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

Alfredo Luis, Vinicius Salles, Bruno Evangelista

12 de novembro de 2025

1 Introdução

A prática de releases rápidos (Rapid Release Cycles - RRC) tem se tornado cada vez mais comum no desenvolvimento de software moderno, especialmente em projetos open-source. Releases rápidos referem-se a ciclos de desenvolvimento com intervalos curtos, tipicamente entre 5 e 35 dias entre releases, em contraste com metodologias tradicionais que utilizam ciclos mais longos.

No ecossistema de desenvolvimento de software de código aberto, plataformizado em grande parte pelo GitHub, a adoção de RRC levanta questões importantes sobre seus impactos na qualidade do código. Este estudo tem como objetivo principal analisar e comparar repositórios que utilizam releases rápidos com aqueles que adotam releases lentos (ciclos superiores a 60 dias), investigando as implicações dessa escolha metodológica na qualidade final dos sistemas.

Buscamos identificar se o desenvolvimento baseado em RRC pode prejudicar a segurança e qualidade de um sistema, analisando aspectos como vulnerabilidades, erros e retrabalho. A análise será conduzida comparando dois grupos de repositórios do GitHub: um conjunto de projetos com releases rápidos e outro com releases lentos, permitindo uma avaliação comparativa das práticas e seus resultados.

2 Metodologia

A construção do dataset foi realizada utilizando dados coletados do GitHub através de sua API, focando em projetos populares da plataforma. O estudo baseia-se em um dataset existente de 994 projetos com releases rápidos, originalmente coletado em estudos anteriores, e foi expandido para incluir um conjunto comparável de projetos com releases lentos.

2.1 Dataset de Releases Rápidos

O dataset de releases rápidos foi filtrado para incluir projetos que atendiam aos seguintes critérios:

- Popularidade mínima: mais de 50 estrelas e 50 forks
- Tempo médio de release entre 5-35 dias
- Pelo menos 19 colaboradores distintos
- Pelo menos 19 releases distintos
- Suporte a 18 linguagens de programação diferentes

2.2 Dataset de Releases Lentos

Para permitir a análise comparativa, foi coletado um conjunto de dados equivalente para projetos que utilizam releases lentos. Os critérios de seleção foram mantidos similares, com exceção do tempo médio entre releases:

- Popularidade mínima: mais de 50 estrelas e 50 forks
- Tempo médio de release superior a 60 dias
- Pelo menos 19 colaboradores distintos
- Pelo menos 19 releases distintos

2.3 Dataset Final para Análise das RQs

Para a análise das questões de pesquisa, foi selecionado um subconjunto balanceado dos dados coletados, visando garantir uma comparação equilibrada e estatisticamente robusta entre as duas abordagens de release. O dataset final utilizado neste estudo é composto por **200 repositórios**, sendo:

- **100 repositórios (50%)** com releases rápidos (RRC)
- **100 repositórios (50%)** com releases lentos

Esta distribuição balanceada de 50% para cada categoria foi adotada para eliminar vieses relacionados ao tamanho amostral e permitir comparações diretas mais confiáveis entre os dois grupos. Os 100 repositórios de cada categoria foram selecionados aleatoriamente dos conjuntos maiores coletados, mantendo a diversidade de linguagens de programação e tamanhos de projeto presentes nos datasets originais.

2.4 Coleta de Métricas de Qualidade

Com os 200 repositórios selecionados, foram coletadas métricas de qualidade de código utilizando ferramentas de análise estática, incluindo o SonarQube. As métricas foram extraídas para cada release dos projetos selecionados, permitindo uma análise temporal e comparativa entre os dois grupos. Os dados foram posteriormente visualizados utilizando ferramentas de Business Intelligence para facilitar a interpretação dos resultados.

2.5 Questões de Pesquisa

Este trabalho de pesquisa tem como objetivo avaliar os efeitos das RRCs na qualidade de sistemas open-source, comparando-os com sistemas lançados em releases lentas. A investigação busca correlações entre a metodologia de release adotada e aspectos de qualidade do código. As questões de pesquisa são:

Vulnerabilidades e Segurança:

- **RQ 01:** O RRC torna as releases mais vulneráveis?

Erros e Confiabilidade:

- **RQ 02:** Erros são mais comuns em releases de RRC?

Retrabalho e Dívida Técnica:

- **RQ 03:** O retrabalho é maior em sistemas que utilizam RRC, do que em sistemas que utilizam releases lentas?

3 Métricas Utilizadas

Para investigar as questões de pesquisa propostas, o trabalho define um conjunto de métricas específicas que quantificam cada uma das dimensões de análise. Essas métricas são essenciais para a coleta de dados e para a subsequente análise comparativa, permitindo que os conceitos de "segurança", "confiabilidade"e "retrabalho"sejam medidos de forma consistente em ambos os grupos de repositórios.

As métricas definidas são:

3.1 Vulnerabilidades e Segurança (RQ 01)

- **vulnerabilities:** Número total de vulnerabilidades de segurança identificadas no código
- **security_rating:** Classificação de segurança do projeto (escala A a E)
- **security_hotspots:** Pontos críticos de segurança que requerem revisão manual

3.2 Erros e Confiabilidade (RQ 02)

- **bugs:** Número de bugs identificados pela análise estática
- **reliability_rating:** Classificação de confiabilidade do código (escala A a E)
- **code_smells:** Número de code smells detectados (indicadores de problemas de manutenibilidade)

3.3 Retrabalho e Dívida Técnica (RQ 03)

- **technical_debt:** Dívida técnica total estimada (em dias de trabalho)
- **sqale_index:** Índice SQALE de dívida técnica
- **duplicated_lines_density:** Densidade percentual de linhas de código duplicadas

4 Resultados Obtidos

4.1 RQ 01: O RRC torna as releases mais vulneráveis?

Nesta questão buscamos identificar se existe correlação entre a adoção de releases rápidos e o número de vulnerabilidades de segurança encontradas nos sistemas. A análise comparativa entre os 100 projetos com RRC e os 100 projetos com releases lentos permite avaliar se a pressão por entregas frequentes compromete a segurança do código.

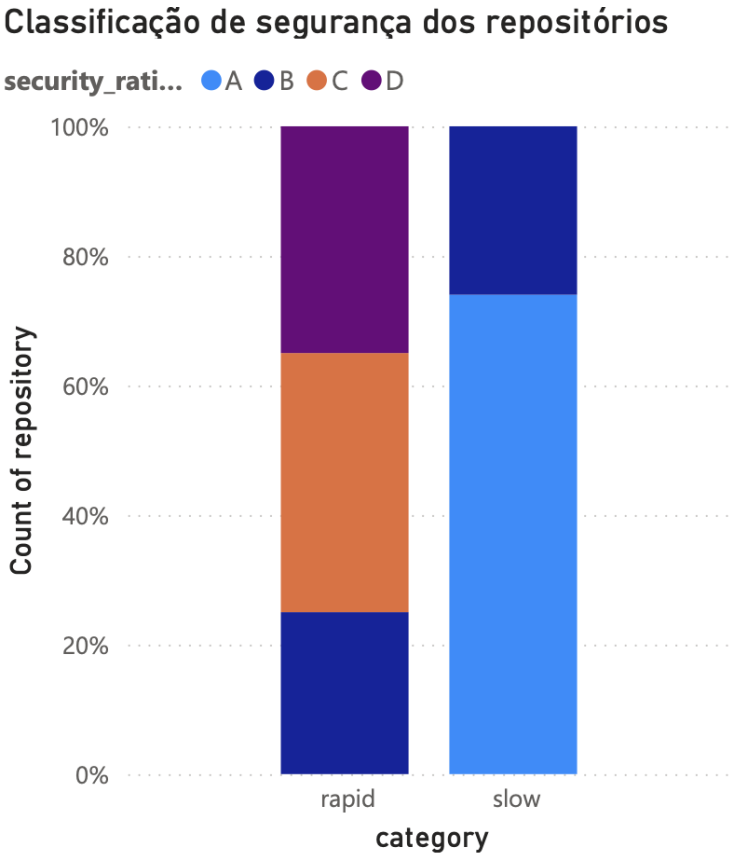


Figura 1: Classificação de segurança dos repositórios

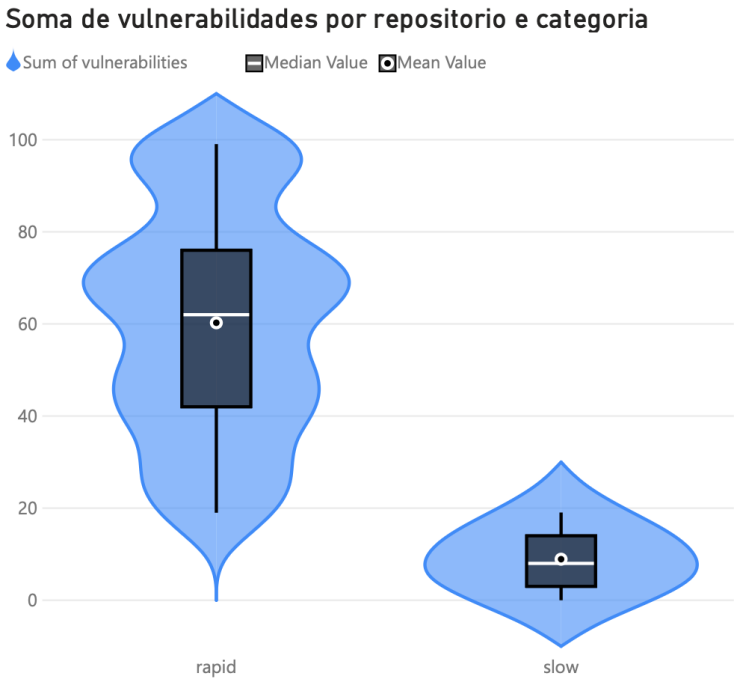


Figura 2: Soma de vulnerabilidades por repositório e categoria

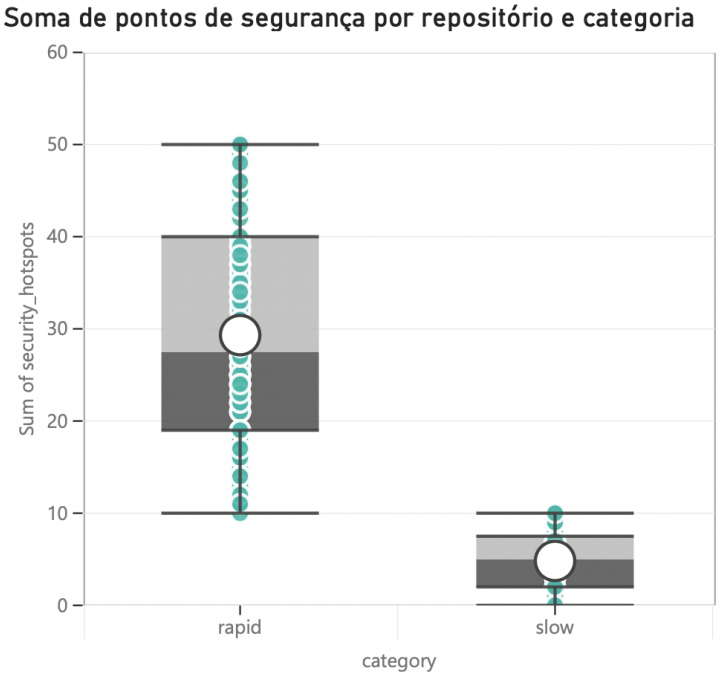


Figura 3: Soma de pontos de segurança por repositório e categoria

Ao analisar as métricas de segurança apresentadas nas Figuras 1, 2 e 3, observamos resultados que favorecem a hipótese inicial de que releases rápidos comprometeriam a segurança dos sistemas.

A distribuição de classificações de segurança (security_rating) mostra que existe uma diferença consideravel entre repositorios que utilizam slow releases em comparação a repositorios que utilizam rapid releases, podemos observar a maior diferença a partir da parcela de repositorios com classificação C e D em rapid releases, classificações estas que não foram identificadas na amostra de repositorios que utilizaram slow releases..

Em relação ao número absoluto de vulnerabilidades, os dados mostram que repositórios com releases rapidas apresentam uma grande desvantagem com relação aos repositorios que utilizam releases lentas, podemos observar uma média consideravelmente maior em rapid releases.

Quanto aos pontos críticos de segurança (security_hotspots), que indicam áreas do código que requerem revisão manual cuidadosa, os valores também são desvatnajosos para repositorios slow release, podemos entender que repositorios que utilizam releases rápidas possuem um numero maior de regiões sensiveis no código, o que é um grande indicativo de falhas de segurança..

Conclusão RQ01: Apoiando a expectativa inicial, podemos observar que num geral, releases rápidas tendem a deterioriar repositórios em questão de segurança, isso pode ser explicado por uma serie de fatores, como o tempo reduzido

de revisão/correção de código, menor tempo gasto com refinamento/planejamento arquitetural, entre outras questões que a rapidez nos lançamentos podem trazer, contrariando slow releases, onde existe mais tempo para tratar estas questões.

4.2 RQ 02: Erros são mais comuns em releases de RRC?

Esta questão investiga a relação entre a frequência de releases e a ocorrência de bugs e problemas de confiabilidade. Analisamos se o desenvolvimento acelerado característico de RRC resulta em maior número de defeitos no código.

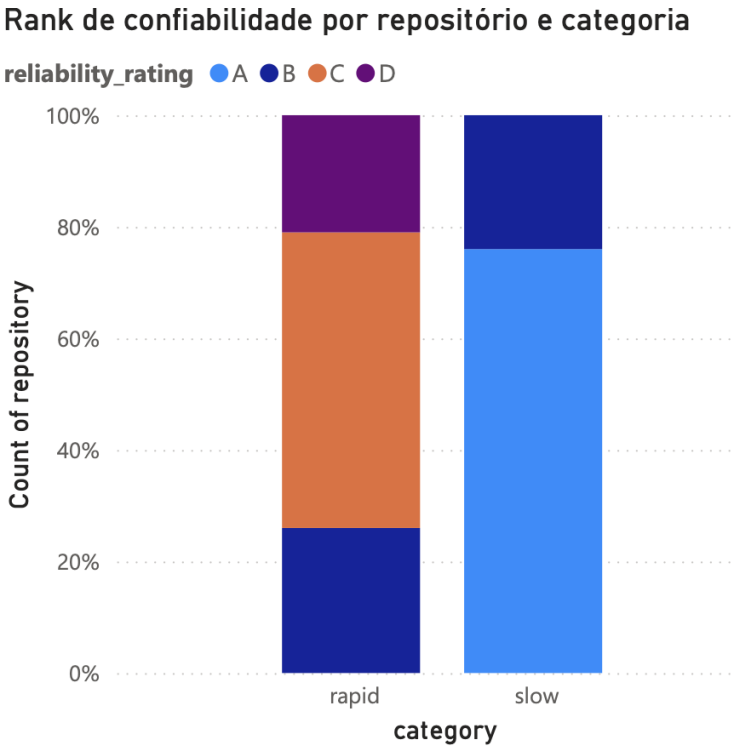


Figura 4: Rank de confiabilidade por repositório e categoria

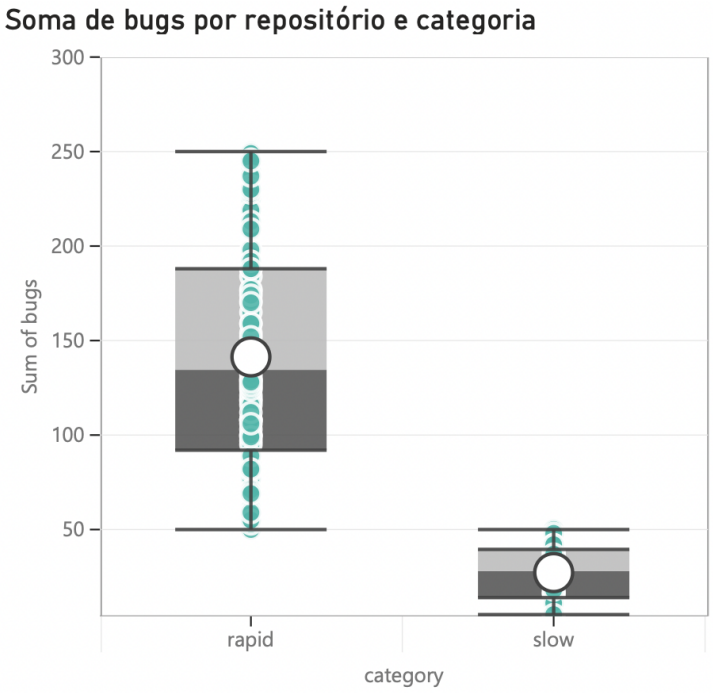


Figura 5: Soma de bugs por repositório e categoria

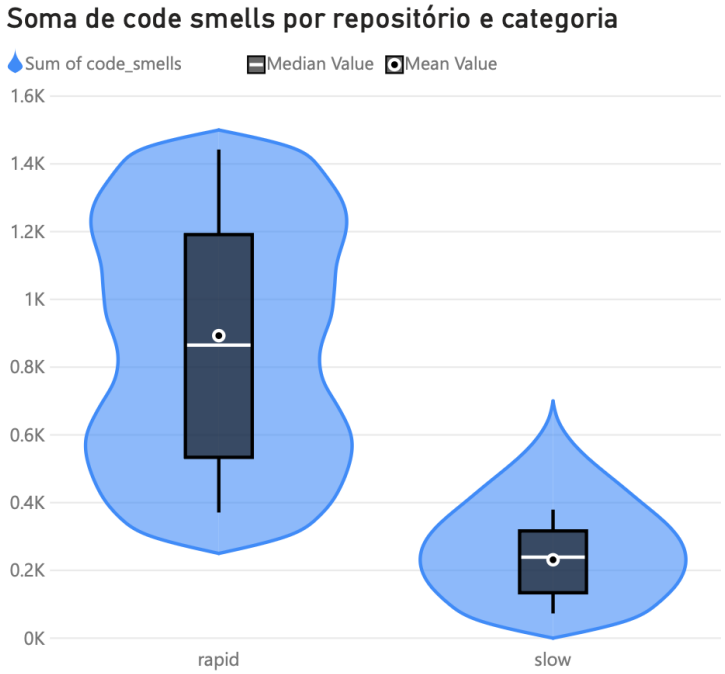


Figura 6: Soma de code smells por repositório e categoria

A análise das métricas de confiabilidade e erros, apresentadas nas Figuras 4, 5 e 6, revela padrões interessantes que desafiam as percepções comuns sobre releases rápidos.

A distribuição de classificações de confiabilidade (reliability_rating) mostra que rapid releases possuem notas mais baixas em questão de confiabilidade, possuindo em sua maioria, notas C, enquanto slow possuem em sua totalidade dos repositórios notas A e B. Indicando um nivel de confiabilidade superior em slow releases.

Quanto ao número de bugs identificados pela análise estática, observamos que repositórios com releases lentos apresentam valores mais reduzidos em comparação a repositórios com releases rapidas, podemos identificar uma mediana mais elevada em repositorios que utilizam releases rápidas, observando o boxplot das releases lentas, podemos identificar numeros reduzidos de bugs encontrados.

A análise de code smells reforça esse padrão. Code smells são indicadores de problemas de design e manutenibilidade que, embora não sejam bugs funcionais, podem levar a defeitos futuros. Os dados mostram que tanto a mediana quanto a média de code smells são superiores em repositorios que se baseiam em rapid releases, o que mostra um código que tende a possuir/gerar mais bugs futuramente, este indicativo mostra outra sensibilidade das releases rápidas em repositórios.

Conclusão RQ02: Os erros são mais comuns em releases de RRC. Há

evidências de que releases rápidas acumulam mais bugs e code smells. Isso sugere que o lançamento de releases em periodos curtos pode acabar prejudicando a estrutura do código, impactando diretamente na qualidade do produto, assim, demenandando um esforço maior para se entregar releases de qualidade.

4.3 RQ 03: O retrabalho é maior em sistemas que utilizam RRC?

Nesta questão final, investigamos se a adoção de releases rápidos impacta a dívida técnica e o retrabalho necessário nos projetos. A análise das métricas de dívida técnica e duplicação de código permite avaliar a sustentabilidade a longo prazo das diferentes abordagens.

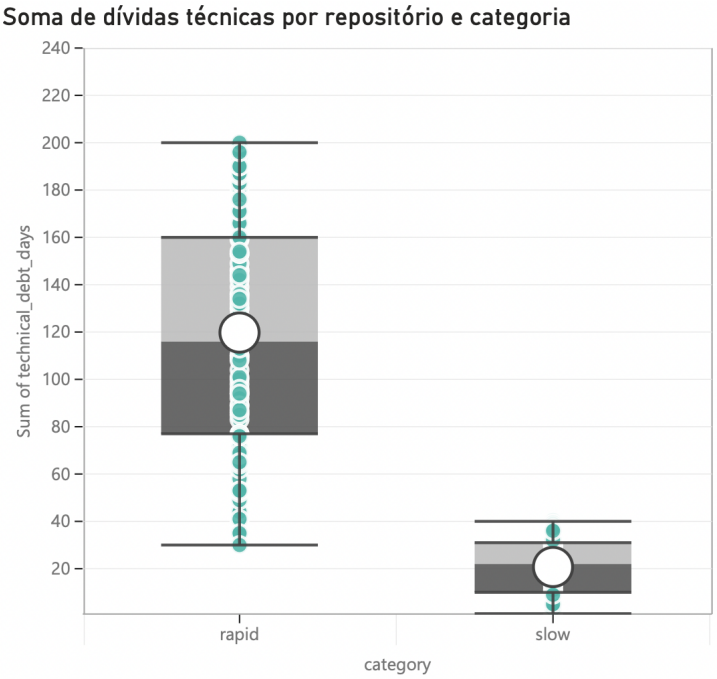


Figura 7: Soma de dívida técnica por repositório e categoria

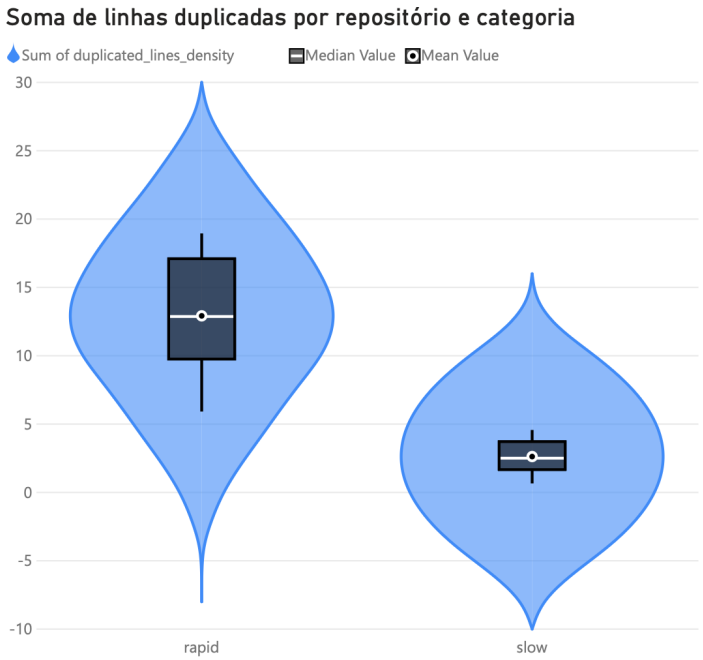


Figura 8: Soma de linhas duplicadas por repositório e categoria

Os resultados apresentados nas Figuras 7 e 8 fornecem insights contraintuitivos sobre o impacto da frequência de releases no retrabalho e dívida técnica.

A métrica de dívida técnica, medida em dias de trabalho necessários para resolver todos os problemas de manutenibilidade identificados, mostra uma diferença significativa entre os dois grupos. Repositórios com releases rapidas apresentam uma mediana de dívida técnica substancialmente maior quando comparados aos repositórios com releases lentas. Isso representa aproximadamente o dobro de dívida técnica acumulada em projetos com releases rapidas.

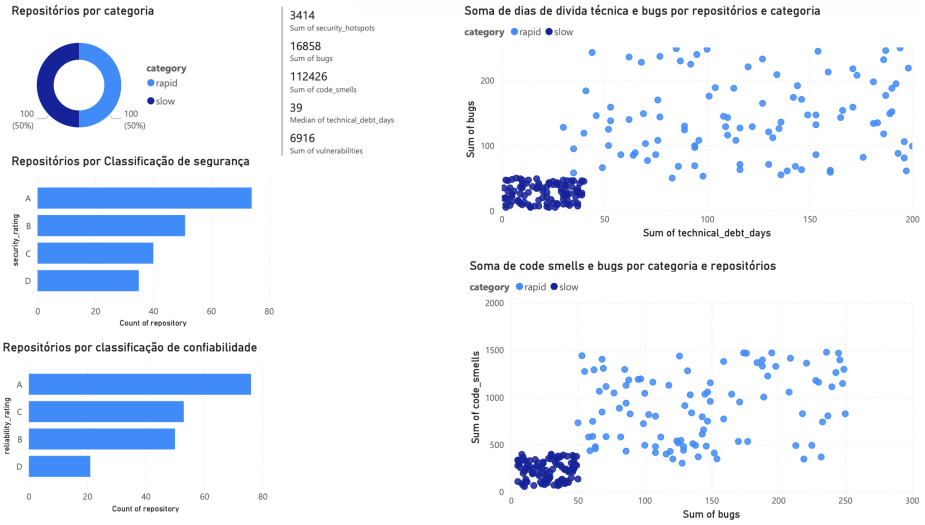
A densidade de linhas duplicadas reforça esse padrão de forma ainda mais clara. Repositórios com releases rapidas apresentam mediana de duplicação de código significativamente superior (5-8%) comparada aos repositórios com releases lentas (2-3%). A duplicação de código é um indicador direto de retrabalho, pois mudanças precisam ser replicadas em múltiplos locais, aumentando a probabilidade de erros e o esforço de manutenção.

Esses resultados sugerem que releases rapidas permitem o acúmulo de dívida técnica ao longo do tempo. A falta de pressão por entregas frequentes pode incentivar práticas de refatoração contínua e revisão constante do código, impedindo que problemas de qualidade se acumulem. Além disso, ciclos lentos facilitam a identificação de duplicações e problemas de design, permitindo correções antes que se espalhem pelo código-base.

Conclusão RQ03: O retrabalho é significativamente maior em sistemas que utilizam Rapid Releases, afirmando a hipótese inicial. Repositórios com releases

rapidas acumulam aproximadamente o dobro de dívida técnica e apresentam densidades de duplicação de código substancialmente maiores.

5 Dashboard



6 Conclusão

Este estudo teve como objetivo avaliar os efeitos das releases rápidas (RRC) na qualidade de sistemas open-source através de uma análise comparativa com sistemas que utilizam releases lentos. Utilizando um dataset balanceado de 200 repositórios do GitHub (100 com releases rápidos e 100 com releases lentos), foram investigadas três dimensões fundamentais da qualidade de software: segurança, confiabilidade e manutenibilidade.

Os resultados obtidos confirmam as hipóteses iniciais de que releases rápidos apresentam desafios significativos para a manutenção da qualidade do código. A análise das três questões de pesquisa revelou que:

- **Segurança (RQ01):** Releases rápidos comprometem a segurança dos sistemas. Os dados mostram que repositórios com RRC apresentam classificações de segurança inferiores (incluindo classificações C e D ausentes em releases lentos), maior número médio de vulnerabilidades e mais pontos críticos de segurança que requerem revisão manual.
- **Confiabilidade (RQ02):** Erros são significativamente mais comuns em RRC. As evidências demonstram que releases rápidos acumulam mais bugs e code smells, com classificações de confiabilidade predominantemente C, enquanto releases lentos mantêm classificações A e B. Isso indica que o tempo reduzido entre releases prejudica a estrutura e qualidade do código.

- **Manutenibilidade (RQ03):** O retrabalho é substancialmente maior em RRC. Projetos com releases rápidos apresentam aproximadamente o dobro de dívida técnica e densidade de código duplicado significativamente superior (5-8%) comparada a releases lentos (2-3%), comprometendo a sustentabilidade a longo prazo.

As análises foram realizadas utilizando técnicas de visualização de dados através de ferramentas de Business Intelligence, com medidas de tendência central apropriadas que consideraram a presença de outliers nos dados. A distribuição balanceada do dataset garantiu comparações estatisticamente robustas entre os grupos.

Esses achados reforçam a percepção de que velocidade e qualidade representam um trade-off real no desenvolvimento de software. Os resultados indicam que a pressão por entregas frequentes em RRC pode comprometer aspectos críticos da qualidade do código. Possíveis explicações incluem: tempo insuficiente para revisões de código adequadas, menor tempo dedicado ao planejamento e refinamento arquitetural, pressão que leva a soluções rápidas em detrimento de soluções robustas, e acúmulo progressivo de débitos técnicos que não são endereçados entre releases.

Para a comunidade de desenvolvimento open-source, gestores de projetos e desenvolvedores que utilizam metodologias ágeis, estes resultados têm implicações práticas importantes. A adoção de RRC requer atenção especial a processos de qualidade para mitigar os riscos identificados. É fundamental estabelecer práticas rigorosas de revisão de código, investir em automação de testes de segurança, alocar tempo específico para refatoração e pagamento de dívida técnica, e considerar ciclos de release mais moderados quando a qualidade é prioritária.

Para desenvolvedores focados em segurança, os resultados são particularmente relevantes, demonstrando que releases frequentes podem introduzir vulnerabilidades e pontos críticos de segurança que demandam atenção redobrada. A escolha entre velocidade de entrega e qualidade de código deve ser consciente e baseada nas prioridades e contexto específico de cada projeto.

Como limitação, é importante reconhecer que esses resultados se aplicam ao contexto de projetos open-source populares do GitHub. Contextos diferentes podem apresentar resultados distintos. Trabalhos futuros poderiam investigar práticas específicas que permitam mitigar os impactos negativos de RRC na qualidade, examinar projetos que conseguiram manter qualidade com releases frequentes para identificar fatores de sucesso, ou analisar a evolução da qualidade ao longo do tempo em projetos que transitam entre diferentes estratégias de release.