Uma Análise sobre a Complexidade de Refatorações Documentadas em Issues

André Silva Maurício da Rocha[1], **Augusto Coutinho de Freitas**[1]

[1]Instituto de Informática e Ciências Exatas
Pontifícia Universidade Católica de Minas Gerais (PUC-Minas)
30.140-100 – Belo Horizonte – MG – Brasil

{andre.silva,augusto.coutinho}@sga.pucminas.br

Abstract. *Refactoring is an indispensable practice during software evolution. Frequently, developers document refactoring needs as issues. In this context, it is relevant to understand the effort required to perform such transformations, aiming to help software developer teams manage their issues tasks. Therefore, in this paper, we investigate the main characteristics of refactorings documented in issues. In particular, we assess their life cycle, effort, and resources. For this purpose, we analyze about 269K textual patterns related to refactorings, in issues from 1K popular GitHub projects. The results show that issues involving feature addition may require more developer interaction. Also, the issue's life cycle is diverse, ranging from days to months. We conclude by discussing the results and prospecting future studies.*
Keywords: *Refactoring, Issues, Mining Software Repositories, Textual Patterns*

Resumo. *A refatoração é uma prática indispensável durante a evolução do software. Frequentemente, os desenvolvedores reportam necessidades de refatoração do código em issues. Nesse contexto, é relevante entender o esforço necessário para realizar tais transformações, visando ajudar as equipes de desenvolvimento de software no gerenciamento das issues. Portanto, neste trabalho, investiga-se as principais características de refatorações documentadas em issues. Em particular, avalia-se o ciclo de vida, esforço e recursos demandados. Para tanto, analisa-se cerca de 269 mil padrões textuais relacionados com refatorações, em issues provenientes de 1.000 projetos populares do GitHub. Os resultados mostram que problemas envolvendo adição de recursos podem exigir mais interação dos desenvolvedores. Além disso, o ciclo de vida das issues é diverso, variando de dias até meses. Por fim, discute-se os resultados e sugere-se novos estudos neste contexto.*
Palavras-chave: *Refatoração, Issues, Mineração de Repositórios de Software, Padrões Textuais*

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

Orientador de conteúdo (TCC I): José Laerte Pires Xavier Júnior - jlpxjunior@sga.pucminas.br
Orientadora de conteúdo (TCC I): Simone de Assis Alves da Silva - saasilva@sga.pucminas.br
Orientador de conteúdo (TCC I): Cleiton Silva Tavares - cleiton.tavares@sga.pucminas.br
Orientadora do TCC II: Aline Brito - alinebrito@pucminas.br

Belo Horizonte, 14 de Maio de 2023.

1. Introdução

Com a crescente demanda da produção de software, surgiram áreas e técnicas responsáveis por garantir a qualidade e o bom funcionamento dos produtos [Sommerville 1988]. Dentre tais técnicas, existe a refatoração de código, prática fundamental no desenvolvimento de software, que consiste no processo de alterar um sistema de software, sem alterar o comportamento externo e melhorando a estrutura interna [AlOmar et al. 2022b]. A refatoração melhora a manutenibilidade do código, favorecendo a extensibilidade e a sua reutilização [Liu et al. 2012]. A documentação de software é outra técnica para alcançar os objetivos anteriormente citados, mas que notoriamente é negligenciada pelos desenvolvedores, devido a pressões em relação ao tempo e dinheiro [McBurney et al. 2018]. A documentação ajuda a manter uma fonte válida e oficial de informações sobre um sistema software, além de ajudar os desenvolvedores a entender o código mais rapidamente, melhorando assim sua produtividade. Considerando estas duas técnicas, é possível que equipes de desenvolvimento de software trabalhem com a documentação de refatorações, que pode ser útil, por exemplo, em projetos de grande porte e com alto grau de complexidade.

Estudos anteriores focam em recomendar refatorações de código a partir da detecção de métricas de qualidade [Mkaouer et al. 2015, Sellitto et al. 2022] ou anti-padrões de desenvolvimento [Oizumi et al. 2020]. Outros trabalhos também evidenciaram que há uma diferença entre o que é considerado uma situação clara de necessidade de refatoração por parte dos desenvolvedores e ferramentas automatizadas para atividades de refatoração [Fernandes et al. 2020, AlOmar et al. 2019b]. Nesse aspecto, observa-se um estudo anterior que preocupa-se em responder algumas questões relacionadas à refatoração de código, como por exemplo, quais padrões textuais os desenvolvedores utilizam para expor suas necessidades de refatoração [AlOmar et al. 2022b]. Os autores abordaram quais atributos de qualidade de código os programadores se preocupam quando estão realizando alguma atividade de refatoração. Contudo, também constataram uma necessidade de estudo e compreensão em relação ao esforço demandado pelas operações de refatoração sinalizadas por padrões ou termos nas *issues*. Portanto, o problema que esse trabalho propõe-se a resolver **é a escassez de análises da complexidade de refatorações reportadas em issues de repositórios públicos no GitHub, identificadas a partir de padrões textuais.**

De acordo com Dyba et al. (2009), as práticas ágeis de desenvolvimento trouxeram mudanças na abordagem e na aplicação da refatoração de código-fonte. A ideia de constantemente aperfeiçoar um código já desenvolvido é visto com bons olhos na maioria das equipes ágeis. Entretanto, esta melhoria contínua na qualidade do código implica em custos a longo prazo. Neste contexto, a literatura apresenta diversos estudos sobre documentação de refatorações [AlOmar et al. 2019a, AlOmar et al. 2022b, Barrozo and Vinhas 2012]. No entanto, a análise da refatoração considerando padrões textuais é um tema pouco explorado na literatura. Dessa forma, esta pesquisa mostra-se relevante para a compreensão dos possíveis esforços presentes na execução de refatorações de código.

Desse modo, este trabalho tem como objetivo **identificar quais são as refatora-**

ções mais complexas de serem resolvidas, agrupando-as e dividindo-as através de padrões textuais utilizados nas *issues* criadas pelos desenvolvedores, de modo a mitigar a falta de documentação nesse assunto. Alguns objetivos específicos desse trabalho são: (1) identificar o esforço demandado para solucionar as *issues* documentadas a partir de repositórios que tem como principal linguagem de programação o Java; (2) identificar quais refatorações têm apresentado maior complexidade na resolução pelos desenvolvedores, agrupando-as por seus padrões textuais; (3) analisar quais as refatorações que necessitaram de mais recursos (quantidade de desenvolvedores e tempo gasto) para serem solucionadas, agrupando-as por seus padrões textuais.

Visando alcançar estes objetivos, neste trabalho propõe-se as seguintes questões de pesquisa:

- Qual a complexidade e esforço demandado para solucionar *issues* relacionadas com refatorações?
- Qual o ciclo de vida das *issues* relacionadas com tarefas de refatoração?
- Qual a frequência de interações em *issues* relacionadas com tarefa de refatoração?

Os resultados podem contribuir para decisões relacionadas ao esforço demandado para realizar tarefas associadas com refatorações. Para tanto, analisa-se características relacionadas com tempo despendido, esforço, e engajamento dos desenvolvedores, de acordo com os padrões textuais.

Em relação ao restante do conteúdo do trabalho, a Seção 2 apresenta o referencial teórico com o detalhamento de alguns conceitos relevantes na área do estudo. A Seção 3 conta com os trabalhos relacionados da área, enquanto a Seção 4 descreve os métodos que são utilizados neste estudo. Apresenta-se os resultados obtidos na Seção 5, seguida pela Seção 6 que promove uma discussão detalhada dos mesmos. As ameaças à validade deste estudo são abordadas na Seção 7, onde são identificados possíveis vieses ou limitações metodológicas. Por fim, a Seção 8 apresenta as conclusões finais do trabalho, destacando os principais resultados e possíveis direções futuras de pesquisa.

2. Referencial Teórico

Nesta seção, são descritos e apresentados conceitos importantes para a compreensão deste trabalho. Na Seção 2.1 é apresentado o conceito de refatoração de código. Na Seção 2.2 é descrita a atividade de documentação de artefatos de software e na Seção 2.3 são aprentadas as ferramentas GitHub e GitHub Issues.

2.1. Refatoração de Código

A refatoração de código é uma atividade que compõe o ciclo de vida de um software que precisa se manter escalável, manutenível, performático e operante, conforme o tempo passa [Fowler 2004]. De acordo com Barrozo et al. (2012), a refatoração surgiu através da observação da dificuldade de inserir novas funcionalidades em projetos de software já existentes. Essa dificuldade foi constatada devido à desestruturação do código, além da existência de muitas duplicações de código e também difícil compreensão do mesmo. “Refatoração é uma alteração feita na estrutura interna do software para torná-lo mais fácil de ser entendido e menos custoso de ser modificado sem alterar seu comportamento observável” [Fowler 2004].

De acordo com Markovi et al. (2007), a atividade de refatoração é composta por seis etapas: (1) identificar qual bloco do código deve sofrer refatoração; (2) determinar quais refatorações devem ocorrer no bloco identificado; (3) garantir que a refatoração aplicada preserve o comportamento esperado pelo sistema; (4) aplicar a refatoração; (5) avaliar o efeito da refatoração aplicada nas características de qualidade do software; (6) manter a consistência entre o código do software e outros artefatos existentes (manuais, documentação, etc).

Adicionalmente, refatoração é um termo frequentemente utilizado por desenvolvedores para indicar outras atividades, como por exemplo, mudanças relacionadas com requisitos não-funcionais [Valente 2022].

2.2. Documentação de Artefatos de Software

A documentação de software é um conjunto de textos, manuais e ilustrações que detalha como o sistema opera e como deve ser usado [Kajko-Mattsson 2005]. Sua interpretação e entendimento podem variar de pessoa para pessoa, a depender do conhecimento técnico e função exercida de quem a lê. Com isso, ela pode apresentar diversas funções, desde dar o suporte necessário para os times de desenvolvimento quanto ajudar o usuário a obter mais conhecimento do software [Souza et al. 2005].

Como essa atividade é comumente utilizada para registrar vários artefatos do software, ela é praticada em várias etapas do ciclo de vida de um sistema, desde sua criação, atualização, modificação e exclusão [Kajko-Mattsson 2005]. Isso pode englobar diversos artefatos como testes, desenhos arquiteturais, detalhamento de código-fonte, requisitos levantados e necessidades de refatoração. Tais documentações precisam abstrair aspectos técnicos do que se está em pauta de modo que sua linguagem e entendimento sejam universais, atendendo as demandas dos usuários finais, que pretendem utilizar do software para alcançar um objetivo ou realizar uma ação [Souza et al. 2005].

Tais documentações realizadas, sejam elas através de diversas ferramentas, devem ser feitas com cautela. Por conta disso, é comum demandarem um alto custo de operação para que sejam escritas. Isso se deve ao fato de que são importantes em projetos de qualquer escala de tamanho. Imprecisões e omissões na documentação podem levar a erros por parte dos usuários finais e consequentes falhas do sistema com seus custos e interrupções associados. Portanto, gerentes e engenheiros devem prestar tanto atenção à documentação e seus custos associados quanto ao desenvolvimento do próprio software [Sommerville 2001].

2.3. GitHub e GitHub Issues

De acordo com Bissyandé et al. (2013), o GitHub[1] é uma plataforma popular de hospedagem, sendo utilizada por diferentes projetos e equipes para obter uma exposição e visualização na comunidade de desenvolvedores. Uma ferramenta comumente utilizada para a documentação de refatorações em projetos que mantém o seu código fonte na plataforma GitHub é o GitHub Issues. Essa ferramenta permite que desenvolvedores rastreiem e mantenham o histórico de todo o trabalho que é realizado a respeito de um problema que pode ser relatado e identificado como uma *issue*. O GitHub Issues fornece importan-

[1] https://Github.com.

tes recursos como a marcação de problemas para categorizá-los, de modo a facilitar sua gestão pelos desenvolvedores [Bissyandé et al. 2013].

3. Trabalhos Relacionados

AlOmar et al. (2022) investigaram quais são os padrões de documentação utilizados pelos desenvolvedores para expressar a necessidade de refatoração através de *issues* do GitHub [AlOmar et al. 2022b]. Para atingir o objetivo proposto, foi projetado e conduzido um estudo exploratório que baseia-se na mineração de texto de 45.477 problemas relacionados à refatoração e na identificação de padrões de refatoração de 77 projetos Java de código aberto que utilizam o GitHub como repositório de controle de versão e o JIRA para rastreamento de problemas. O estudo revela que os desenvolvedores são principalmente impulsionados pela redução da complexidade e aumento da compreensão e do desempenho. Enquanto vários estudos firmemente associam a refatoração com a correção de mal cheiros de código, o único anti-padrão que foi destacado é o código duplicado. O artigo se relaciona com este trabalho dado que ambos estudam a utilização de termos e frases relacionados à refatoração para expressar a necessidade de melhorias em blocos específicos de código.

AlOmar et al. (2019) têm como objetivo explorar as estratégias que os desenvolvedores utilizam para documentar suas atividades de refatoração durante o ciclo de vida de desenvolvimento de software. Foi observado que os desenvolvedores de software podem relatar explicitamente atividades de refatoração nas mensagens de *commit* de repositórios versionados, atividade chamada de *Self-Affirmed Refactoring (SAR)*. Também foi descoberto que os desenvolvedores usam uma variedade de padrões para direcionar propositalmente eventos de refatoração, mencionando explicitamente a melhoria de certos atributos de qualidade e cheiros de código. Outra conclusão foi que mensagens de *commit* em repositórios versionados com padrões SAR tendem a ter uma atividade de refatoração mais significativa do que aquelas sem o padrão. O artigo se relaciona com este trabalho pois foi realizado um estudo exploratório sobre como os desenvolvedores documentam suas atividades de refatoração através de mensagens de *commit*.

Rao et al. (2022) realizaram um estudo sobre o quanto os comentários no código afetam a compreensão dos leitores [Rao et al. 2022]. Para atingir o objetivo proposto, foram analisados 700 mil comentários em 2 mil projetos Java e Python do GitHub, sendo propostos vários filtros para identificar quais comentários provavelmente correspondem a uma mudança na documentação e/ou solicitaram tal mudança. Com a identificação de 65 mil casos, foi desenvolvido uma taxonomia das intenções por trás desses "comentários sobre comentários". Esses comentários geralmente se concentram no esclarecimento, seguidos pela indicação de problemas a serem corrigidos, como erros de digitação e comentários desatualizados. A análise quantitativa e qualitativa do estudo destaca o importante papel da documentação na discussão das contribuições do código. O artigo se relaciona com este trabalho dado que ele conclui que alcançar um entendimento compartilhado de código é vital na manutenção e refatorações de projetos de código aberto.

Liu et al. (2022) analisaram a documentação de tipos de refatoração por *commit* e constataram que é muito pouco conhecida a forma com que desenvolvedores documentam as necessidades de refatoração, quando comparada com a documentação de atualização de recursos e correção de *bugs*. Com o objetivo de desafiar a capacidade de documentação

de refatorações, esse trabalho analisou através da mineração de mensagens de *commit*, a extração de padrões textuais específicos para cada tipo de refatoração (por exemplo: renomear, extrair, mover, etc.). Com os resultados obtidos, concluiu-se que existe uma nítida necessidade de melhorar a qualidade de documentação de refatoração. Também foi percebida a necessidade de encorajar a invenção de um gerador de documentação de refatorações feitas, utilizando como base os padrões textuais mais utilizados nos *commits* [AlOmar et al. 2022a]. Este trabalho pode ser considerado um complemento do artigo citado pois, como relatado, identificou-se uma demanda para melhorar a qualidade das refatorações feitas, problema relacionado com o que este trabalho visa sanar, que é a análise da complexidade dessas refatorações.

AlOmar et al. (2021) têm como objetivo explorar as melhorias nas técnicas e métodos de refatoração. Os desenvolvedores podem incorporar algumas estratégias de refatoração em outra atividades relacionadas ao desenvolvimento de software, atividades essas que estão presentes nos desafios emergentes na engenharia de software contemporânea [AlOmar et al. 2021]. O trabalho citado busca uma melhor compreensão do que motiva os desenvolvedores a aplicar uma refatoração, tendo como método a mineração e classificação automática de um grande conjunto de *commits* contendo atividades de refatoração, extraídas de 800 projetos Java de código aberto. Os resultados do trabalho se mostraram bastante complexos e interessantes, indicando que correções de *code smells* não são o principal fator para os desenvolvedores refatorarem seus códigos. Outra conclusão importante do artigo é que a distribuição das operações de refatoração difere entre os arquivos de produção e arquivos de teste. Por fim, a conclusão que mais se relaciona com este trabalho é que os padrões textuais extraídos das mensagens de *commit* fornecem melhor cobertura de como os desenvolvedores documentam suas refatorações. Este trabalho pode ser bastante relacionado com esse artigo, pois o mesmo se propõe a explorar a escassez da documentação das refatorações.

4. Materiais e Métodos

A pesquisa realizada nesse estudo possui caráter quantitativo e descritivo. Busca-se medir, a partir dos valores de métricas coletadas, quais as *issues*, agrupadas por padrões textuais, requerem maior esforço e interação para serem resolvidas. Já o caráter quantitativo está presente pois os valores das métricas coletadas são analisados e traduzidos em números para se chegar em possíveis conclusões.

As próximas subseções detalham a metodologia, padrões textuais de refatoração considerados, bem como as questões de pesquisa propostas. A Figura 1 mostra o fluxo principal de trabalho. A etapa inicial envolve a seleção dos repositórios e *issues*. Em seguida, os dados são agrupados pelos padrões textuais de refatoração. Por fim, realiza-se o cálculo das métricas e análise dos resultados.

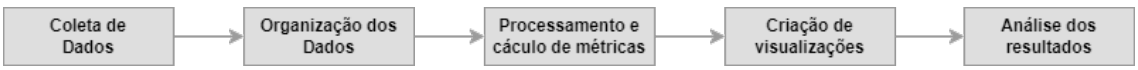


Figura 1. Visão geral da metodologia utilizada

4.1. Questões de Pesquisa

Este trabalho inclui três questões de pesquisa:

(QP1) Qual a complexidade e esforço demandado para solucionar issues relacionadas com refatorações? Nesta primeira questão de pesquisa analisa-se a distribuição de *code churn* presente nos *pull requests* vinculados com as *issues*, por padrão textual de refatoração. Similar à estudos anteriores [Brito and Valente 2021], utiliza-se esta métrica para discutir a complexidade e esforço associado com tarefas de de refatoração.

(QP2) Qual o ciclo de vida das issues relacionadas com tarefas de refatoração? Para responder esta questão de pesquisa, analisa-se o ciclo de vida das *issues*, isto é, o intervalo de tempo entre a data de abertura e fechamento, em dias. Dessa forma, é possível analisar o tempo demandado para solucionar *issues*, que estão associadas com padrões textuais de refatorações.

(QP3) Qual a frequência de interações em issues relacionadas com tarefas de refatoração? Por fim, nesta terceira questão de pesquisa, analisa-se a quantidade de comentários e participantes envolvidos nas *issues*, visando discutir o engajamento [McIntosh et al. 2014].

4.2. Mineração de Repositórios

Neste trabalho, analisou-se 1.000 projetos *open source* populares hospedados na plataforma GitHub. O estudo concentra-se na linguagem de programação Java, devido a sua popularidade.[2] Os repositórios populares foram selecionados considerando a quantidade de estrelas, visto que é uma métrica relevante [Silva and Valente 2018, Borges et al. 2016]. Entretanto, no GitHub, podem existir projetos que não são necessariamente sistemas de software. Logo, adota-se outros critérios com o objetivo de identificar projetos de interesse [Neto et al. 2021, Martins 2022]. Para a seleção dos projetos, utilizou-se os seguintes critérios adicionais:

- **C1:** não é *fork*, para evitar resultados duplicados [Martins 2022, Brito et al. 2021, Hora et al. 2018, Brito et al. 2020].
- **C2:** possui mais de mil *commits*, com o intuito de evitar projetos com baixa atividade [Brito et al. 2021, Hora et al. 2018];
- **C3:** possui pelo menos 100 *issues*, visando projetos que fazem uso do sistema de gerenciamento [Neto et al. 2021, Martins 2022].

Considerando os critérios estabelecidos, a coleta de repositórios foi realizada através da ferramenta *GitHub Search* (GHS) [Dabic et al. 2021].

4.3. Coleta de *Issues*

Após a coleta dos repositórios, obteve-se todas as *issues* fechadas de cada repositório pela API REST do GitHub, em um script automatizado na linguagem *Python*. Dentre as *issues*, analisou-se apenas aquelas associadas com padrões textuais que remetem à atividades de refatoração. A Tabela 1 mostra os padrões textuais considerados, que são definidos em trabalhos anteriores [AlOmar et al. 2019b].

Para cálculo do *code churn* de cada *issue*, é necessário utilizar o seu identificador único, obtido no passo anterior. A partir desse identificador, recupera-se todos os eventos associados àquela *issue*, incluindo os *pull requests*. Por fim, a partir dos *pull*

[2]Ranking GitHub de linguagens populares de 2022 disponível em: https://octoverse.github.com/2022/top-programming-languages. Ultimo acesso: 20 fev. 2023

requests, é possível identificar a quantidade de linhas alteradas. Para calcular a taxa de *code churn* é considerada a soma das linhas adicionadas e removidas em cada *pull request* [Nagappan and Ball 2005].

Tabela 1. Lista de padrões textuais que apontam necessidade de refatorações ('*' captura a extensão da palavra-chave).

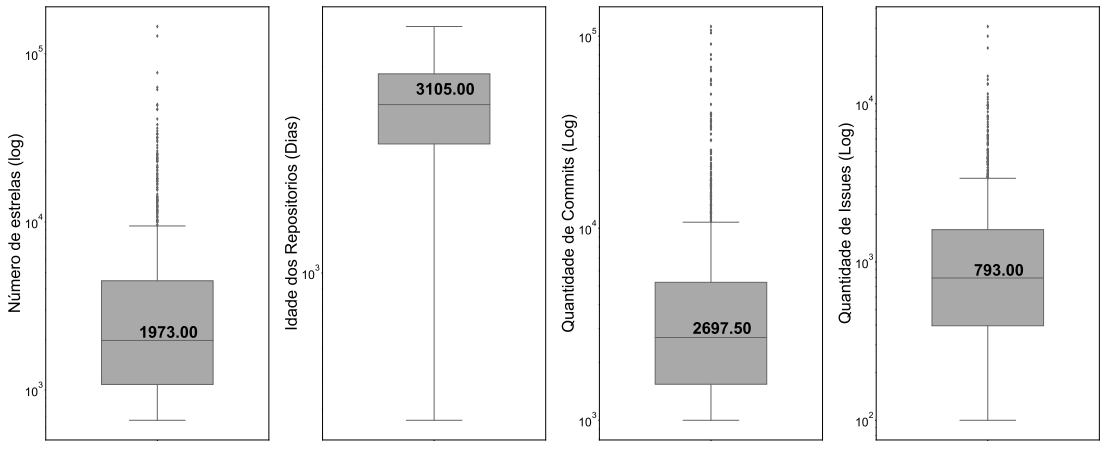
Padrões Textuais		
Add*	Change*	Chang* the name
Clean* up code	Cleanup	Code clean*
Code optimization	Creat*	Extend*
Extract*	Fix*	Fix* code style
Improv*	Improv* code quality	Inlin*
Introduc*	Merg*	Mov*
Pull* up	Push* down	Repackag*
Redesign*	Reduc*	Refactor*
Refin*	Remov*	Renam*
Reorganiz*	Replac*	Restructur*
Rewrit*	Simplify* code	Split*

5. Resultados

Os resultados apresentados nesta seção incluem a caracterização do dados coletados e os resultados das métricas calculadas para as questões de pesquisa definidas na Seção 4.1.

5.1. Caracterização do Conjunto de Dados

A caracterização do conjuntos de dados foi realizada pela elaboração de gráficos *boxplot* que permitem a visualização da dispersão dos valores obtidos, conforme apresentado na Figura 2. Nos próximos parágrafos discute-se as características dos 1.000 projetos GitHub analisados, assim como o percentual de issues coletadas.



(a) *Estrelas* (b) Idade (c) Commits (d) *Issues*

Figura 2. Distribuição de estrelas, idade, *commits* e *issues* por projeto GitHub

Repositórios. A Figura 2(a) mostra a distribuição do número de estrelas por projeto GitHub. Os valores variam entre entre 661 e 14.5397, com a mediana igual 1.973, o primeiro quartil em 661 e o terceiro quartil em 1.973. O conjunto de dados inclui projetos populares, como por exemplo, `Spring`, um *framework* popular para criação de aplicações *web*, com mais de 51 mil estrelas, e `Google Gson`, uma biblioteca de serialização e desserialização de arquivos *JSON*, com mais de 22 mil estrelas. A Figura 2(d) apresenta a distribuição do número de *issues* em cada repositório. Os número de estrelas varia entre 100 e 30.812, com a mediana igual a 793, o primeiro quartil igual a 497 e o terceiro quartil igual a 793. Em relação à idade dos projetos (Figura 2(b)), os valores variam entre 315 e 5.195 dias, com a mediana de 3.049 dias (i.e., aproximadamente 8 anos). Por fim, o número de *commits* variam entre 1.000 e 112.236, sendo que 75% dos projetos possuem até 2.697 *commits*, conforme apresentado na Figura 2(c).

Issues. Neste trabalho, analisa-se 1.514.552 *issues* dos repositórios, visando identificar e caracterizar *issues* relacionadas com tarefas de refatoração. Conforme apresentado na Figura 2(d), o número de issues por projeto varia entre 100 e 30.812, com a mediana em 793. Observa-se que a proporção de *issues* fechadas é significativamente maior do que a de *issues* abertas. Especificamente, identifica-se 1.284.882 *issues* fechadas representam (84,8%), e 229670 *issues* abertas (15,2%). Este trabalho concentra-se em *issues* fechadas, que incluem pelo menos um padrão textual relacionado com refatorações (216.349 *issues*).

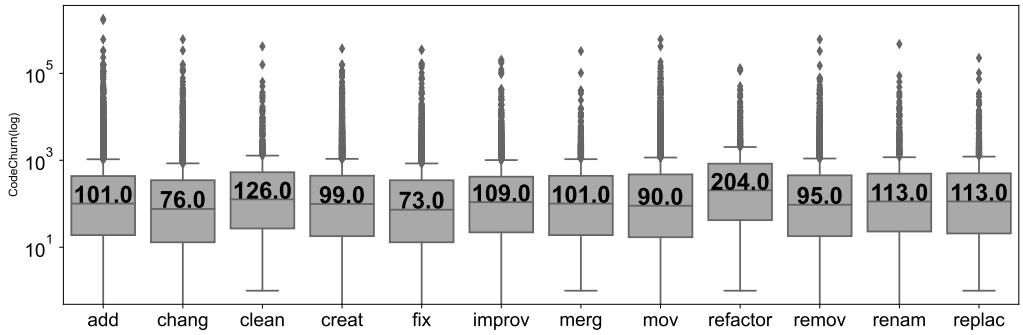
5.2. Qual a complexidade e esforço demandado para solucionar *issues* relacionadas com refatorações?

Nesta primeira questão de pesquisa, busca-se ocorrências de padrões textuais em *issues* associadas com *pull requests*. Dentre as 269.916 ocorrências de padrões textuais, foram selecionadas 47.363 que possuem *pull requests* vinculados (17,5%). Em seguida, agrupa-se as *issues* com essas ocorrências, considerando 32 padrões textuais relacionados com tarefas de refatoração. A Figura 3 mostra a distribuição de *code churn* por *ocorrência*, considerando os padrões frequentes. Estes padrões representam aproximadamente 99% dos resultados (47.229 ocorrências).

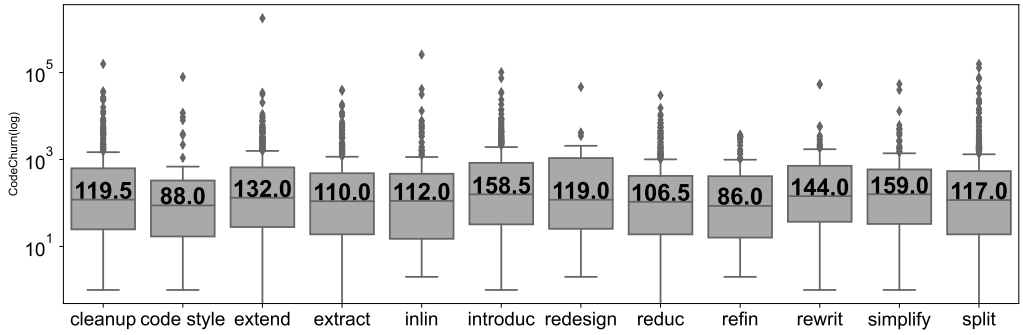
Observa-se que as maiores medianas referem-se aos padrões textuais *refactor*, *simplify*, *introduc* e *rewrit*, com os valores de 204, 159, 158.5 e 144 de *code churn*, respectivamente. Considerando todas as issues, os *outilers* incluem, por exemplo, as *labels add*, *chang* e *mov*, que incluem *issues* com code churn $\geq$ 60 mil, isto é, mais 60 mil linhas modificadas. Por exemplo, o projeto `adempiere` inclui este cenário, onde 157.481 linhas de código foram modificadas, em uma *issue* associada com a *label add*.[3]

Outras issues associadas com refatorações. Considerando a distribuição do número de *issues* por padrão textual, a mediana consiste em aproximadamente 524 *issues*. O primeiro quartil inclui oito *labels* menos recorrentes, com o número de *issues* variando entre 5 e 28. Por exemplo, *pull up* e *push down* são as *labels* menos frequentes, 5 e 10 issues, respectivamente. As demais *issues* incluem as *labels restructur* (28 ocorrências), *code optimization* (23), *reorganiz* (22), *code clean* (17), *code quality* (15), e *repackag** (13). Os valores de *code churn* variam entre 0 e 36.430.

[3]https://github.com/adempiere/adempiere/issues/1586



(a) *Padrões textuais recorrentes*



(b) *Padrões textuais menos frequentes*

Figura 3. Distribuição de code churn por padrão textual de refatoração

5.3. Qual o ciclo de vida de *issues* relacionadas com tarefas de refatoração?

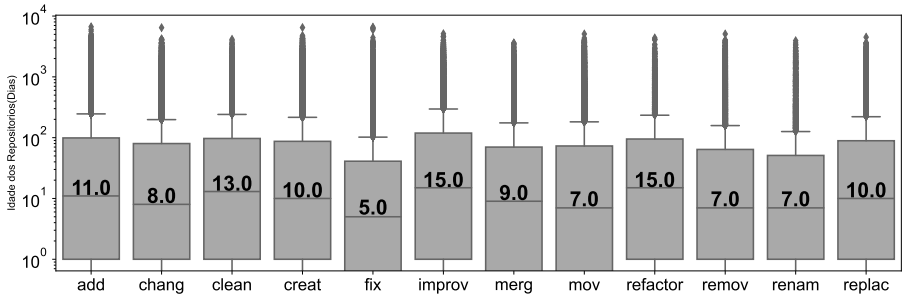
Nesta segunda questão de pesquisa, analisa-se o ciclo de vida de *issues* relacionadas com tarefas de refatoração. Especificamente, verifica-se o intervalo de tempo entre a a data de criação e conclusão das *issues*. Calcula-se a durabilidade das 216.349 *issues* que foram encerradas, relacionadas com as 269.916 ocorrências de padrões textuais, considerando o número de dias. Em seguida, as *issues* são agrupadas em 32 padrões textuais relacionados com refatorações.

A Figura 4 mostra a distribuição do ciclo de vida das *issues*, ordenadas pela frequência dos padrões textuais. Considerando os valores de mediana, os padrões com ciclos de vida maiores são *redesign* (40,5 dias), *introduc* (19 dias), *simplify* e *cleanup* (ambas com 16 dias).

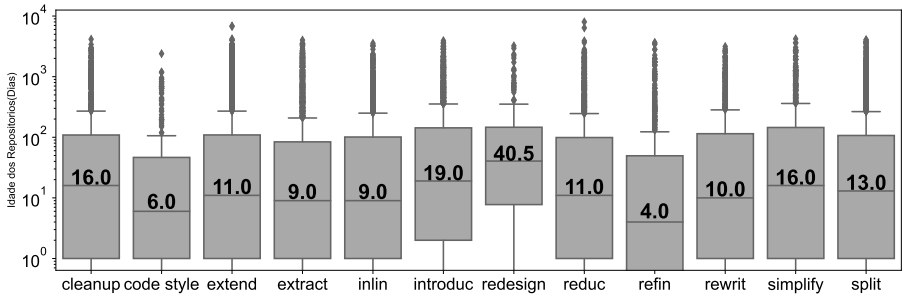
A Figura 5 mostra um exemplo, a ocorrência do padrão textual *refactor* no projeto `spring-kafka`.[4] Pode-se observar que no corpo da *issue*, que existe uma discussão sobre refatorações em larga escala, associadas com uma funcionalidade do sistema. Neste caso, a *issue* foi encerrada em 75 dias.

Outras issues associadas com refatorações. Assim como na primeira questão de pesquisa, o primeiro quartil também representa os oito padrões textuais menos recorrentes. Dentre eles, o tempo de resolução das *issues* varia entre zero e aproximadamente 95 meses (isto é, 2.911 dias).

[4]https://github.com/spring-projects/spring-kafka/issues/1745



(a) *Padrões textuais recorrentes*



(b) *Padrões textuais menos frequentes*

Figura 4. Distribuição do ciclo de vida das *issues* por padrão textual de refatoração

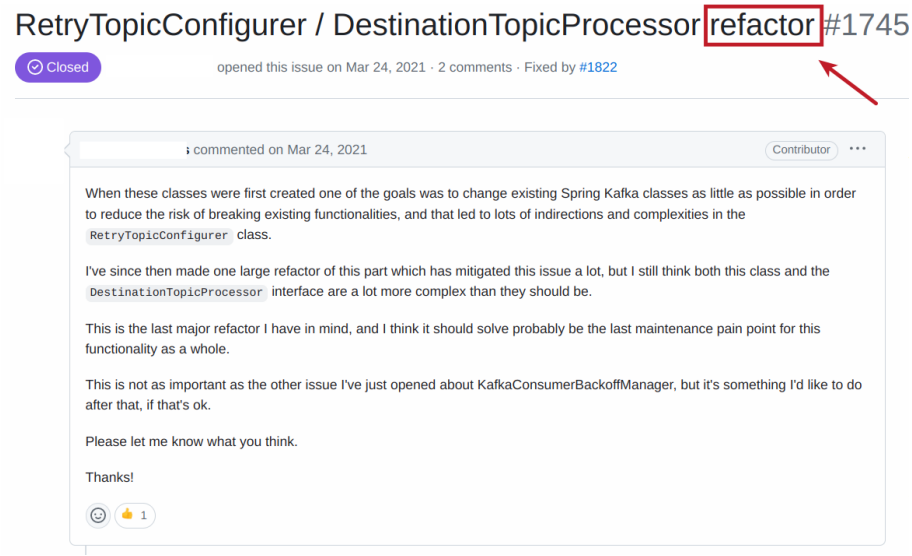
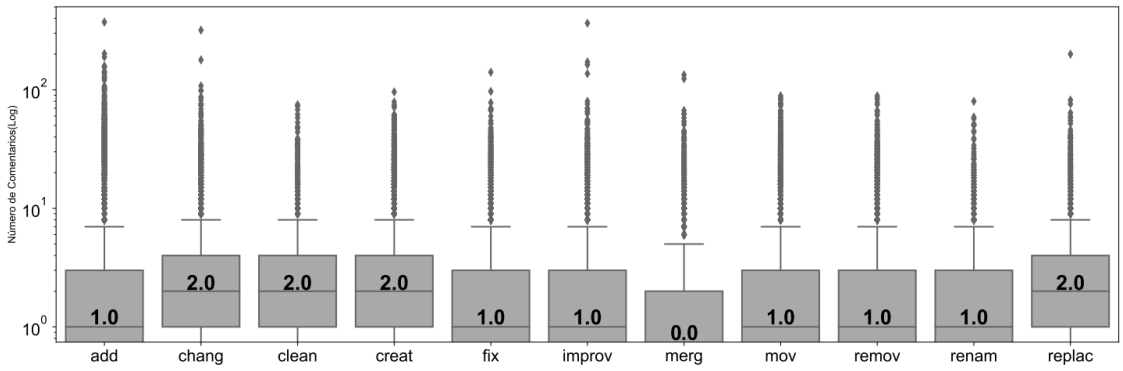


Figura 5. Exemplo de issue relacionada com padrão *refactor* (Spring Kafka)

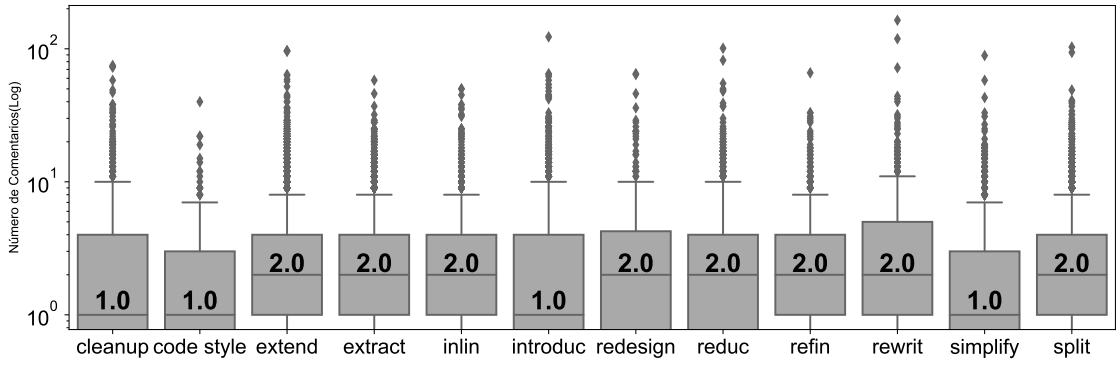
5.4. Qual a frequência de interações em *issues* relacionadas com tarefas de refatoração?

Por fim, nesta terceira questão de pesquisa analisa-se a frequência de interações em *issues* com ocorrências de padrões textuais relacionados com tarefas de refatorações. Especificamente, investiga-se o número de comentários e participantes distintos. Similar às demais análises, as *issues* são agrupadas por padrões textuais.

A Figura 6 mostra a distribuição do número de interações por ocorrência de padrão textual, considerando os padrões mais frequentes. Em relação à mediana, detecta-se até dois comentários e dois participantes. Os valores seguem uma similar tendência para os demais padrões relacionados com tarefas de refatoração. O padrão textual *add* inclui *outliers*, como por exemplo, uma *issue* do projeto `PojavLauncher`, que inclui 373 comentários.[5]



(a) *Distribuição de comentários (Padrões textuais frequentes)*



(b) *Distribuição de participantes (Padrões textuais frequentes)*

Figura 6. Distribuição do número de interações por ocorrência de padrão textual relacionada com refatorações

Considerando todas as *issues*, os valores variam entre 0 e 89 participantes, e entre 0 e 373 comentários. A Tabela 2 mostra a frequência de ocorrência de padrões textuais por número de participantes. Cerca de 14,19% das ocorrências (38.313) envolvem a atuação de mais de cinco participantes, enquanto 75,08% dos resultados envolvem até três participantes (202.671 ocorrências).

Em relação ao número de comentários, pode-se observar pela Tabela 3, que existe uma taxa significativa de ocorrência dos padrões textuais com até três comentários 84,74% (228.746 ocorrências). Ao todo, 269.395 padrões textuais tiveram pelo menos um comentário durante o ciclo de vida da sua *issue* associada.

[5]https://github.com/PojavLauncherTeam/PojavLauncher/issues/59

Tabela 2. Participantes por ocorrência de padrões textuais relacionados com tarefas de refatoração

#	Ocorrência	%
0	70.777	26,22
1	67.750	25,10
2	39.417	14,60
3	24.727	9,16
4	17.169	6,36
5	11.773	4,36
6+	38.313	14,19
Total	269.916	100

Tabela 3. Comentários por ocorrência de padrões textuais relacionados com tarefas de refatoração

#	Ocorrência	%
0	521	0,19
1	79.429	29,43
2	98.745	36,58
3	50.051	18,54
4	22.070	8,18
5	9.612	3,56
6+	9.488	3,52
Total	269.916	100

6. Discussão dos Resultados

Esta seção apresenta discussões e implicações baseadas nas questões de pesquisa respondidas neste estudo.

Issues relacionadas com os padrões textuais *refactor*, *simplify*, *introduc* e *rewrit* demandam maior esforço. Na primeira questão de pesquisa, discute-se o esforço e complexidade em *issues* relacionadas com refatorações. Utiliza-se a métrica de *code chrun*, usualmente considerada para discutir complexidade de refatorações [Brito and Valente 2021]. Os maiores valores referem-se aos padrões *simplify*, *introduc*, *refactor*, *rewrit*, considerando a mediana. Considerando as *issues* mais frequentes, os padrões *add*, *chang* e *mov* apresentaram os maiores valores de *code chrun*, isto é, estas *issues* incluem *outliers* do conjunto de dados. Os grupos de *issues* com maior dispersão nos dados foram *code optimization, code quality, push down e redesign*, com intervalos interquartis de 1.710,5, 1.133, 1.197,5 e 1.054,5, respectivamente.

O ciclo de vida de *issues* relacionadas com refatorações é diverso. A segunda questão de pesquisa permite observar que o tempo de resolução de *issues* pode variar significativamente. Algumas *issues* são encerradas rapidamente, como por exemplo, em até um dia. Entretanto, algumas issues possuem um ciclo de vida longo, demandando meses para serem finalizadas. De fato, trabalhos recentes discutem que atividades de refatoração evoluem ao longo do tempo [Brito et al. 2020, Brito et al. 2021]. Entretanto, estes estudos

não concentram-se no contexto de *issues* e padrões textuais. Além disso, é comum que projetos de software possuam tempo de inatividade [Coelho et al. 2020], o que poderia influenciar no ciclo de vida das *issues*.

A maioria das *issues* incluem padrões textuais relacionados com correção de defeitos, adição, melhoria, e movimentação de código-fonte. Os resultados sugerem que os desenvolvedores geralmente realizam refatorações em resposta a problemas específicos encontrados no código. Por exemplo, os resultados obtidos neste estudo mostraram que a maioria das *issues* incluem padrões relacionadas com correção de defeitos, adição, melhoria e movimentação de código-fonte. De fato, trabalhos recentes mostram que desenvolvedores frequentemente refatoram o código por estas razões [Silva et al. 2016, Pantiuchina et al. 2020, Brito et al. 2021].

***Issues* relacionadas com adição de funcionalidades envolvem as maiores taxas de interações de desenvolvedores.** Na terceira questão de pesquisa, discute-se a frequência de interações de desenvolvedores em issues relacionadas com tarefas de refatoração. Para a maioria dos padrões, os valores de mediana concentram-se em aproximadamente dois comentários e participantes. Entretanto, o padrão *add* destaca-se nos resultados. Por exemplo, existem *issues* relacionadas com este padrão textual quem incluem centenas de comentários e participação de diversos desenvolvedores. De fato, trabalhos recentes discutem que refatorações são frequente realizadas para adicionar novas funcionalidades [Silva et al. 2016]. Dessa forma, os resultados complementam estes estudos, sugerindo que adição de novas funcionalidades, além de ser um motivador para novas refatorações, pode ser uma tarefa custosa em termos de interação de desenvolvedores.

7. Ameaças à Validade

Nesta seção são apresentadas as ameaças a validade deste estudo, assim como as estratégias adotadas para mitigá-las. Primeiro, em relação à ameaça à validade externa, os resultados do estudo não possam ser generalizados para outros cenários, como por exemplo, outros ecossistemas de software ou sistemas privados de empresas. Entretanto, este trabalho utilizou um conjunto de dados diverso, incluindo 269.916 ocorrências de padrões textuais detectados em *issues*, provenientes de 1.000 projetos Java populares da plataforma GitHub.

Em relação às ameaças internas, existe a possibilidade de falhas durante a coleta coleta de dados. Por exemplo, a API GraphQL do GitHub pode ser interrompida ou ficar indisponível, e pode haver erros nos *scripts* de coleta de dados. Para mitigar essa ameaça, foram criadas estratégias para lidar com interrupções e falhas na coleta de dados, utilizando declarações *try-except* para capturar exceções que podem ocorrer ao fazer solicitações.

Por fim, em relação à validade de construção, este trabalho concentrou-se em *issues* com padrões textuais comumente relacionados com refatorações. Portanto, o código vinculado com a *issue* pode incluir mudanças que não necessariamente catalogadas com operações de refatoração. Entretanto, os padrões textuais baseiam-se em estudos anteriores sobre práticas de refatoração [AlOmar et al. 2019b]. Além disso, este trabalho contempla também o conceito mais amplo de refatoração, que pode indicar, por exemplo, a *"melhoria de outros requisitos não-funcionais, que não estão relacionados com manutenibilidade"* [Valente 2022].

8. Conclusão

Neste trabalho, investigou-se o esforço, ciclo de vida, e interações dos desenvolvedores em *issues* relacionadas com tarefas de refatoração. Para tanto, foram analisadas 269.916 ocorrências de padrões textuais relacionados com tarefas de refatoração, em *issues* provenientes de 1.000 projetos em Java populares da plataforma GitHub. Especificamente, estas *issues* incluem padrões textuais que remetem à necessidade de refatoração.

Com relação à primeira questão de pesquisa, observou-se que, dentre os padrões textuais mais frequentes, *rewrit*, *refactor*, *introduc* e *simplify* demandam um maior esforço, conforme indicado pelos valores de *code churn*. Na segunda questão de pesquisa, verificou-se que o ciclo de vida das *issues* de refatoração pode variar significativamente. *Issues* com alguns padrões textuais, como *redesign*, *introduc*, *simplify e cleanup* apresentaram maior tempo para serem resolvidos. Já na terceira questão de pesquisa, foi constatado que as *issues* relacionadas com adição de funcionalidades apresentam as maiores taxas de interações de desenvolvedores, sugerindo uma maior complexidade e entendimento para implementação.

Como trabalhos futuros sugere-se a utilização de ferramentas para a detecção e análise de refatorações, como RefDiff [Brito and Valente 2020, Silva et al. 2021] e RefactoringMiner [Tsantalis et al. 2020], visando caracterizar operações específicas. Estas ferramentas podem fornecer informações detalhadas sobre as mudanças realizadas no código, permitindo uma análise considerando outros cenários. Além disso, estas ferramentas possuem suporte para outras linguagens de programação, permitindo a extensão do estudo em outros ecossistemas. Por exemplo, pode-se verificar se linguagens de programação interpretadas como JavaScript possuem padrões textuais de refatoração específicos.

Pacote de Replicação

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

`github.com/ICEI-PUC-Minas-PPLES-TI/plf-es-2022-2-tcci-5308100-pes-augusto-coutinho-e-andre-silva`

Referências

AlOmar, E., Mkaouer, M. W., and Ouni, A. (2019a). Can refactoring be self-affirmed? an exploratory study on how developers document their refactoring activities in commit messages. In *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*, pages 51–58.

AlOmar, E. A., Liu, J., Addo, K., Mkaouer, M. W., Newman, C., Ouni, A., and Yu, Z. (2022a). On the documentation of refactoring types. *Automated Software Engg.*, 29(1).

AlOmar, E. A., Mkaouer, M. W., Ouni, A., and Kessentini, M. (2019b). On the impact of refactoring on the relationship between quality attributes and design metrics. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–11. IEEE.

AlOmar, E. A., Peruma, A., Mkaouer, M. W., Newman, C., Ouni, A., and Kessentini, M. (2021). How we refactor and how we document it? on the use of supervised

machine learning algorithms to classify refactoring documentation. *Expert Systems with Applications*, 167:114176.

AlOmar, E. A., Peruma, A., Mkaouer, M. W., Newman, C. D., and Ouni, A. (2022b). An exploratory study on refactoring documentation in issues handling. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 107–111.

Barrozo, G. and Vinhas, H. (2012). *Refatoração: Aperfeiçoando um código existente*. Bookman.

Bissyandé, T. F., Lo, D., Jiang, L., Réveillère, L., Klein, J., and Traon, Y. L. (2013). Got issues? who cares about it? a large scale investigation of issue trackers from github. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 188–197.

Borges, H., Hora, A., and Valente, M. T. (2016). Understanding the factors that impact the popularity of GitHub repositories. In *32nd International Conference on Software Maintenance and Evolution (ICSME)*, pages 334–344.

Brito, A., Hora, A., and Valente, M. T. (2020). Refactoring graphs: Assessing refactoring over time. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 367–377.

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

Brito, R. and Valente, M. T. (2020). RefDiff4Go: Detecting refactorings in Go. In *14th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS)*, pages 101–110.

Brito, R. and Valente, M. T. (2021). RAID: Tool support for refactoring-aware code reviews. In *29th International Conference on Program Comprehension (ICPC)*, pages 1–11.

Coelho, J., Valente, M. T., Milen, L., and Silva, L. L. (2020). Is this GitHub project maintained? measuring the level of maintenance activity of open-source projects. *Information and Software Technology*, 1:1–35.

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.

Dyba, T. and Dingsoyr, T. (2009). What do we know about agile software development? *IEEE Software*, 26(5):6–9.

Fernandes, E., Chávez, A., Garcia, A., Ferreira, I., Cedrim, D., Sousa, L., and Oizumi, W. (2020). Refactoring effect on internal quality attributes: What haven't they told you yet? *Information and Software Technology*, 126:106347.

Fowler, M. (2004). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley.

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.

Kajko-Mattsson, M. (2005). A survey of documentation practice within corrective maintenance. *Empirical Software Engineering*, 10(1):31–55.

Liu, H., Gao, Y., and Niu, Z. (2012). An initial study on refactoring tactics. In *2012 IEEE 36th Annual Computer Software and Applications Conference*, pages 213–218.

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.

McBurney, P. W., Jiang, S., Kessentini, M., Kraft, N. A., Armaly, A., Mkaouer, M. W., and McMillan, C. (2018). Towards prioritizing documentation effort. *IEEE Transactions on Software Engineering*, 44(9):897–913.

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.

Mkaouer, W., Kessentini, M., Shaout, A., Koligheu, P., Bechikh, S., Deb, K., and Ouni, A. (2015). Many-objective software remodularization using nsga-iii. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):1–45.

Nagappan, N. and Ball, T. (2005). Use of relative code churn measures to predict system defect density. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 284–292.

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*.

Oizumi, W., Bibiano, A. C., Cedrim, D., Oliveira, A., Sousa, L., Garcia, A., and Oliveira, D. (2020). Recommending composite refactorings for smell removal: Heuristics and evaluation. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, SBES '20, page 72–81, New York, NY, USA. Association for Computing Machinery.

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.

Rao, N., Tsay, J., Hirzel, M., and Hellendoorn, V. J. (2022). Comments on comments: Where code review and documentation meet. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 18–22.

Sellitto, G., Iannone, E., Codabux, Z., Lenarduzzi, V., De Lucia, A., Palomba, F., and Ferrucci, F. (2022). Toward understanding the impact of refactoring on program comprehension. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 731–742.

Silva, D., da Silva, J. P., Santos, G., Terra, R., and Valente, M. T. (2021). Refdiff 2.0: A multi-language refactoring detection tool. *IEEE Transactions on Software Engineering*, 47(12):2786–2802.

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.

Sommerville, I. (1988). *Software Engineering*. Addison-Wesley.

Sommerville, I. (2001). *Software documentation*. Addison-Wesley.

Souza, S. C. B., Anquetil, N., and de Oliveira, K. M. (2005). A study of the documentation essential to software maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*, SIGDOC '05, page 68–75, New York, NY, USA. Association for Computing Machinery.

Tsantalis, N., Ketkar, A., and Dig, D. (2020). RefactoringMiner 2.0. *IEEE Transactions on Software Engineering (TSE)*.

Valente, M. T. (2022). *Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade*. Independente.