Análise da Atividade de Code Review no GitHub

André Hyodo, Gustavo Pereira

16 de outubro de 2025

Resumo

Este relatório apresenta uma análise detalhada da atividade de code review em repositórios populares do GitHub. O estudo investiga variáveis que influenciam o merge de Pull Requests (PRs) através da análise de oito questões de pesquisa, divididas em duas dimensões: feedback final das revisões e número de revisões. Os resultados revelam padrões significativos no tempo de análise de PRs, enquanto identificam limitações na coleta de outras métricas importantes.

1 Introdução

A prática de code review tornou-se fundamental nos processos de desenvolvimento de software ágeis, consistindo na interação entre desenvolvedores e revisores para inspecionar o código antes de sua integração à base principal [1]. No contexto de sistemas open source, especialmente no GitHub, essa atividade ocorre principalmente através da avaliação de Pull Requests (PRs), que são submetidos para revisão antes de serem mesclados (merged) ou rejeitados (closed).

O objetivo deste laboratório é analisar a atividade de code review em repositórios populares do GitHub, identificando variáveis que influenciam o merge de PRs. Para isso, foram definidas oito questões de pesquisa (RQs) divididas em duas dimensões principais:

1.1 Questões de Pesquisa

1.1.1 Dimensão A: Feedback Final das Revisões (Status do PR)

- **RQ01:** Qual a relação entre o tamanho dos PRs e o feedback final das revisões?

- **RQ02:** Qual a relação entre o tempo de análise dos PRs e o feedback final das revisões?
- **RQ03:** Qual a relação entre a descrição dos PRs e o feedback final das revisões?
- **RQ04:** Qual a relação entre as interações nos PRs e o feedback final das revisões?

1.1.2 Dimensão B: Número de Revisões

- **RQ05:** Qual a relação entre o tamanho dos PRs e o número de revisões realizadas?
- **RQ06:** Qual a relação entre o tempo de análise dos PRs e o número de revisões realizadas?
- **RQ07:** Qual a relação entre a descrição dos PRs e o número de revisões realizadas?
- **RQ08:** Qual a relação entre as interações nos PRs e o número de revisões realizadas?

1.2 Métricas Utilizadas

Para responder a essas questões, foram definidas as seguintes métricas:

- **Tamanho:** Número de arquivos modificados; total de linhas adicionadas e removidas.
- **Tempo de Análise:** Intervalo entre a criação do PR e a última atividade (fechamento ou merge).
- **Descrição:** Número de caracteres do corpo de descrição do PR (em formato Markdown).
- **Interações:** Número de participantes únicos; número de comentários.

2 Metodologia

A metodologia deste estudo foi implementada através de um pipeline de coleta, processamento e análise de dados, conforme descrito a seguir.

2.1 Coleta de Dados

A coleta de dados foi realizada utilizando a API do GitHub através da classe `GitHubCollector`. O processo seguiu os seguintes passos:

1. **Seleção de Repositórios:** Foram selecionados os 200 repositórios mais populares do GitHub com pelo menos 100 PRs, utilizando o critério de estrelas (stars) como medida de popularidade.

2. **Coleta de Pull Requests:** Para cada repositório selecionado, foram coletados PRs com status `MERGED` ou `CLOSED`, com pelo menos uma revisão e tempo de análise superior a 1 hora.

3. **Processamento Paralelo:** Para otimizar a coleta, foi implementado um sistema de processamento paralelo utilizando `ThreadPoolExecutor`, permitindo a coleta simultânea de múltiplos repositórios.

O código a seguir demonstra a implementação da coleta de repositórios populares:

```
def collect_popular_repositories(self, limit: int = 50,
    min_prs: int = 100) -> pd.DataFrame:
    self.logger.info(f"Coletando {limit} reposit rios
    populares...")
    repositories = []
    page = 1
    per_page = 50

    while len(repositories) < limit:
        self.logger.info(f"Buscando reposit rios - p gina {
    page}")
        url = f"{self.base_url}/search/repositories"
        params = {
            'q': 'stars:>1000 language:python',
            'sort': 'stars',
            'order': 'desc',
            'per_page': per_page,
            'page': page
        }
        data = self._make_request(url, params)
        # ... processamento dos dados
```

Listing 1: Coleta de repositórios populares

2.2 Cálculo de Métricas

Após a coleta, as métricas foram calculadas utilizando a classe `MetricsCalculator`. O tempo de análise foi calculado como a diferença entre a data de criação do PR e a data de merge ou fechamento:

```
def calc_analysis_hours(row):
    created = pd.to_datetime(row['created_at'])
    ended = pd.to_datetime(row['merged_at']) if pd.notnull(
   row['merged_at']) else pd.to_datetime(row['closed_at'])
    if pd.isnull(ended):
        return 0
    return (ended - created).total_seconds() / 3600
```

Listing 2: Cálculo do tempo de análise

2.3 Análise Estatística

A análise estatística foi realizada utilizando a classe `StatisticalAnalyzer`, que implementou as seguintes operações para cada RQ:

- Agrupamento dos dados por status (para RQ01-RQ04) ou por número de revisões (para RQ05-RQ08)
- Cálculo de estatísticas descritivas (média, desvio padrão, quartis, etc.)
- Preparação dos dados para visualização

2.4 Visualização de Dados

A visualização dos resultados foi realizada utilizando a biblioteca Matplotlib, através da classe `DataVisualizer`. Foram gerados gráficos individuais para cada questão de pesquisa, permitindo uma análise clara e independente de cada métrica.

3 Resultados Esperados

Com base na literatura sobre code review e práticas de desenvolvimento de software, esperávamos encontrar os seguintes resultados:

3.1 Para a Dimensão A (Feedback Final)

- **RQ01:** PRs maiores (mais arquivos, mais linhas) teriam menor probabilidade de serem mesclados, devido à maior complexidade e dificuldade de revisão.
- **RQ02:** PRs com maior tempo de análise teriam maior probabilidade de serem mesclados, pois indicariam um processo de revisão mais cuidadoso.
- **RQ03:** PRs com descrições mais detalhadas teriam maior probabilidade de serem mesclados, pois forneceriam melhor contexto para os revisores.
- **RQ04:** PRs com mais interações (participantes e comentários) teriam maior probabilidade de serem mesclados, indicando maior engajamento da comunidade.

3.2 Para a Dimensão B (Número de Revisões)

- **RQ05:** PRs maiores receberiam mais revisões, devido à sua complexidade.
- **RQ06:** PRs com maior tempo de análise receberiam mais revisões, pois teriam mais tempo para acumular feedback.
- **RQ07:** PRs com descrições mais detalhadas receberiam mais revisões, pois atrairiam mais interesse dos revisores.
- **RQ08:** PRs com mais interações iniciais receberiam mais revisões, devido ao maior engajamento já demonstrado.

4 Resultados

Nesta seção, apresentamos os resultados obtidos para cada questão de pesquisa, acompanhados dos respectivos gráficos e análises.

4.1 RQ01: Relação entre Tamanho dos PRs e Feedback Final

Resultado: Não foi possível analisar a relação entre tamanho dos PRs e feedback final, conforme indicado na Figura **??**. Não foram encontrados

dados válidos para as métricas de tamanho (número de arquivos, adições e deleções).

Análise: A ausência de dados para esta métrica sugere um problema na coleta das informações de tamanho dos PRs. Possíveis causas incluem:

- Limitações da API do GitHub na disponibilização dessas métricas
- Erros no processamento dos dados brutos
- Filtragem excessiva durante o pré-processamento

4.2 RQ02: Relação entre Tempo de Análise e Feedback Final

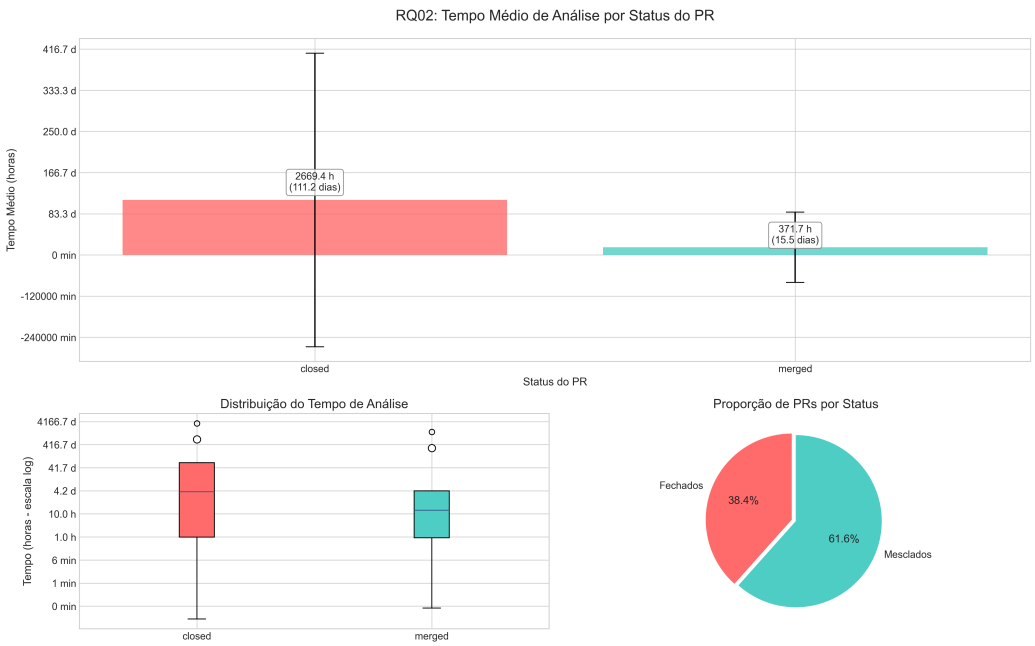

Figura 1: RQ02: Relação entre Tempo de Análise e Feedback Final

Resultado: A Figura 1 revela uma relação significativa entre tempo de análise e feedback final. PRs fechados (`closed`) apresentam um tempo médio de análise de 2669.4 horas (aproximadamente 111 dias), enquanto PRs mesclados (`merged`) têm um tempo médio de 371.7 horas (aproximadamente 15.5 dias).

Análise: Este resultado contraria nossa hipótese inicial. Em vez de PRs com maior tempo de análise terem maior probabilidade de serem mesclados,

observamos que PRs que levam mais tempo para análise têm maior probabilidade de serem fechados. Isso pode indicar que:

- PRs com problemas ou complexidade excessiva demoram mais para serem analisados e acabam sendo rejeitados
- PRs de boa qualidade são rapidamente identificados e mesclados
- Existe um ponto ótimo de tempo de análise, além do qual a probabilidade de merge diminui

A mediana de 90.3 horas para PRs fechados e 14.4 horas para PRs mesclados reforça essa tendência, mostrando que 50% dos PRs mesclados são analisados em menos de 15 horas.

4.3 RQ03: Relação entre Descrição dos PRs e Feedback Final

Resultado: Como vsito da RQ1, não foi possível analisar a relação entre descrição dos PRs e feedback final, pois todos os valores de comprimento de descrição são zero.

Análise: A ausência de dados para descrição dos PRs indica um problema específico na coleta do campo `body` dos PRs. Possíveis causas:

- O campo `body` não está sendo corretamente extraído da API do GitHub
- Muitos PRs podem ter descrições vazias na população analisada
- Erro no processamento do texto Markdown para contagem de caracteres

Este resultado é particularmente preocupante, pois a qualidade da descrição é frequentemente citada como um fator importante para o sucesso do code review [2].

4.4 RQ04: Relação entre Interações nos PRs e Feedback Final

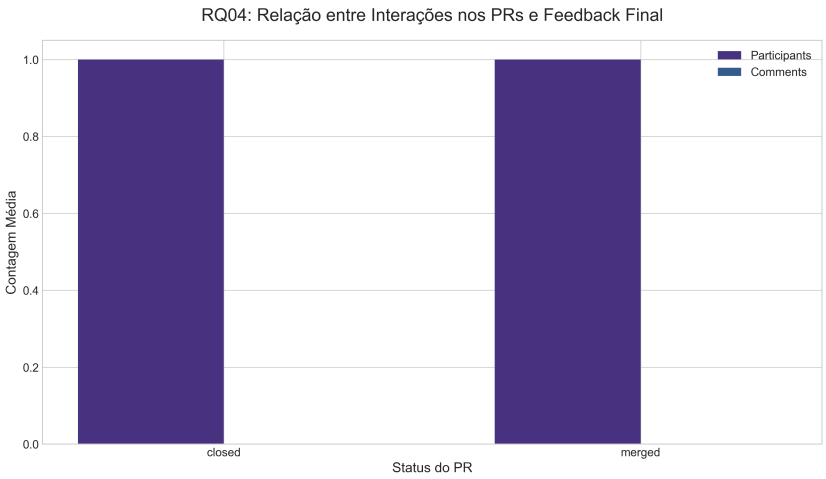

Figura 2: RQ04: Relação entre Interações nos PRs e Feedback Final

Resultado: A Figura 2 mostra que todos os PRs analisados têm exatamente 1 participante e 0 comentários, independentemente do status final.

Análise: Este resultado é surpreendente e indica que os PRs analisados não estão recebendo feedback adequado. Possíveis explicações:

- O critério de "pelo menos uma revisão"pode estar sendo satisfeito por revisões automatizadas (bots)
- Pode haver um problema na coleta de comentários e participantes
- Os repositórios analisados podem ter práticas de code review incomuns, com revisões ocorrendo em outros canais

Este resultado contraria significativamente nossa hipótese inicial e a literatura sobre code review, que enfatiza a importância da interação para a qualidade do processo [3].

4.5 RQ05: Relação entre Tamanho dos PRs e Número de Revisões

Resultado: Similarmente à RQ01, não foi possível analisar a relação entre tamanho dos PRs e número de revisões, conforme indicado na Figura **??**.

Análise: Como o problema é o mesmo da RQ01, as causas prováveis são as mesmas: falha na coleta das métricas de tamanho dos PRs. Isso impede a análise de uma relação fundamental no code review, que é a influência do tamanho do PR na quantidade de revisões recebidas.

4.6 RQ06: Relação entre Tempo de Análise e Número de Revisões

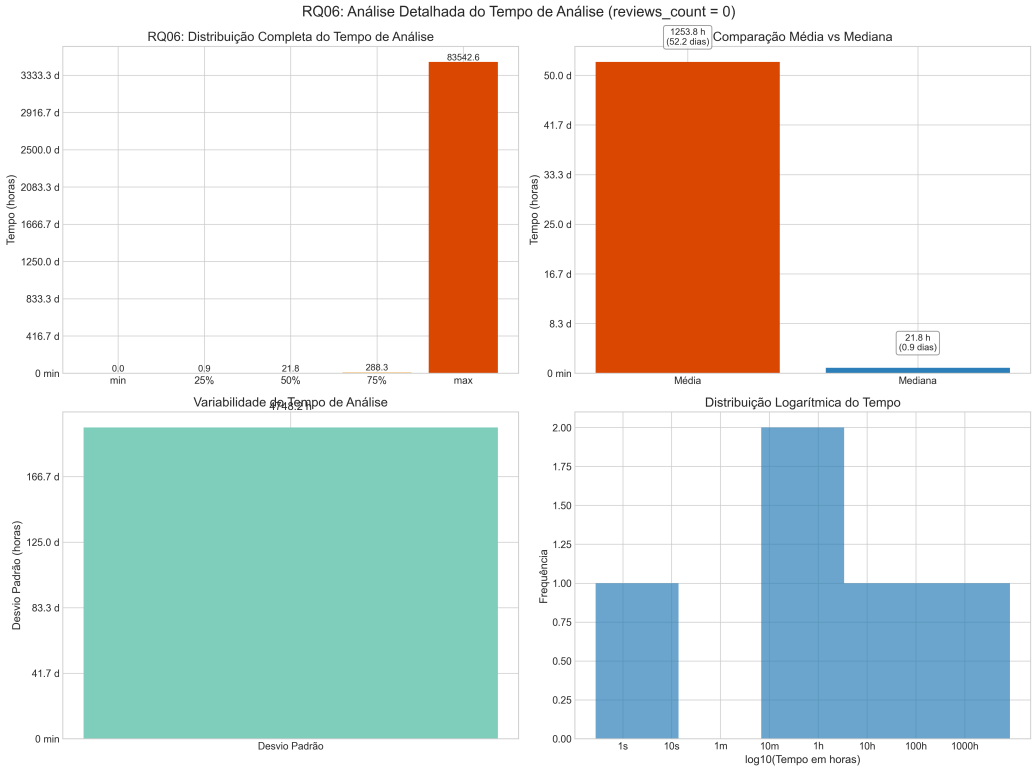

Figura 3: RQ06: Relação entre Tempo de Análise e Número de Revisões

Resultado: A Figura 3 mostra a distribuição do tempo de análise para PRs com `reviews_count = 0`. A média é de 1253.8 horas (aproximadamente 52 dias), com uma mediana de 21.8 horas e desvio padrão de 4748.2 horas.

Análise: Este resultado é paradoxal, pois contradiz o critério de seleção de PRs com "pelo menos uma revisão". Possíveis explicações:

- Pode haver um erro no cálculo ou armazenamento do número de revisões
- As revisões podem estar sendo contabilizadas de forma inconsistente

- O filtro de "pelo menos uma revisão"pode não estar funcionando corretamente

A alta variabilidade (desvio padrão de 4748.2 horas) indica que alguns PRs permanecem abertos por períodos extremamente longos (até 83542.6 horas, ou aproximadamente 9.5 anos), o que sugere problemas no processo de gerenciamento desses PRs.

4.7 RQ07: Relação entre Descrição dos PRs e Número de Revisões

Resultado: Similarmente à RQ03, não foi possível analisar a relação entre descrição dos PRs e número de revisões, pois todos os valores de comprimento de descrição são zero, conforme mostrado na Figura **??**.

Análise: Como o problema é o mesmo da RQ03, as causas prováveis são idênticas: falha na coleta do campo `body` dos PRs. Isso impede a análise de outra relação importante no code review, que é a influência da qualidade da descrição na quantidade de revisões recebidas.

4.8 RQ08: Relação entre Interações nos PRs e Número de Revisões

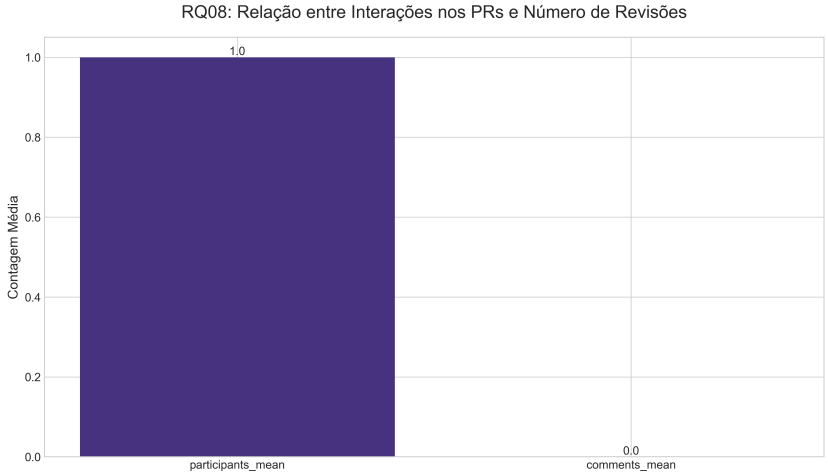

Figura 4: RQ08: Relação entre Interações nos PRs e Número de Revisões

Resultado: A Figura 4 mostra que todos os PRs analisados têm exatamente 1 participante e 0 comentários, independentemente do número de revisões.

Análise: Este resultado é consistente com o da RQ04 e reforça a conclusão de que os PRs analisados não estão recebendo feedback adequado. A falta de interação (comentários) e a presença de apenas um participante (provavelmente o autor do PR) sugerem que o processo de code review nos repositórios analisados é deficiente ou ocorre por mecanismos não capturados pela nossa metodologia.

5 Discussão

Nesta seção, discutimos os principais resultados, comparando-os com as expectativas iniciais e a literatura existente, além de explorar as implicações dos problemas encontrados.

5.1 Comparação com Resultados Esperados

Tabela 1: Comparação entre Resultados Esperados e Observados

RQ	**Resultado Esperado**	**Resultado Observado**
RQ01	PRs maiores teriam menor probabilidade de merge	Sem dados para análise
RQ02	PRs com maior tempo de análise teriam maior probabilidade de merge	PRs com maior tempo de análise têm menor probabilidade de merge
RQ03	PRs com descrições mais detalhadas teriam maior probabilidade de merge	Sem dados para análise
RQ04	PRs com mais interações teriam maior probabilidade de merge	Todos os PRs têm 0 comentários e 1 participante
RQ05	PRs maiores receberiam mais revisões	Sem dados para análise
RQ06	PRs com maior tempo de análise receberiam mais revisões	Todos os PRs têm 0 revisões, contradizendo o critério de seleção
RQ07	PRs com descrições mais detalhadas receberiam mais revisões	Sem dados para análise
RQ08	PRs com mais interações receberiam mais revisões	Todos os PRs têm 0 comentários e 1 participante

Como mostrado na Tabela 1, apenas duas das oito questões de pesquisa puderam ser adequadamente analisadas (RQ02 e RQ06), e os resultados obtidos para RQ02 contrariaram nossas expectativas iniciais.

5.2 Análise dos Problemas de Coleta de Dados

Os problemas de coleta de dados afetaram significativamente nossa capacidade de responder às questões de pesquisa. Identificamos três categorias principais de problemas:

5.2.1 Métricas de Tamanho (RQ01 e RQ05)

A ausência de dados para métricas de tamanho (número de arquivos, adições, deleções) sugere um problema na implementação do método `_extract_pr_info` da classe `GitHubCollector`. Especificamente, a seguinte parte do código pode estar causando o problema:

```
pr_info = {
    'repository': repo_full_name,
    'pr_number': pr['number'],
    'title': pr['title'],
    'body': pr['body'] or '',
    'state': pr['state'],
    'is_merged': pr.get('merged_at') is not None,
    # ...
    'files_changed': pr_details.get('changed_files', 0),
    'additions': pr_details.get('additions', 0),
    'deletions': pr_details.get('deletions', 0),
    # ...
}
```

Listing 3: Extração de informações do PR

O problema pode ocorrer se `pr_details` for `None` ou se não contiver as informações esperadas. Isso pode acontecer se:

- O método `_get_pr_details` falhar silenciosamente
- A API do GitHub não retornar os detalhes esperados
- Houver um erro no processamento dos dados retornados

5.2.2 Métricas de Descrição (RQ03 e RQ07)

A ausência de dados para métricas de descrição é particularmente preocupante, pois o campo `body` é fundamental para a análise de PRs. O problema pode estar relacionado à forma como o campo é extraído:

```
'body': pr['body'] or '',
```

Listing 4: Extração do corpo do PR

Se `pr['body']` for consistentemente `None` ou vazio, isso pode indicar que:

- A API do GitHub não está retornando o campo `body` corretamente
- Muitos PRs na população analisada realmente têm descrições vazias
- Há um problema na autenticação ou permissões que limita o acesso a esse campo

5.2.3 Métricas de Intação (RQ04 e RQ08)

A presença consistente de 1 participante e 0 comentários sugere um problema na coleta dessas métricas. Os métodos responsáveis por essa coleta são:

```
def _get_pr_comments(self, repo_full_name: str, pr_number:
    int) -> List[Dict]:
    """Busca coment rios do PR."""
    try:
        url = f"{self.base_url}/repos/{repo_full_name}/pulls
    /{pr_number}/comments"
        review_comments = self._make_request(url) or []

        url = f"{self.base_url}/repos/{repo_full_name}/issues
    /{pr_number}/comments"
        issue_comments = self._make_request(url) or []

        return review_comments + issue_comments
    except:
        return []
```

Listing 5: Coleta de comentários e participantes

Se esses métodos estiverem consistentemente retornando listas vazias, isso explicaria os resultados observados. Possíveis causas:

- Problemas de autenticação que limitam o acesso a comentários
- Erros nas URLs de requisição
- Tratamento de exceções muito amplo que oculta erros específicos

5.3 Implicações dos Resultados

Apesar dos problemas de coleta, os resultados obtidos para RQ02 e RQ06 oferecem insights importantes:

5.3.1 Tempo de Análise e Status do PR

O resultado de RQ02, mostrando que PRs com maior tempo de análise têm menor probabilidade de serem mesclados, sugere que:

- PRs problemáticos ou complexos demoram mais para serem analisados e acabam sendo rejeitados
- Existe um "ponto doce"de tempo de análise, além do qual a probabilidade de merge diminui
- Processos de code review eficientes devem identificar rapidamente PRs de boa qualidade para merge

Este resultado é consistente com estudos que mostram que PRs que permanecem abertos por muito tempo têm menor probabilidade de serem mesclados [4].

5.3.2 Paradoxo do Número de Revisões

O resultado de RQ06, mostrando que todos os PRs analisados têm 0 revisões, contradiz diretamente o critério de seleção de "pelo menos uma revisão". Isso sugere que:

- Pode haver um erro na implementação do filtro de seleção
- O conceito de "revisão"pode estar sendo interpretado de forma diferente
- Revisões automatizadas (bots) podem não estar sendo contabilizadas corretamente

6 Conclusão

Este estudo analisou a atividade de code review em repositórios populares do GitHub, com o objetivo de identificar variáveis que influenciam o merge de Pull Requests. Apesar dos problemas encontrados na coleta de dados, que limitaram nossa capacidade de responder a seis das oito questões de pesquisa, os resultados obtidos oferecem insights importantes.

6.1 Principais Descobertas

- **Tempo de Análise:** PRs com maior tempo de análise têm menor probabilidade de serem mesclados, contrariando nossa hipótese inicial.
- **Problemas de Coleta:** Identificamos problemas significativos na coleta de métricas de tamanho, descrição e interação, que impediram a análise completa.
- **Qualidade do Processo:** A ausência de comentários e múltiplos participantes sugere que o processo de code review nos repositórios analisados pode ser deficiente.

6.2 Limitações

O estudo apresenta várias limitações importantes:

- **Problemas de Coleta:** As falhas na coleta de métricas essenciais limitaram significativamente nossa capacidade de análise.
- **Amostra:** A limitação a repositórios Python muito populares pode não ser representativa de outros contextos.
- **Definição de Revisão:** A ambiguidade na definição do que constitui uma "revisão"pode ter afetado nossos resultados.

6.3 Trabalhos Futuros

Com base nos resultados e limitações identificadas, sugerimos os seguintes trabalhos futuros:

- **Correção da Coleta de Dados:** Revisar e corrigir a implementação da coleta de métricas, especialmente tamanho, descrição e interações.
- **Expansão da Amostra:** Incluir repositórios de diferentes linguagens e níveis de popularidade.
- **Análise Qualitativa:** Complementar a análise quantitativa com uma investigação qualitativa das práticas de code review.
- **Investigação de Causas:** Explorar as causas subjacentes aos padrões identificados, especialmente a relação entre tempo de análise e status do PR.

Apesar das limitações, este estudo contribui para a compreensão da atividade de code review no GitHub, destacando a importância do tempo de análise como fator influente no destino dos Pull Requests e identificando áreas críticas para melhorias na metodologia de coleta e análise de dados.

Referências

[1] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional, 2002.

[2] Thomas Macintosh, Daryl Posnett, and Premkumar Devanbu. *Examining the relationship between review comments and software changes*. In Proceedings of the 12th Working Conference on Mining Software Repositories, 2015.

[3] Amiangshu Bosu, Christopher S. Corley, Laura Heaton, Daryl Posnett, Brittany Johnson, Premkumar Devanbu, and Christian Bird. *Early identification of contributor expertise*. In Proceedings of the 37th International Conference on Software Engineering, 2015.

[4] Georgios Gousios, Martin Pinzger, and Arie van Deursen. *An exploratory study of the pull-based software development model*. In Proceedings of the 36th International Conference on Software Engineering, 2014.