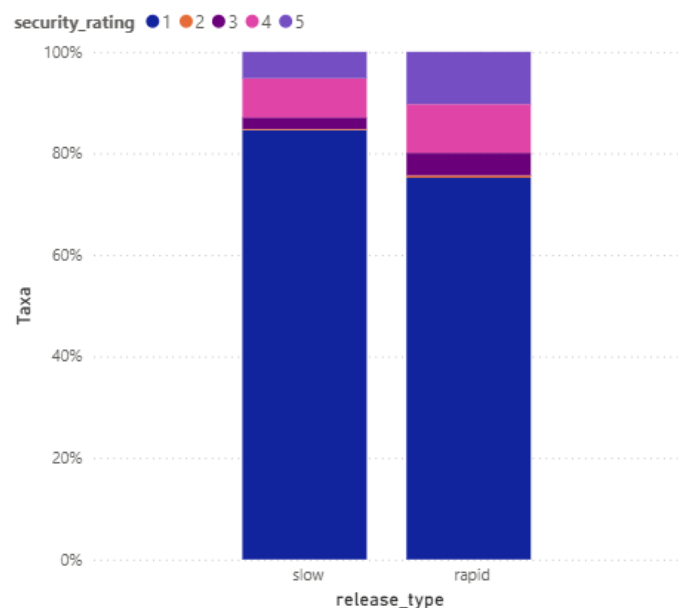


Figura 2. Taxa de notas de segurança por tipo de release

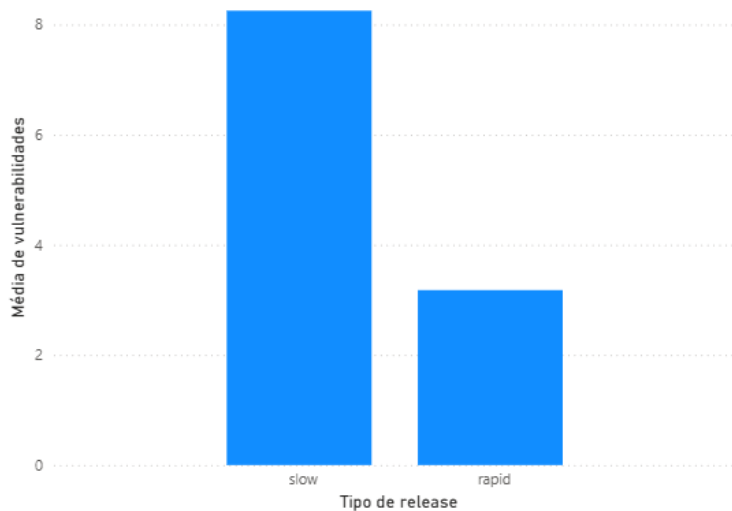


Figura 3. Média de vulnerabilidade por tipo de release

2. Q2: Erros são mais comuns em releases de RRC?

Primeiramente, observando o boxplot relacionado aos bugs, obtemos uma diferença considerável, onde repositórios baseados em *rapid release* possuem um número consideravelmente maior, ultrapassando 15.0, enquanto repositórios que seguem o modelo de *slow release* possuem quase 1/3 deste valor, possuindo um

valor um pouco acima de 5.0.

Já em questão de notas de confiabilidade, onde 1 é a maior nota, e 5 é a menor, podemos observar que repositórios que seguem as *slow releases* possuem uma taxa pouco abaixo de 40% de repositórios com nota 1.0, enquanto esse número nas *rapid releases* é um pouco acima de 20.0. A vantagem também continua quando observamos a taxa de repositórios com notas 4.0 e 5.0, onde as *rapid releases* possuem mais exemplares.

Agora falando de *code smells*, que representa o número absoluto de code smells encontrados na análise estática, observamos no boxplot, que repositórios que seguem *rapid release* possuem um número mais elevado, possuindo sua mediana quase alcançando os 300.0. Enquanto a mediana dos repositórios que seguem *slow release*, é 3x menor, sendo um pouco menos que 100.0.

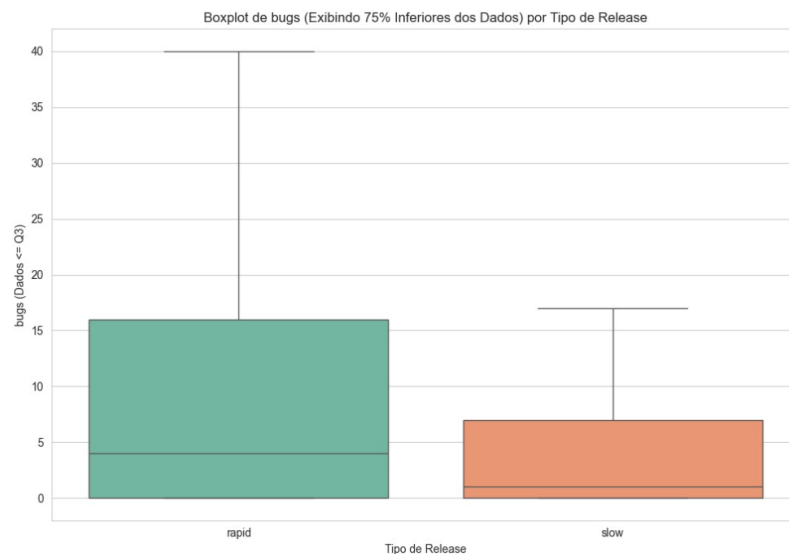


Figura 4. Boxplot de bugs por tipo de release

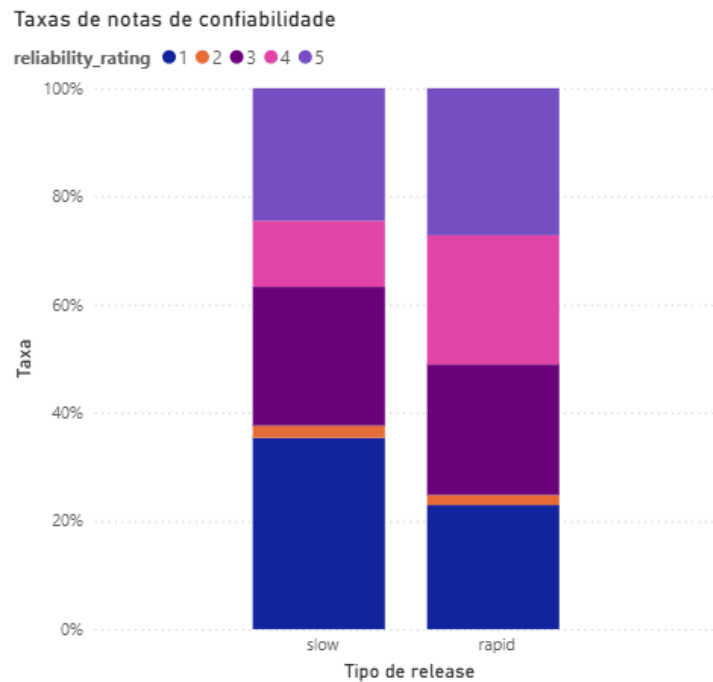


Figura 5. Taxa de notas de confiabilidade por tipo de release

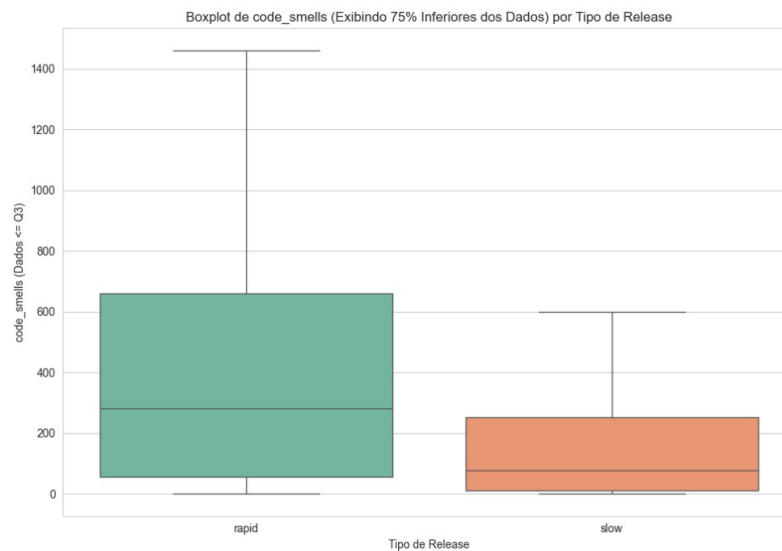


Figura 6. Boxplot de code smells

3. Q3: O retrabalho é maior em sistemas que utilizam RRC, do que em sistemas que utilizam releases lentas??

Para analisar o retrabalho, calculamos a mediana dos valores *sqale index*, que é o cálculo do esforço total para corrigir problemas de manutenibilidade, e o *duplicate line density*, que como o nome diz, é o nível de densidade de linhas duplicadas

em cada repositório. Ao analisarmos os gráficos obtidos, podemos observar que ambos os valores são maiores para repositórios que seguem as *rapid releases*, sendo no *sqale index*, alcançando uma taxa quase 150% maior.

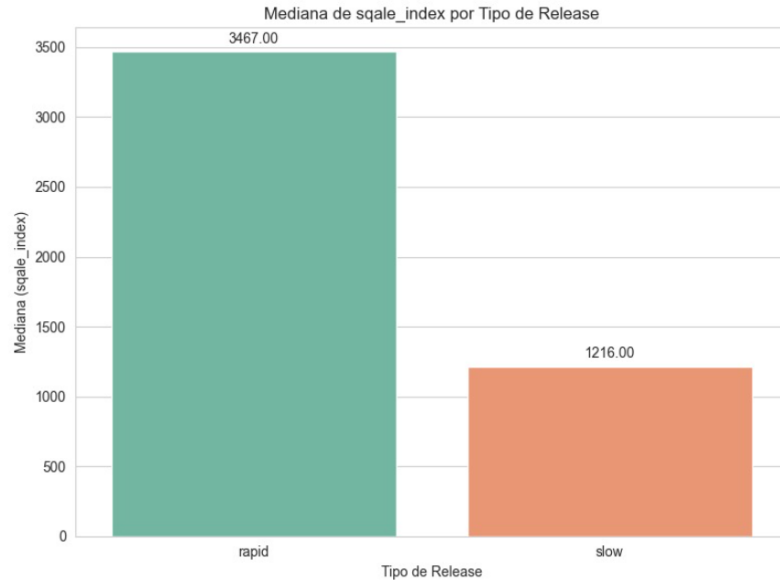


Figura 7. Boxplot do Sqale Index (Dívida Técnica)

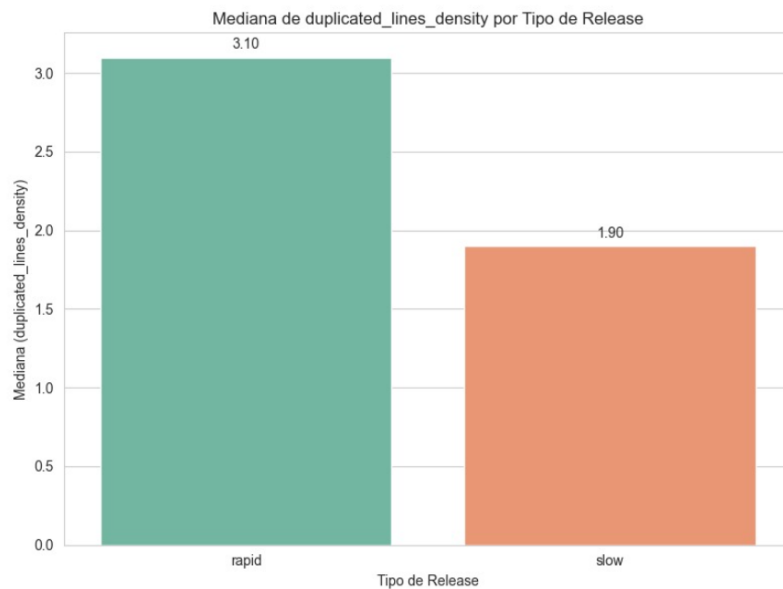


Figura 8. Boxplot da Densidade de Linhas Duplicadas

Referências

Joshi, S. D. and Chimalakonda, S. (2019). Rapidrelease - a dataset of projects and issues on github with rapid releases. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pages 587–591.