Engajamento de Desenvolvedores no Processo de Revisão de Código: um Estudo sobre sua Influência na Legibilidade do Código

João M. L. Santos[1], **Samuel A. C. Baker**[1]

[1] Pontifícia Universidade Católica de Minas Gerais (PUC MG)
30140-002 – Belo Horizonte – MG – Brasil

{jmlsantos,sacbaker}@sga.pucminas.br

Abstract. *Code review depends on developers' participation to identify issues and suggest changes to improve the software quality. In this context, we can verify the software quality by relying on source code metrics, such as code readability. However, the impact of low developer participation on code readability is not deeply studied in the literature. Therefore, aiming to better understand this scenario, we compute the readability of releases from about 900 GitHub projects. Then, we identify about 155.296 pull requests associated with each release to analyze the developer's participation. The results show that there is not a significant impact of developer participation in code readability. However, we observe other characteristics, such as a significant rate of self-approved contributions.*
Keywords: *Code review, participation, code readability, repository mining*

Resumo. *A revisão de código depende da participação dos desenvolvedores, que identificam problemas e sugerem mudanças para melhorar a qualidade do software. Neste contexto, podemos verificar a qualidade do software por meio de métricas, como por exemplo a legibilidade. No entanto, o impacto do baixo engajamento de desenvolvedores na legibilidade do código não é um tema profundamente estudado pela literatura. Portanto, visando compreender melhor este cenário, calcula-se neste trabalho a legibilidade de aproximadamente 900 projetos GitHub. Em seguida, identifica-se 155.296 pull requests associados com as releases, para computar o engajamento dos desenvolvedores. Os resultados mostram que não existe um impacto significativo do engajamento dos desenvolvedores na legibilidade. Entretanto, observa-se outras características, como uma taxa significativa de contribuições autoaprovadas.*
Palavras-chave: *Revisão de código, engajamento, legibilidade, mineração de repositórios*

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

Orientador de conteúdo (TCC I): Cleiton Tavares - cleitontavares@pucminas.br
Orientadora de conteúdo (TCC I): Simone de Assis - simone@pucminas.br
Orientador acadêmico (TCC I): Laerte Xavier - laertexavier@pucminas.br
Orientadora do TCC II: Aline Brito - alinebrito@pucminas.br

Belo Horizonte, 14 de Maio de 2023.

1. Introdução

A revisão de código se trata de uma atividade do processo de desenvolvimento de *software* com o intuito de garantir a qualidade do código, encontrando defeitos, transferindo conhecimento e encorajando os desenvolvedores a aderirem a padrões de codificação [Ebert et al. 2019]. Além disso, essa prática, quando feita com qualidade, tem efeitos significativos na redução de problemas no código [Pascarella et al. 2019]. Vale ressaltar que a legibilidade, a qual se trata de um julgamento humano visando identificar a facilidade em entender o código [Buse and Weimer 2010], é um atributo de qualidade importante a ser considerado no processo de revisão. Uma vez que um código ilegível pode dificultar a compreensão do que está sendo feito e, assim, gerar futuros problemas para o produto [Scalabrino et al. 2018].

Dentre os fatores que impactam de forma negativa a qualidade do código, evidencia-se o baixo engajamento no processo de revisão de código [Chouchen et al. 2021].[1] Este fator é definido pelo nível de envolvimento dos participantes durante a análise de determinados trechos de código, considerando os índices de autoaprovações, discussões, e tempo de revisão [McIntosh et al. 2014]. Entretanto, investigações sobre o engajamento no processo de revisão e seu impacto na compreensão do código é um fator pouco explorado. Portanto, o problema que este trabalho busca resolver é a **escassez de informações sobre o impacto do engajamento de desenvolvedores durante o processo de revisão na legibilidade do código**.

Algumas pesquisas buscaram entender os fatores que impactam na qualidade, mostrando ser necessário identificar aspectos que possam interferir na compreensão do código [Fakhoury et al. 2019]. Além disso, o processo de revisão de código tem grande participação na detecção e prevenção de *code smells* [Han et al. 2021] e violações de código [Han et al. 2020], evitando falhas e melhorando a compreensão. Portanto, como a legibilidade envolve a facilidade de entendimento, e a revisão consiste no processo de análise do código visando melhorias, é relevante entender as relações entre ambos os conceitos. Visando, através desta relação, discutir a importância da revisão de código como fator de impacto na legibilidade do *software*.

Dessa forma, o objetivo deste trabalho é **avaliar o impacto do engajamento durante o processo de revisão na legibilidade final do código**. Sendo assim, almeja-se alcançar os seguintes objetivos específicos: (i) analisar a evolução da legibilidade em *releases* de projetos populares no GitHub, escritos na linguagem Java; (ii) analisar o engajamento dos contribuidores desses projetos no processo de revisão de código em períodos entre *releases*; (iii) avaliar por meio de métodos estatísticos a correlação entre o engajamento e a legibilidade das *releases* do projeto.

O resultado desta pesquisa é uma análise sobre a evolução do engajamento e da legibilidade ao longo de *releases*, assim como, investigações sobre o impacto de fatores de engajamento. O estudo baseou-se no fato de que o processo de revisão está relacionado com a avaliação de trechos de código, visando decidir se eles podem ser integrados ao código principal [Tufano et al. 2021]. Observou-se que as métricas analisadas possuem uma baixa variação ao longo das *releases*. Além disso, alguns fatores de engajamento são mais frequentes, como por exemplo, contribuições aprovadas sem revisões externas.

[1] Termo original denominado como *participation* [McIntosh et al. 2014].

O restante deste trabalho está organizado da seguinte forma: a Seção 2 aborda a fundamentação teórica, explorando os conceitos utilizados nesta pesquisa. A Seção 3 apresenta os trabalhos relacionados a este estudo. A Seção 4 abrange os materiais e métodos utilizados nesta pesquisa. As Seções 5 e 6 apresentam e discutem os resultados obtidos, respectivamente. A Seção 7 contempla as ameaças à validade, e as formas de mitigação encontradas durante o estudo. Por fim, a Seção 8 apresenta a conclusão e os trabalhos futuros.

2. Fundamentação Teórica

Nesta seção, apresenta-se os principais conceitos utilizados como base para este trabalho. Na Seção 2.1, aborda-se a revisão de código e fatores importantes deste processo. Na Seção 2.2, apresenta-se o funcionamento do processo de revisão na plataforma do GitHub. Na Seção 2.3 são mostrados aspectos sobre a legibilidade de código.

2.1. Revisão de Código

A revisão de código é um processo sistêmico muito difundido dentro de empresas de *software* e em projetos de código aberto [Chouchen et al. 2021]. Sendo realizada através da inspeção de trechos de código modificados, conhecidos como *pull requests*, buscando identificar e prevenir que *code smells* e violações, sejam encorporados ao código-fonte [Cunha et al. 2021]. Esse processo, visa garantir que o código inserido seja legível, e tenha facilidade de compreensão e manutenção [Han et al. 2021]. O processo de revisão de código não deve ser finalizado até que a modificação esteja adequada para integrar o código-fonte, e pode ser divido nas seguintes etapas: (i) solicitação de análise das modificações; (ii) análise e possíveis sugestões de melhorias por parte dos revisores; (iii) execução das correções por parte do desenvolvedor [Ueda et al. 2019].

Durante a etapa de análise do código, são validadas as modificações realizadas e os desenvolvedores envolvidos discutem sobre melhorias e correções a serem implementadas [Tufano et al. 2021]. Nessa etapa, um dos fatores que influenciam negativamente o resultado é o engajamento, podendo refletir diretamente na qualidade final do código [McIntosh et al. 2014]. A partir disso, o engajamento pode ser definido pelo nível de envolvimento dos participantes durante a análise das modificações, considerando-se três aspectos: (i) proporção de autoaprovações; (ii) proporção de revisões rápidas; (iii) proporção de revisões sem nenhuma discussão [McIntosh et al. 2014].

2.2. Revisão de Código Através do GitHub

O GitHub se trata de um serviço em nuvem para hospedagem de repositórios através do sistema de controle de versão Git. Por meio dessa plataforma, os desenvolvedores utilizam o processo de revisão de código para validar a qualidade do que está sendo modificado. O processo, dentro da plataforma, recorre às mesmas etapas vistas acima: envio de um novo *pull request*; avaliação e discussão das modificações; realização das alterações solicitadas. Além disso, a plataforma disponibiliza recursos que ajudam na execução do processo de revisão, tais como: número mínimo de revisores; integração do código somente após a resolução dos comentários; possibilidade de comentar sobre trechos específicos; e outros. Dessa maneira, é possível perceber a importância da plata-

forma para este trabalho, pois além de disponibilizar uma interface[2] de integração prática para a coleta de dados, pode fornecer um grande acervo para o estudo de revisões de código [Kalliamvakou et al. 2014].

2.3. Legibilidade de Código

Legibilidade é uma característica relacionada à manutenibilidade de *software* [Buse and Weimer 2010]. Existe uma norma criada para padronizar a avaliação da qualidade de *software*, chamada de ISO/IEC 9126[3]. Essa norma define alguns atributos externos e internos que podem ser verificados no produto de *software*. Os atributos internos estão relacionados com o processo do desenvolvimento e os externos com particularidades não funcionais do *software*. Em relação aos atributos internos, a manutenibilidade se destaca por permitir que os desenvolvedores consigam modificar facilmente o *software*, incluindo melhorias, extensões e até correções de defeitos.

A legibilidade é um julgamento humano que verifica o quão fácil é o entendimento de um texto [Buse and Weimer 2010]. Assim, o código também precisa ser legível para que os desenvolvedores consigam modificá-lo quando necessário, já que o ato de ler o código é uma das tarefas que consomem mais tempo dos desenvolvedores [Raymond 1991]. O cálculo utilizado neste trabalho para verificar esse atributo foi proposto por Posnett et al. (2011), considerando as características de volume, entropia e quantidade de linhas do código analisado.

3. Trabalhos Relacionados

Esta seção explora e discute artigos relacionados à legibilidade e a revisão de código, temas centrais abordados na pesquisa. Os trabalhos escolhidos contribuíram com ideias, conceitos e métricas para a elaboração deste estudo.

Muitos estudos abordam como tema a revisão de código, seja identificando possíveis melhorias para o processo, ou avaliando os impactos dessa atividade no *software*. Chouchen et al. (2021) investigaram o fenômeno de anti-padrões no processo de revisão de código, visando mostrar quais são as práticas que mais afetam o processo. Os anti-padrões foram divididos em: *confused reviewers*, *divergent reviewers*, *low review participation*, *shallow review* e *toxic review*. A partir disto, realizaram uma avaliação manual sobre 100 revisões coletadas da plataforma Opendev,[4] dos quais buscaram analisar a frequência e a prevalência dos anti-padrões dentro do processo. Por fim, constaram que 67% das revisões analisadas continham ao menos um anti-padrão, sendo que o *low review participation*, o qual também pode ser definido como o engajamento durante a revisão, foi o mais frequente dentro da análise. Sendo assim, este trabalho busca complementar esse estudo, visando entender como o engajamento durante o processo de revisão se relaciona com os aspectos de qualidade, tendo como foco a legibilidade de código.

Uchôa et al. (2020) buscaram demonstrar os impactos do processo de revisão sobre a degradação de *desing* do *software*. Em sua pesquisa, analisaram os *pull requests* de

[2]Disponível em: https://docs.github.com/en/graphql/overview/explorer. Último acesso: 11 de Outubro de 2022.

[3]Disponível em: https://www.iso.org/standard/22749.html. Último acesso: 29 de Outubro de 2022.

[4]Disponível em: https://opendev.org. Último acesso: 29 de Outubro de 2022

sete projetos escritos na linguagem Java. Os dados foram coletados com auxílio da plataforma CROP (do inglês *Code Review Open Platform*), a qual contém grande acervo de dados sobre a revisão de repositórios de código aberto. Após a análise dos dados, constaram que quanto maior o tempo de revisão, maior o risco de serem adicionados sintomas de degradação no *software*. Além disso, seus resultados evidenciaram que revisões mais ativas tendem a diminuir a recorrência de sintomas de degradação. Portanto, visto que os resultados mostram a relevância do engajamento para prevenir sintomas de degradação, este trabalho tem como intuito identificar se essa mesma relação acontece para sintomas que prejudicam a legibilidade do *software*. Entretanto, a versão atual da plataforma CROP inclui apenas 11 projetos. Neste trabalho, realiza-se uma análise em larga escala, envolvendo centenas de projetos.

McIntosh et al. (2014) estudaram a relação da revisão de código e da quantidade de componentes no *software* propensos a ter algum defeito. Os objetos de pesquisa foram os projetos de código aberto: *Qt*, *VTK*, e *ITK*, coletados através da API (do inglês, *Application Program Interface*) fornecida pelo *Gerrit*. Para avaliar a revisão de código, analisaram as métricas de cobertura e de engajamento dessa atividade. Como resultado, foi ressaltado que componentes com maior cobertura tendem a ter menos defeitos após o lançamento de novas versões e que o baixo engajamento no processo de revisão tem um impacto negativo na qualidade do *software*. Desse modo, esta pesquisa visa, através do conceito de engajamento fornecido pelos autores, complementar esse estudo buscando entender os impactos desse fator sobre outros aspectos de qualidade do *software*.

Mannan et al. (2018) avaliaram a evolução da legibilidade em projetos de código aberto. Com esse objetivo, foi realizado uma pesquisa com 49 projetos no GitHub, avaliando 8296 *commits* e 1766 arquivos. Calculou-se a legibilidade repetidas vezes no mesmo projeto a cada semana de código que havia sido gerado desde sua criação. A medição da legibilidade utilizou da fórmula criada por Posnett e os *code smells* foram identificados usando uma ferramenta chamada inFusion. Os projetos de código aberto obtiveram bons resultados no cálculo da legibilidade e a mesma não varia muito ao longo do ciclo de vida. Além disso, os autores fizeram uma reflexão de que alguns processos do desenvolvimento poderiam contribuir para bons resultados de legibilidade, como a própria revisão de código. O presente trabalho busca entender essa relação da revisão de código com a legibilidade. Os cálculos de legibilidade são feitos com a mesma fórmula de Posnett. Este trabalho, utiliza a ideia de analisar os projetos ao longo do tempo, porém, somente com as *releases* disponíveis.

Barbosa et al. (2022) investigaram o impacto do uso do desenvolvimento cognitivo[5] na legibilidade final do código. Para entender isso, os autores propuseram duas abordagens. A primeira foi um questionário onde os desenvolvedores escolhiam o código mais legível e a segunda foi a execução da métrica fornecida por Posnett nos trechos de código para verificar a legibilidade. Em relação ao questionário foi percebido que os profissionais escolheram códigos desenvolvidos usando a técnica cognitiva como mais legíveis em 7 casos dos 10 apresentados. Já em relação à execução das métricas de Posnett, somente em 2 dos 10 casos o desenvolvimento cognitivo teve o resultado melhor. Os resultados apresentaram uma evidência da melhora que o desenvolvimento cognitivo causa na legibilidade do código. O presente trabalho busca usar a mesmo cálculo de legi-

[5]Termo original denominado como *Cognitive-Driven Development*.

bilidade para verificar a relação do engajamento na revisão de código com essa métrica.

4. Materiais e Métodos

A pesquisa proposta neste trabalho pode ser classificada como exploratória e quantitativa. Isso se deve ao fato deste estudo buscar, por meio da medição e da análise de um conjunto de dados numéricos, compreender e aprimorar os conhecimentos atuais referentes ao processo de revisão e aos aspectos de qualidade, especificamente a legibilidade do código.

A partir disso, utiliza-se de servidores locais e remotos para o processamento dos dados e métricas. Após a seleção dos repositórios, a API do GitHub baseada em *GraphQL* é utilizada para coleta dos dados. A escolha desse modelo de API se deve à facilidade de acesso e criação de *queries*, por meio da interface de exploração citada na Seção 2.2, além da quantidade de repositórios disponíveis para coleta dos dados. Ademais, a escolha da linguagem tem como base a relevância da mesma, sendo Java uma das linguagens de programação mais utilizadas dentro da plataforma analisada [GitHub 2021]. Adicionalmente, a Linguagem Java também foi utilizada por Mannan et al. (2018), em um estudo que se concentra no cálculo das métricas de legibilidade

4.1. Métricas

O cálculo do engajamento dos desenvolvedores baseia-se em parâmetros definidos por McIntosh et al. (2014). Os atributos considerados são: contribuições aprovadas pelo próprio autor (CA), isto é, sem revisões externas; tempo de aprovação do *pull request* inferior a 1 hora para cada 200 *churns*, isto é, a soma das linhas adicionadas e removidas (TA); e *pull requests* aprovados sem comentários (PR). Especificamente, os autores discutem que estas métricas indicam um baixo engajamento nos projetos, visto que, contribuições aprovadas pelo próprio autor, provavelmente não foram revisadas; revisões rápidas sugerem uma baixa qualidade no processo; e que existe uma propensão a defeitos, em integrações de código realizadas sem discussões.

Neste trabalho, o valor do engajamento é definido entre 0 e 3, considerando a ocorrência das métricas mencionadas anteriormente. Dessa forma, 0 refere-se a ocorrência de todos os fatores que indicam baixo engajamento, e 3 indica a ausência deles, isto é, um bom engajamento. As métricas também são discutidas individualmente. Vale ressaltar que existe diferença do cálculo proposto por McIntosh et al. (2014) referente ao tempo de aprovação. Os autores utilizaram 200 linhas por hora e este trabalho usa 200 *churns* por hora, pois se trata de um atributo mais relacionado as mudanças realizadas no código. A Equação 1, descrita abaixo, representa a fórmula utilizada para discutir o engajamento dos projetos.

$$engajamento = 3 - ((CA) + (TA) + (PR)) \tag{1}$$

A legibilidade é quantificada utilizando a fórmula de Posnett et al. (2011). Os atributos avaliados são: entropia de Shannon (H), isto é, complexidade, ou grau de desordem, ou quantidade de informação em um sinal, ou dados definidos [Posnett et al. 2011]; número total de linhas no programa (L); volume obtido nas métricas de Halstead (V), isto é, número de bits necessários para representar todos os operadores e operandos multiplicados pelo número total de *tokens* [Posnett et al. 2011]. De maneira semelhante a Mannan

et al. (2018), o presente estudo obtém os valores desses atributos individualmente e depois os aplica na Equação 2, representada pela letra z. Com o resultado de z, é possível calcular a legibilidade de um método utilizando a Equação 3 (Sigmoide), com a função de normalizar o valor final entre 0 e 1.

$$z = 8,88 - 0,033(V) + 0,40(L) - 1,5(H) \tag{2}$$

$$legibilidade = \frac{1}{1 + e^{-z}} \tag{3}$$

Para avaliar o impacto do engajamento dos desenvolvedores na legibilidade do código, este trabalho verifica a variação da métrica de legibilidade em relação à variação da métrica de engajamento. Em cada projeto minerado no GitHub é realizado o cálculo da legibilidade das 10 últimas *releases* e o cálculo do engajamento dos períodos entre o lançamento dessas *releases*. Após isso, é possível calcular o coeficiente de correlação de Spearman para entender se essas duas medidas são diretamente correlacionadas.

4.2. Coleta de Dados

Conforme apresentado na Figura 1, a etapa de coleta dos dados é realizada por meio da utilização de ferramentas de mineração de repositórios e de *scripts* baseados na linguagem Python. O objetivo é realizar a busca dos dados para condução da pesquisa, assim como computar as métricas. Existem quatro grandes etapas: (i) seleção e coleta de repositórios; (ii) coleta de *pull requests*; (iii) coleta de *releases*; (iv) cálculo das métricas, isto é, cálculo do engajamento e legibilidade. As próximas subseções detalham cada etapa.

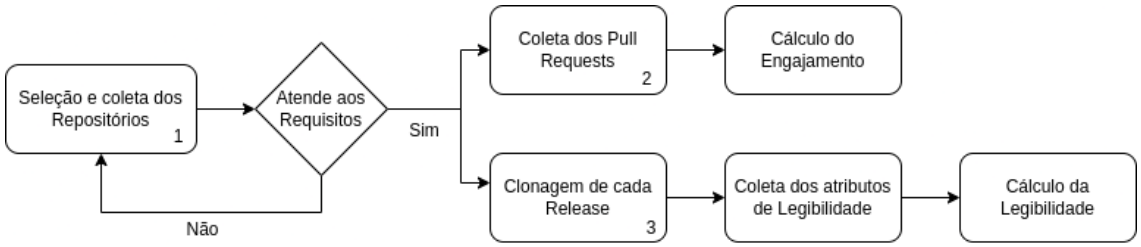


Figura 1. Processo utilizado para coleta dos dados necessários para análise

4.2.1. Mineração de Repositórios

Inicialmente, realiza-se a seleção de 1.000 repositórios populares, disponíveis na plataforma GitHub. O estudo concentra-se na linguagem de programação Java, devido a sua popularidade [GitHub 2021]. Os projetos são selecionados em ordem de maior popularidade através do número de estrelas, visto que é uma métrica relevante para mostrar a popularidade de repositórios [Silva and Valente 2018]. Entretanto, existem repositórios populares no GitHub que não são necessariamente sistemas de *software*, como por exemplo, repositórios para demonstrar exemplos de código ou documentações. Dessa forma, outros critérios são adotados, visando identificar projetos de interesse [Neto et al. 2021, Martins 2022]. Especificamente, além do número de estrelas, este estudo inclui também os seguintes critérios para seleção dos repositórios:

- **C1**: não ser *fork* de outro projeto, para evitar a análise de cópias de repositórios [Brito et al. 2021, Martins 2022];
- **C2**: possuir pelo menos 1.000 *commits*, para evitar projetos com baixa atividade [Brito et al. 2021, Hora et al. 2018];
- **C3**: possuir pelo menos 100 *pull requests*, com o intuito de evitar projetos com uma baixa taxa de contribuições;
- **C4**: possuir pelo menos 10 *releases* lançadas, para realização da análise evolucionária [Grijó and Hora 2018];

Após a definição dos requisitos necessários para a seleção dos repositórios, realiza-se a filtragem e a busca dos repositórios através da ferramenta *GitHub Search* (GHS) [Dabic et al. 2021].

4.2.2. Coleta de *Releases* e *Pull Requests*

A segunda etapa é realizada por meio de um *script* para coletar as *releases* dos projetos GitHub. Ordena-se as *releases* de forma decrescente pela data de criação, coletando os seguintes atributos: *tag* (usado como identificador) e *createdAt* (atributo do tipo data que faz referência à criação da *release*). Os dados coletados são armazenados em um arquivo CSV (do inglês *Comma Separated Values*). Em seguida, busca-se as últimas 10 *releases* de cada projeto, considerando a data de criação da *release*.

Após a identificação das *releases*, um segundo *script* busca todos os *pull requests* já mesclados ao código, criados no período entre *releases*. As solicitações criadas por robôs são desconsideradas. Nesta etapa são coletados atributos relacionados a cada *pull request*, tais como: (i) *createdAt* e *mergedAt*, atributos relacionados à criação de uma solicitação e a sua integração ao código, respectivamente; (ii) *comments*, atributo referente à quantidade de comentários; (iii) *additions* e *deletions*, atributos referentes às modificações realizadas; e (iv) *reviews*, atributo relacionado aos revisores de um *pull request*. Estes atributos são necessários para a realização do cálculo do engajamento descrito na Seção 4.1. Os valores coletados nessa etapa são armazenados em um arquivo CSV. Após a coleta das *releases* foram removidos os projetos que tinham 10 *releases*, mas que não respeitavam o padrão semântico.[6]

4.2.3. Cálculo da Legibilidade e do Engajamento

Utilizando os dados coletados, realiza-se o cálculo do engajamento relativo a cada *pull request*. Entretanto, estes valores não são suficientes para a análise da legibilidade de um projeto. Dessa forma, por meio de um novo *script* é realizado o *checkout* de cada uma das últimas dez *releases* por projeto. Os métodos existentes nas *releases* são extraídos, para assim realizar o cálculo da mediana da legibilidade, considerando o modelo detalhado na Seção 4.1 [Posnett et al. 2011]. Para tanto, utiliza-se a ferramenta *Program Comprehension Metrics*, que foi utilizada em estudos anteriores neste mesmo contexto de análise de legibilidade [Barbosa et al. 2022, Mendonça et al. 2020].[7]

[6]Disponível em: https://semver.org. Último acesso: 27 de Outubro de 2022.

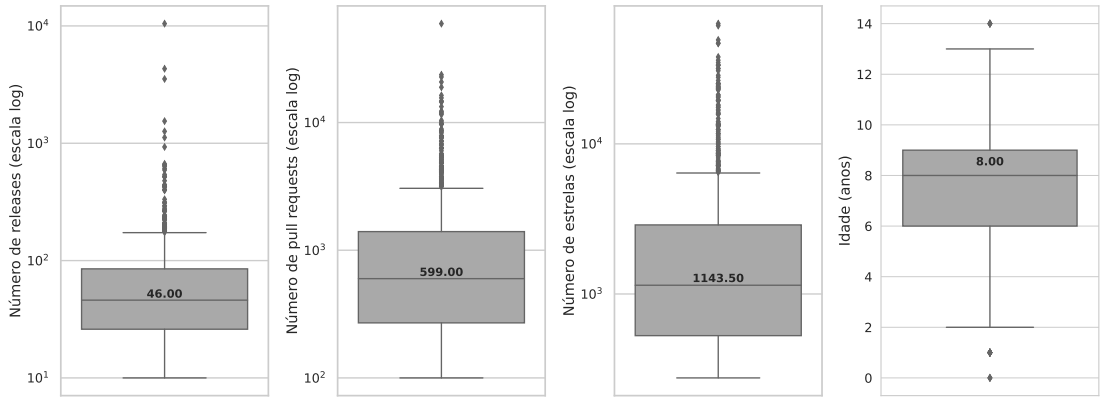
[7]Disponível em: https://github.com/rbonifacio/program-comprehension-metrics. Último acesso: 25 de Fevereiro de 2023.

5. Resultados

Nesta seção, apresenta-se a caracterização do conjunto de dados e os resultados das métricas descritas na Seção 4.1, visando atingir os objetivos propostos.

5.1. Caracterização do Conjunto de Dados

A caracterização dos projetos é realizada por meio de gráficos *boxplot*, conforme apresentado na Figura 2. Especificamente, apresenta-se as características dos repositórios, considerando a quantidade de *releases* publicadas, o total de *pull requests*, o número de estrelas e a idade dos projetos. Removeu-se da análise 141 projetos GitHub que não gerenciam as *releases* respeitando o versionamento semântico.[8] Dessa forma, este trabalho inclui a análise de 859 projetos populares, como, por exemplo, `elastic/elasticsearch`, um sistema popular, que provê um mecanismo distribuído para pesquisa e análise *RESTful*, e `netflix/hystrix`, uma biblioteca para gerenciar pontos de acessos remotos.



(a) *Releases* (b) *Pull requests* (c) Estrelas (d) Idade

Figura 2. Caracterização dos projetos GitHub

Conforme apresentado na Figura 2(a), a quantidade de *releases* por projeto varia entre 10 e 10.459, sendo a mediana igual a 46. Em relação ao número de *pull requests*, os valores variam entre 100 e 59.511, sendo a mediana igual a 599 (Figura 2(b)). Aproximadamente 75% dos projetos possuem até 1.399 *pull requests*. A Figura 2(c) mostra a distribuição do número de estrelas. Os valores variam entre 276 e 63.230, sendo a mediana igual a 1.143,5, o primeiro quartil igual a 527,75 e o terceiro quartil com 2.884,5. Por fim, a Figura 2(d) mostra a distribuição da idade dos projetos, com valores variando entre 0 e 14 anos.

5.2. Evolução da legibilidade das *releases* ao longo do tempo

Nesta primeira questão de pesquisa estuda-se como a legibilidade dos projetos varia ao longo do tempo. Para tanto, analisa-se 10 *releases* de cada projeto, totalizando 8.590 *releases*. A Figura 3 apresenta a distribuição da legibilidade. Os resultados são sumarizados por meio de um *boxplot*, agrupados por *release*. `R` refere-se à *release* mais recente dos projetos, enquanto `R-9` indica a última *release* analisada.

[8] `https://semver.org/lang/pt-BR`

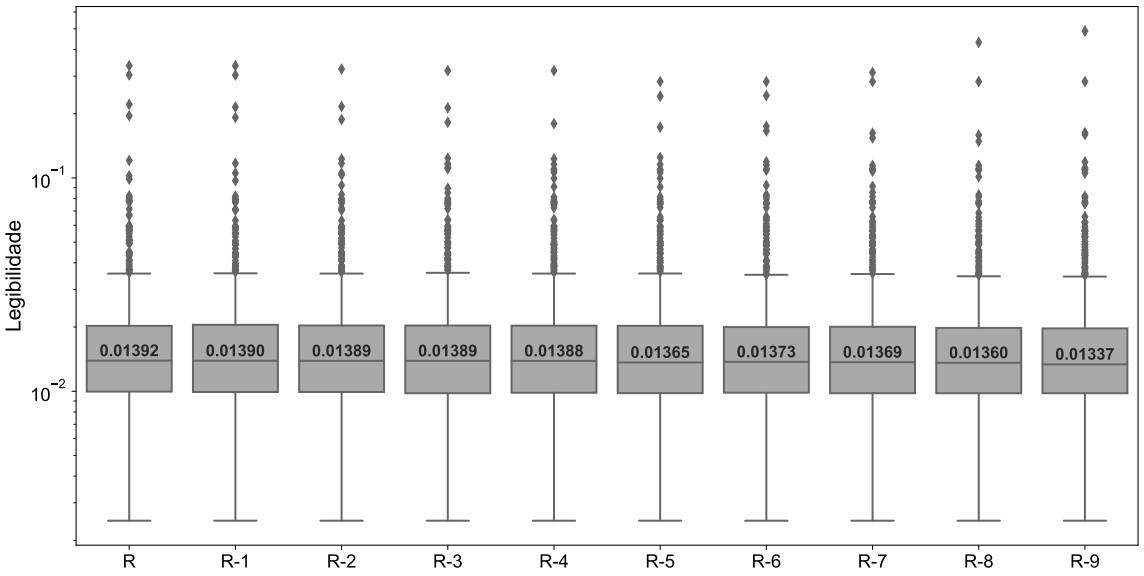


Figura 3. Evolução da legibilidade por *release*

Observa-se que a legibilidade possui uma variação menor que $\sim 0,01$ ao longo das *releases*, reforçando resultados de estudos sobre legibilidade em outros contextos [Mannan et al. 2018, Barbosa et al. 2022]. Lembrando que valores que se distanciam de zero indicam que a legibilidade está piorando. Neste contexto, percebe-se uma baixa variações nos valores, comparando a *release* mais antiga (`R9`) com a mais recente (`R`).

5.3. Evolução do engajamento dos projetos ao longo do tempo

Para responder a segunda questão de pesquisa, calcula-se o engajamento de 155.296 *pull requests* realizados entre as últimas 10 *releases* de cada projeto. No geral, os valores de engajamento estão distribuídas da seguinte forma, entre os *pull requests*: 8.706 *pull requests* (5,60%) com engajamento zero; 26.202(16,87%) com engajamento igual a 1; 55.167 (35,52%) *pull requests* com valor igual a 2; e por fim 65.221(41,99%) possuindo engajamento igual a 3. Agrupa-se os dados por período, coletando a mediana de engajamento dos *pull requests* em cada um. Considerando todos os projetos válidos, analisa-se 7.292 períodos entre *releases*.

A Figura 4 mostra a distribuição do engajamento, desprezando projetos com engajamento nulo em todas as *releases*, isto é, sem *pull requests* aprovados. Considerando os valores de mediana, identifica-se a ocorrência de pelo menos um dos critérios que acusa baixo engajamento. O projeto `alibaba/fastjson` mostra este cenário, onde os valores de engajamento oscilam entre 1 e 2 em todos os períodos.[9] Não se observa uma variação significativa nos valores de engajamento. Entretanto, percebe-se nos três períodos mais recentes, uma menor frequência de valores inferiores a 1, isto é, projetos com o baixo engajamento.

[9]Disponível em: https://github.com/alibaba/fastjson. Último acesso: 15 de Março de 2023.

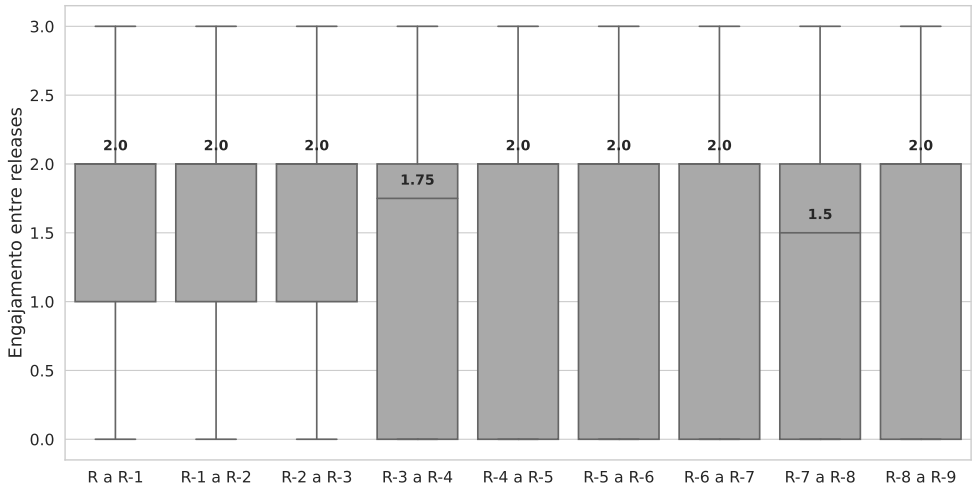


Figura 4. Evolução do engajamento nos períodos entre as *releases*

Dentre os resultados, 356 projetos (41,44%) possuem $engajamento = 0$ em pelo menos uma *release*, como por exemplo, o projeto `JorelAli/CommandAPI`,[10] uma API de comandos usados em um jogo. Neste caso, as revisões são aprovadas pelo próprio autor, sem comentários, e realizadas em um curto período. Em contraste, existem projetos com bom engajamento, como por exemplo, `elastic/elasticsearch`,[11] onde em todas as *releases* analisadas, obteve-se $engajamento = 3$. Nos próximos parágrafos, discuti-se os atributos de engajamento individualmente, evidenciando as características descritas anteriormente. No geral, observa-se que dois atributos são mais frequentes, complementando os resultados da análise ao longo do tempo.

Revisões aprovadas pelo próprio autor. Na Figura 5 apresenta-se a distribuição da proporção de autoaprovações ao longo das *releases* dos projetos coletados. É possível notar que, em todos os períodos, pelo menos um projeto obteve 100% de autoaprovação em todos os *pull requests* das *releases* analisadas.

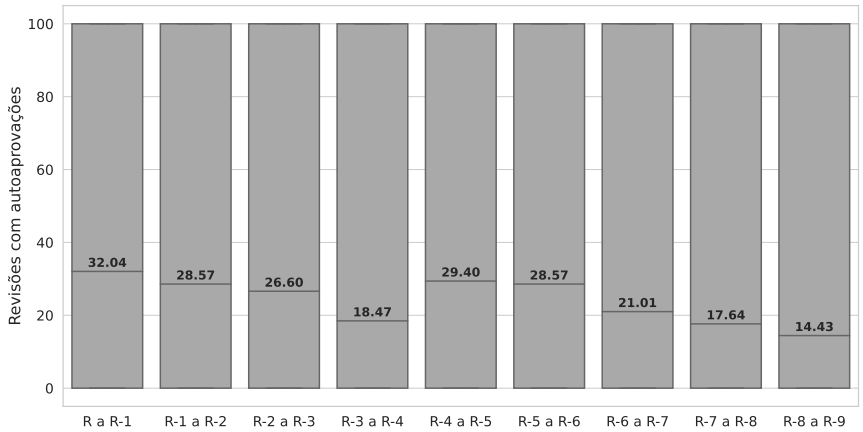


Figura 5. Evolução da proporção de *pull requests* aprovados pelo próprio autor

[10]Disponível em: https://github.com/JorelAli/CommandAPI. Último acesso: 15 de Março de 2023.
[11]Disponível em: https://github.com/elastic/elasticsearch. Último acesso: 15 de Março de 2023.

Revisões aprovadas sem comentários. A Figura 6 mostra a distribuição da proporção de *pull requests* sem comentários ao longo das *releases*. Observa-se que a maioria dos valores está abaixo de 80%, sendo que a mediana dos intervalos varia entre 20% e 40%.

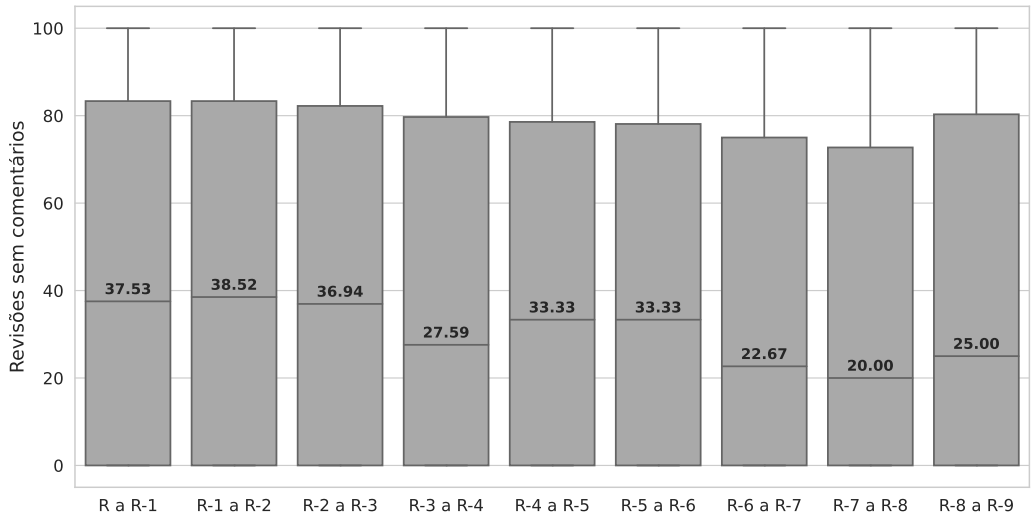


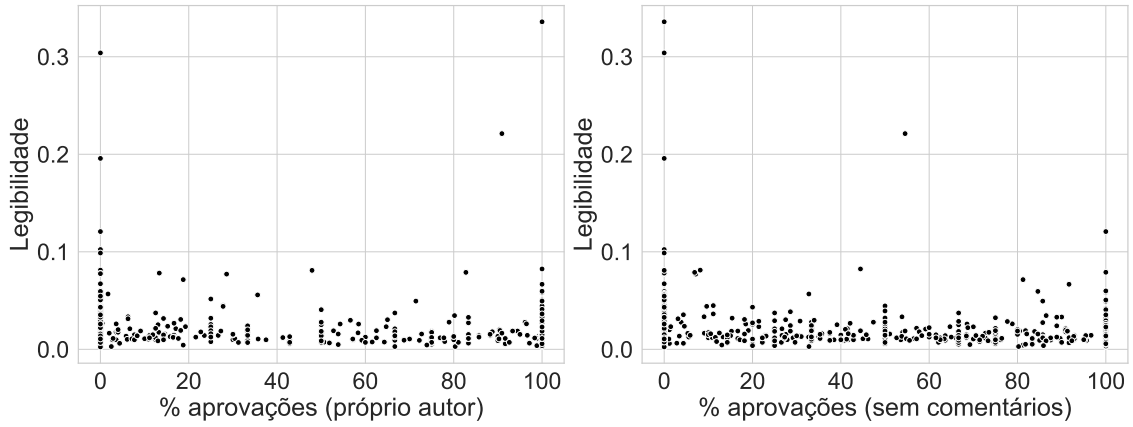
Figura 6. Evolução da proporção de *pull requests* sem comentários

Revisões rápidas. Em relação à distribuição da proporção de revisões rápidas ao longo das *releases*, observa-se que a maior concentração dos dados está abaixo de 20%, sugerindo que este é um dos atributos de menor impacto. Por exemplo, os valores de mediana são zero em todos os períodos.

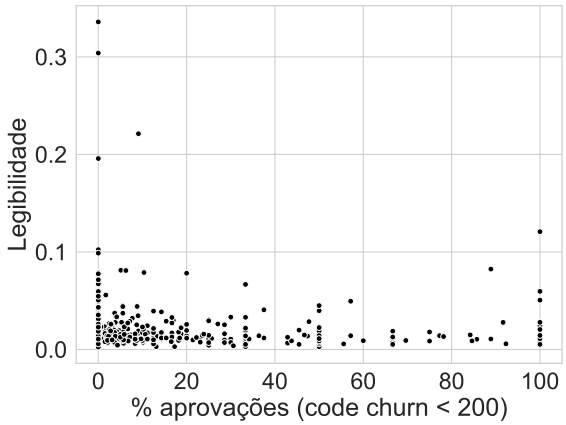
5.4. Relação entre legibilidade e engajamento das *releases* no GitHub

A correlação entre legibilidade e engajamento é realizada através do cálculo do coeficiente de Spearman. Seguindo valores de referência previamente definidos na literatura [Silva and Valente 2018, Hinkle et al. 2003], interpreta-se o teste de Spearman como: $0,00 \leq \text{rho} < 0,30$ (correlação insignificante), $0,30 \leq \text{rho} < 0,50$ (correlação fraca), $0,50 \leq \text{rho} < 0,70$ (correlação moderada), $0,70 \leq \text{rho} < 0,90$ (correlação forte), e $0,90 \leq \text{rho} < 1$ (correlação muito forte). Correlacionando a legibilidade com engajamento da última *release* dos projetos, obtém-se o coeficiente de correlação (rho) igual a 0,024, sugerindo uma relação insignificante entre as variáveis. Entretanto, o valor de *p-value* é igual a 0,4861.

A Figura 7 mostra os gráficos de dispersão, considerando a legibilidade em relação à proporção de cada métrica de engajamento da última *release* dos projetos. Correlacionando a legibilidade e os *pull requests* com revisões aprovadas pelo próprio autor, obtém-se o coeficiente de Spearman igual a -0,1099 (*p-value* = 0,0013), indicando que a correlação entre essas duas variáveis é insignificante (Figura 7(a)). Em relação às revisões aprovadas sem comentários (Figura 7(b)), o resultado do coeficiente da correlação é igual a -0,0867 (*p-value* = 0,0117). Resultados similares são encontrados na correlação entre tempo de revisão e a legibilidade, rho = -0,0519 (Figura 7(c)).



(a) Aprovação do próprio autor por legibilidade (b) *Pull requests* sem comentários por legibilidade



(c) Revisões rápidas por legibilidade

Figura 7. Correlação da legibilidade com cada métrica de engajamento

6. Discussão dos Resultados

Nesta seção são apresentas as discussões e implicações geradas pelas questões respondidas neste estudo.

Projetos com baixo engajamento. Durante a análise dos resultados é possível observar projetos com baixo engajamento. Por exemplo, existem projetos sem *pull requests* aprovados entre os períodos de *releases*. O projeto `spring-projects/spring-boot` mostra este cenário.[12] Apesar da sua popularidade, existe um baixo engajamento nos períodos analisados, devido ao número de *pull requests* rejeitados. Por exemplo, existe uma taxa significativa de contribuições com *status "declined"*. Dessa forma, os resultados inspiram novas linhas de pesquisa, visando caracterizar e compreender projetos *open-source* populares com baixa taxa de engajamento.

Autoaprovações e distribuição de conhecimento. Durante a análise dos resultados, foi possível notar que 32,58% dos *pull requests* foram autoaprovados (50.596 ocorrências). Além disso, os dados mostram que existe pelo menos uma *release* em cada período onde todos os *pull requests* foram autoaprovados. Esta característica pode ser visualizada na Figura 5. Em resumo, existem *releases* onde um único desenvolvedor é responsável pe-

[12]Disponível em: https://github.com/spring-projects/spring-boot. Último acesso: 15 de Março de 2023.

las modificações e revisões. De fato, resultados semelhantes são reportados na literatura, onde se discute a existência de "heróis" no desenvolvimento de sistemas de *software*, que concentram conhecimento e tarefas [Ferreira et al. 2019, Ricca and Marchetto 2010]. O algoritmo de *Truck Factor* [Ferreira et al. 2019], por exemplo, calcula a distribuição de conhecimento em times de desenvolvimento de *software*. Dessa forma, sugere-se investigações futuras, sobre a relação entre repositórios com altas taxas de autoaprovação e distribuição de conhecimento no projeto.

Engajamento e legibilidade ao longo do tempo. Analisando os dados de engajamento e legibilidade, percebe-se uma baixa variação dos valores entre as *releases*. Os valores de legibilidade, por exemplo, mostram poucas variações ($\sim 0,01$), similar aos estudos sobre legibilidade em outros contextos [Mannan et al. 2018, Barbosa et al. 2022]. Como consequência, não foi possível inferir que existe correlação significativa entre as métricas.

7. Ameaças à Validade

Nesta seção são apresentadas as ameaças à validade deste estudo, assim como as estratégias adotadas para mitigá-las [Wohlin et al. 2012]. Primeiro, em relação à ameaça à validade externa, como usual em estudos empíricos da Engenharia de Software, os resultados não podem ser generalizados para outros contextos, como por exemplo, sistemas privados de empresas e outras linguagens de programação. Porém, este estudo analisa um vasto conjunto de dados, incluindo aproximadamente 900 repositórios populares hospedados no GitHub, mais de 115 mil *pull requests* e cerca de 8,5 mil *releases*.

Sobre as ameaças à validade interna, existe a possibilidade de erros durante o cálculo da legibilidade e do engajamento. Para mitigar este problema, os autores fizeram uso de ferramentas previamente conhecidas. Para o cálculo de legibilidade, utilizou-se a ferramenta *Program Comprehension Metrics* [Barbosa et al. 2022, Mendonça et al. 2020], enquanto para o cálculo do engajamento, utilizou-se dados de APIs do GitHub.

Em relação às ameaças à construção, 154 projetos foram removidos, sendo que 13 projetos foram retirados da análise por apresentarem algum erro no cálculo de legibilidade, similar a estudos anteriores utilizando a mesma ferramenta [Barbosa et al. 2022]. Além disso, 141 projetos foram retirados por não respeitarem o versionamento semântico para os nomes das *releases*. Entretanto, eles representam uma pequena parte do conjunto de dados. Ademais, podem existir diferenças em relação ao tempo de lançamento das *releases*. Para mitigar esta ameaça foi analisado um conjunto de dados composto por 8.590 *releases* provenientes de 859 projetos. Além disso, é comum que diferentes versões dos projetos sejam alvos de mudanças. Por exemplo, versões *patch* incluem normalmente correções de defeitos, enquanto versões *minor* incluem adição de novas funcionalidades. Adicionalmente, este trabalho tem como enfoque *pull requests* que foram integrados ao *branch* principal do projeto, visando evitar *branches* temporárias e instáveis.

Por fim, ainda com relação à construção, o cálculo de engajamento se baseia na ocorrência de três métricas: autoaprovações; tempo de validação e comentários. Estas são discutidas individualmente, tendo como base estudos no mesmo contexto [McIntosh et al. 2014]. Ademais, para análise de autoaprovações utilizou-se os revisores distintos vinculados ao *pull request*. Porém, existe a possibilidade da aprovação sem discussões. Para mitigar essa ameaça, fez-se uso do campo *reviews* provido pela API

do GitHub, além de que o número de comentários também é uma métrica utilizada para computar o engajamento.

8. Conclusão

Nesta pesquisa, investigou-se como ocorre a evolução do engajamento e legibilidade em projetos de código aberto. Buscou-se, também, compreender como estes dois fatores se relacionam ao longo do tempo. Com este intuito, foram analisados aproximadamente 900 projetos Java populares da plataforma GitHub. Assim, coletou-se as últimas 10 *releases* de cada projeto, resultando na análise de legibilidade de 8.590 *releases* distintas. Nos períodos entre *releases* coletou-se um total de 155.296 *pull requests*. Apresenta-se a seguir os principais resultados desse estudo:

- A legibilidade das *releases* se manteve estável durante as versões, com valores de mediana variando entre 0,0139 e 0,0133.
- O engajamento também obteve uma baixa variação.
- Observou-se autoaprovações em todos os períodos analisados, representando 32,58% dos *pull requests* analisados (50.596 ocorrências).
- Por fim, não foi possível inferir uma relação entre o impacto do engajamento na legibilidade, devido às características do conjunto de dados. Entretanto, discute-se implicações relacionadas às métricas computadas, como por exemplo, gestão de conhecimento e taxa de engajamento.

Como trabalhos futuros, sugere-se realizar estudos com equipes de desenvolvimento, visando compreender como a melhoria do engajamento na revisão de código impacta na qualidade do *software*. Adicionalmente, sugere-se um estudo sobre o impacto de atividades de refatoração na legibilidade de projetos, visto que é uma prática comumente realizada para melhorar o código-fonte [Silva et al. 2016, Pantiuchina et al. 2020]. Por fim, sugere-se extensões deste trabalho considerando outras bases de dados além do GitHub, como por exemplo, *Gerrit*.[13]

Pacote de Replicação

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

`https://github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2022-2-tcci-5308100-pes-joao-marcos-samuel-baker`

Referências

Barbosa, L. F., Pinto, V. H., de Souza, A. L. O. T., and Pinto, G. (2022). To what extent cognitive-driven development improves code readability? In *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '22, page 238–248, New York, NY, USA. Association for Computing Machinery.

Brito, A., Hora, A., and Valente, M. T. (2021). Characterizing refactoring graphs in Java and JavaScript projects. *Empirical Software Engineering*, 26(6).

[13]https://www.gerritcodereview.com/

Buse, R. P. and Weimer, W. R. (2010). Learning a metric for code readability. *IEEE Transactions on Software Engineering*, 36(4):546–558.

Chouchen, M., Ouni, A., Kula, R. G., Wang, D., Thongtanunam, P., Mkaouer, M. W., and Matsumoto, K. (2021). Anti-patterns in modern code review: Symptoms and prevalence. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 531–535.

Cunha, A., Conte, T., and Gadelha, B. (2021). Code review is just reviewing code? a qualitative study with practitioners in industry. In *Brazilian Symposium on Software Engineering*, SBES '21, page 269–274, New York, NY, USA. Association for Computing Machinery.

Dabic, O., Aghajani, E., and Bavota, G. (2021). Sampling projects in github for MSR studies. In *18th IEEE/ACM International Conference on Mining Software Repositories, MSR 2021*, pages 560–564. IEEE.

Ebert, F., Castor, F., Novielli, N., and Serebrenik, A. (2019). Confusion in code reviews: Reasons, impacts, and coping strategies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 49–60.

Fakhoury, S., Roy, D., Hassan, A., and Arnaoudova, V. (2019). Improving source code readability: Theory and practice. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, pages 2–12.

Ferreira, M., Mombach, T., Valente, M. T., and Ferreira, K. (2019). Algorithms for estimating truck factors: A comparative study. *Software Quality Journal*, 1:1–37.

GitHub (2021). The 2021 state of the octoverse. Disponível em: `https://octoverse.github.com`. Acesso em: 25 de Outubro 2022.

Grijó, L. and Hora, A. (2018). Minerando código comentado. 6th Brazilian Workshop on Software Visualization, Evolution and Maintenance (VEM), pages 1–8.

Han, D., Ragkhitwetsagul, C., Krinke, J., Paixao, M., and Rosa, G. (2020). Does code review really remove coding convention violations? In *2020 IEEE 20th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 43–53.

Han, X., Tahir, A., Liang, P., Counsell, S., and Luo, Y. (2021). Understanding code smell detection via code review: A study of the openstack community. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*, pages 323–334.

Hinkle, D., Wiersma, W., and Jurs, S. (2003). *Applied Statistics for the Behavioral Sciences*. Houghton Mifflin.

Hora, A., Silva, D., Robbes, R., and Valente, M. T. (2018). Assessing the threat of untracked changes in software evolution. In *40th International Conference on Software Engineering (ICSE)*, pages 1102–1113.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 92–101, New York, NY, USA. Association for Computing Machinery.

Mannan, U. A., Ahmed, I., and Sarma, A. (2018). Towards understanding code readability and its impact on design quality. In *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering*, NL4SE 2018, page 18–21, New York, NY, USA. Association for Computing Machinery.

Martins, M. F. F. (2022). Análise da influência da multitarefa de desenvolvedores no ciclo de vida de issues do github. In *Trabalhos de Conclusão de Curso. Engenharia de Software. PUC Minas*, pages 1–19.

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 192–201, New York, NY, USA. Association for Computing Machinery.

Mendonça, W. L. M., Fortes, J., Lopes, F. V., Marcílio, D., de Almeida, R. B., Canedo, E. D., Lima, F., and Saraiva, J. (2020). Understanding the impact of introducing lambda expressions in java programs. *Journal of Software Engineering Research and Development*, 8:7–1.

Neto, L., Silva, G., and Comarela, G. (2021). Estimativa do tempo de resolução de issues no github usando atributos textuais e temporais. In *35th Simpósio Brasileiro de Engenharia de Software (SBES*.

Pantiuchina, J., Zampetti, F., Scalabrino, S., Piantadosi, V., Oliveto, R., Bavota, G., and Penta, M. D. (2020). Why developers refactor source code: A mining-based study. *ACM Transactions on Software Engineering and Methodology*, 37(4):1–32.

Pascarella, L., Spadini, D., Palomba, F., and Bacchelli, A. (2019). On the effect of code review on code smells. *CoRR*, abs/1912.10098.

Posnett, D., Hindle, A., and Devanbu, P. (2011). A simpler model of software readability. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, page 73–82, New York, NY, USA. Association for Computing Machinery.

Raymond, D. R. (1991). Reading source code. In *Proceedings of the 1991 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '91, page 3–16. IBM Press.

Ricca, F. and Marchetto, A. (2010). Are heroes common in floss projects? In *4th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–4.

Scalabrino, S., Linares-Vásquez, M., Oliveto, R., and Poshyvanyk, D. (2018). A comprehensive model for code readability. *Journal of Software: Evolution and Process*, 30(6):e1958. e1958 smr.1958.

Silva, D., Tsantalis, N., and Valente, M. T. (2016). Why we refactor? Confessions of GitHub contributors. In *24th International Symposium on the Foundations of Software Engineering (FSE)*, pages 858–870.

Silva, H. and Valente, M. T. (2018). What's in a GitHub star? Understanding repository starring practices in a social coding platform. *Journal of Systems and Software*, 146:112–129.

Tufano, R., Pascarella, L., Tufano, M., Poshyvanyk, D., and Bavota, G. (2021). Towards automating code review activities. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 163–174.

Uchôa, A., Barbosa, C., Oizumi, W., Blenilio, P., Lima, R., Garcia, A., and Bezerra, C. (2020). How does modern code review impact software design degradation? an in-depth empirical study. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 511–522.

Ueda, Y., Ishio, T., Ihara, A., and Matsumoto, K. (2019). Mining source code improvement patterns from similar code review works. In *2019 IEEE 13th International Workshop on Software Clones (IWSC)*, pages 13–19.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in software engineering*. Springer Science & Business Media.