UNIVERSIDADE FEDERAL DE CAMPINA GRANDE
CENTRO DE ENGENHARIA ELÉTRICA E INFORMÁTICA
UNIDADE ACADÊMICA DE SISTEMAS E COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

JOSÉ RAMON FRAGOSO DA SILVA

UM ESTUDO QUALITATIVO SOBRE O IMPACTO DE REVISÕES DE CÓDIGO EM REFATORAMENTOS NO CONTEXTO DE PULL REQUETS

CAMPINA GRANDE - PB

2021

JOSÉ RAMON FRAGOSO DA SILVA

UM ESTUDO QUALITATIVO SOBRE O IMPACTO DE REVISÕES DE CÓDIGO EM REFATORAMENTOS NO CONTEXTO DE PULL REQUETS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Professor Dr. Everton Leandro Galdino Alves.

CAMPINA GRANDE - PB

2021

S586e Silva, José Ramon Fragoso da.
Um estudo qualitativo sobre o impacto de revisões de código em refatoramentos no contexto de pull requests. / José Ramon Fragoso da Silva. – 2021.

10 f.

Orientador: Prof. Dr. Everton Leandro Galdino Alves.
Trabalho de Conclusão de Curso - Artigo (Curso de Bacharelado em Ciência da Computação) - Universidade Federal de Campina Grande; Centro de Engenharia Elétrica e Informática.

1. Pull requests. 2. Comentários – pull requests. 3. Github. 4. Código-fonte. 5. Repositórios Java. 6. Projeto Apache. 7. Desenvolvedores de software. 8. Revisores. 9. Refatoramento I. Alves, Everton Leandro Galdino. II. Título.

CDU:004(045)

Elaboração da Ficha Catalográfica:

Johnny Rodrigues Barbosa
Bibliotecário-Documentalista
CRB-15/626

JOSÉ RAMON FRAGOSO DA SILVA

UM ESTUDO QUALITATIVO SOBRE O IMPACTO DE REVISÕES DE CÓDIGO EM REFATORAMENTOS NO CONTEXTO DE PULL REQUETS

Trabalho de Conclusão Curso apresentado ao Curso Bacharelado em Ciência da Computação do Centro de Engenharia Elétrica e Informática da Universidade Federal de Campina Grande, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

BANCA EXAMINADORA:

Professor Dr. Everton Leandro Galdino Alves
Orientador – UASC/CEEI/UFCG

Professora Dra. Patrícia Duarte de Lima Machado
Examinador – UASC/CEEI/UFCG

Professor Dr. Tiago Lima Massoni
Disciplina TCC – UASC/CEEI/UFCG

Trabalho aprovado em: 25 de outubro 2021.

CAMPINA GRANDE - PB

ABSTRACT

Pull-based development is a fundamental practice that constitutes Modern Code Review (MCR) as developers can contribute to code improvements such as edits to refactorings. An analysis of comments made by reviewers on pull requests provides new insight into doing refactorings in Open Source projects. In this article we define *pull requests* into *RIPRs* (Refactoring-Inducing Pull Request), when a Pull Request has refactorings in any of its commits and this refactoring happened because of some reviewer comment, and *nonRIPRs* (non-Refactoring-Inducing Pull Request), that is a Pull Request that has comments but no refactoring happened. The objective of this study is to analyse Pull Requests from repositories written in Java, all of them collected from Apache's Software Foundation most popular repositories. With the results collected from this analysis our focus is to characterize review comments, and understand how these comments affect on how the author or *Pull Requests* modify their code, making refactorings when needed. In order to achieve these goals, 3 questions were developed by the research team, involved in this work: (i) How do reviewers suggest refactorings in review comments in RIPRs? (ii) Do the reviewers justify their reasons? (iii) What is the relation between refactoring recommendations and actual refactorings? With these questions we can characterize review comments, understanding the components that compose a comment, and how these components affect change in code. After analysis, some results were observed by the researchers in both RIPRs and nonRIPRS; comments had characteristics and patterns that could occur. These patterns were compared with the occurrence of refactorings and the results were shown in the Results section. The results showed that RIPRs and nonRIPRs had similar or different characteristics: In all of *Pull Requests* we observed that the reviewers showed the exact location in code where the changes should be made. In RIPRs the patterns observed were: Comments can directly suggest a refactoring, or can suggest other changes in code that indirectly can result in refactorings. Comments can also have the presence of justifications about the suggestions and it also can be written in a more polite way, asking the author if changes could be made, or more directly, when an obvious refactoring should be made. In *nonRIPRs* it was observed that there are reasons a refactoring could not occur: The presence of a discussion between the author and the reviewer after a suggestion, making this suggestion not implemented. The other reason is when a reviewer suggests changes that don't result in refactoring, and other changes were made. The main impact of this research is to expand the knowledge about contemporary code review in *Pull Requests*, providing implications that can contribute to other researchers, developers and programmers.

Um Estudo Qualitativo sobre o Impacto de Revisões de Código em Refatoramentos no Contexto de Pull Requests

José Ramon Fragoso da Silva
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
jose.ramon.silva@ccc.ufcg.edu.br

Everton Leandro Galdino Alves
Universidade Federal de Campina Grande
Campina Grande, Paraíba, Brasil
everton@computacao.ufcg.edu.br

RESUMO

Os comentários feitos em *pull requests (PRs)* de repositórios Git, podem induzir a refatoramentos, que são melhorias feitas no código-fonte, preservando o seu comportamento. Com base nos comentários de revisão, desenvolvedores são capazes de decidir por refatorar o código-fonte. Este trabalho tem como objetivo analisar PRs de repositórios Java do projeto Apache [19] no GitHub à luz dos comentários dos revisores e refatoramentos realizados em tempo de revisão de código (minerados previamente pelo *RefactoringMiner* - uma ferramenta estado-da-arte para a detecção de refatoramentos). Para tanto, propõem-se uma análise manual de comentários de revisão em uma base de dados que contém 118 PRs. Foram definidos pontos específicos a serem analisados em cada comentário, buscando atingir objetivos específicos definidos para ajudar na pesquisa. Essa análise foi feita de forma manual, lendo, caracterizando e verificando o comportamento desses pontos levantados e fazendo anotações sobre cada *Pull Request*. Ao final deste trabalho foram notados diversos padrões que se repetiam entre os PRs, esses padrões foram compilados e serão apresentados, e com isso espera-se entender de que forma os comentários de revisão influenciam nos refatoramentos de código-fonte.

Palavras Chave

Refatoramento, *pull requests*, *commit, git*, *github*, comentários, revisores, código-fonte, versionamento, repositório.

1. INTRODUÇÃO

O refatoramento de código-fonte desempenha um importante papel no desenvolvimento de software, pois promove mudanças no código sem que isso modifique seu comportamento enquanto favorece a manutenção e a legibilidade do código [1]. Edições de refatoramento também facilitam a descoberta de bugs e o tratamento de *code smells* [5]. Por exemplo, para o *smell Long Method* [2], que denota um método muito extenso, é possível utilizar o refatoramento do tipo *Extract Method* [3], que reestrutura o método original em um ou mais métodos menores, melhorando a legibilidade do código.

No contexto de desenvolvimento de software, existem ferramentas que auxiliam no desenvolvimento colaborativo, tais como o *Github* [6], a maior plataforma de código colaborativo do mundo. Neste modelo de desenvolvimento, para realizar mudanças em código que está armazenado em um repositório, faz-se necessário a criação de uma *branch* com as alterações e em seguida realizar um *Pull Request* (PR) [7]. Um PR é uma solicitação para que as alterações sejam inseridas na branch principal desse repositório.

Estes conceitos fazem parte do que é chamado de *Revisão Moderna de Código* (MCR) [14], uma técnica de garantia de qualidade, capaz de identificar defeitos e melhorias no código, e acelerar o processo de desenvolvimento. O MCR é composto de 6 etapas [15]:

1. Preparação: o desenvolvedor modifica algum código fonte, e o envia para revisão.
2. Seleção: é feita uma seleção automática ou manual de quem irá revisar o código.
3. Notificação: o revisor recebe uma notificação virtual.
4. Checagem de código: o revisor inspeciona o código enviado e compara com a versão antiga.
5. Interação: revisores disponibilizam feedbacks com ou sem o autor, e discutem as mudanças.
6. Decisão: O revisor decide se irá aprovar as mudanças, rejeitar, ou propor a implementação de mais mudanças por parte do autor.

Entender quando refatoramentos são realizados no contexto de PRs e revisões de código pode ser importante para melhor gerenciar e promover a melhoria contínua do código de um projeto. Para esse trabalho, classificamos os PRs em dois tipos: *Refactoring-Inducing Pull Requests* (RIPRs), que contém apenas PRs que sofreram algum tipo de refatoramento como resultado de uma revisão de código ou apenas como ação espontânea do desenvolvedor, e *non-Refactoring-Inducing Pull Requests* (nonRIPRs), com PRs que não contém refatoramentos.

Nesse contexto, para que mudanças sejam aplicadas ao código, os desenvolvedores elaboram PRs que são revisados por outros desenvolvedores (revisores), que analisam se as mudanças sugeridas estão corretas, além de propor melhorias [4]. Essa comunicação ocorre por meio de comentários nos PRs.

Neste trabalho, realizaremos um estudo onde foram avaliados manualmente um conjunto de PRs coletados de projetos open-source. Para cada PRs, buscamos entender melhor como os revisores de código sugerem refatoramentos, e a partir disso identificar relações entre as sugestões e os refatoramentos aplicados.

2. OBJETIVO

Esta pesquisa tem como objetivo perceber como comentários são feitos em PRs, suas características e como esses comentários se relacionam com a realização de posteriores refatoramentos. A análise de PRs foi a forma usada para se chegar a este objetivo, já que antes de analisar PRs, foram definidos pontos a serem observados em cada comentário, o que é explicado com mais detalhes na seção de metodologia. Três questões de pesquisa foram definidas para nos ajudar a chegar ao objetivo:

- Q1: Como revisores sugerem refatoramentos em comentários de review em RIPRs?
- Q2: Os revisores justificam suas razões?

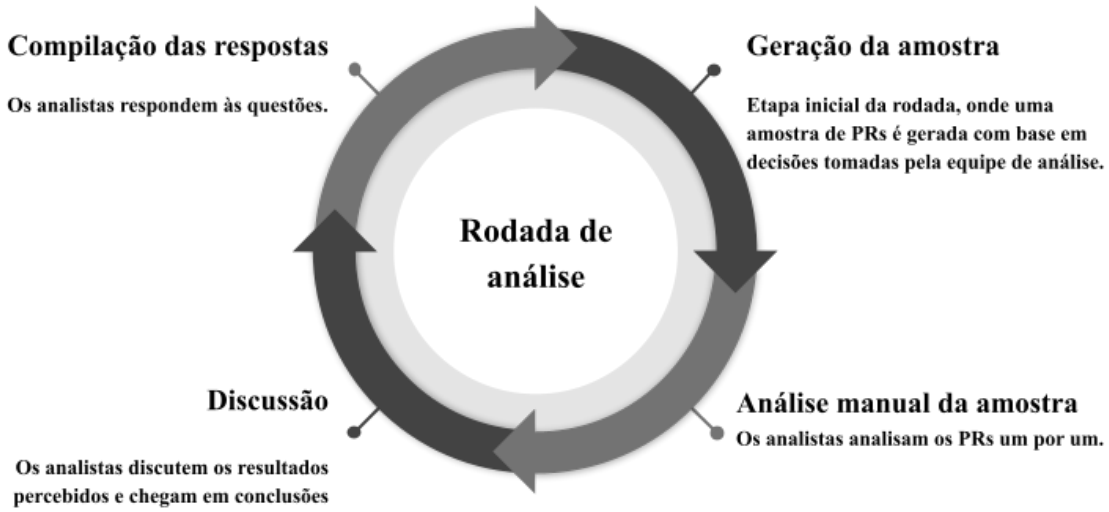

Figura 1. Representação de uma rodada de análise.

- Q3: Qual a relação entre recomendações de refatoramento e reais refatoramentos em RIPRs.

3. METODOLOGIA

A metodologia deste trabalho ocorreu por meio da análise de RIPRs. Os refatoramentos foram minerados utilizando a ferramenta *RefactoringMiner* [8], que é uma ferramenta desenvolvida em *Java*, capaz de detectar refatoramentos que ocorreram no histórico de commits de um projeto *Java*. Para este trabalho, utilizamos uma análise incremental, onde novos dados foram introduzidos até até que houvesse uma saturação, onde novas conclusões não foram identificadas. Desta forma, a análise ocorreu em quatro etapas, que são chamadas de rodadas. A cada rodada, a equipe de analistas analisou uma amostra diferente, e discutiu o que foi percebido em cada análise.

Após a discussão, os analistas responderam individualmente às perguntas do objetivo deste trabalho, para que fosse feita uma comparação entre resultados percebidos entre diferentes ciclos. Ao final de cada rodada, a equipe os analistas identificavam padrões entre os PRs e para ter uma melhor investigação sobre algum padrão observado, era julgada a necessidade de alguma mudança na forma que a próxima rodada de análise seria feita.

3.1 Amostragem de dados

No início de cada rodada foi gerada uma lista de RIPRs para ser analisada. Para gerar essa lista foi usada a ferramenta *RefactoringMiner*. Este software coletou dados de repositórios da Fundação Apache. Escolhemos estes projetos devido a sua popularidade, já que a Apache contém mais de 350 repositórios públicos, contando com mais de 7 mil contribuidores. O *RefactoringMiner* possui uma precisão de 99.6% para detectar refatoramentos em projetos Java, e 94% de revocação [10], além disso o *RefactoringMiner* é capaz de detectar 40 tipos diferentes de refatoramento em sua versão 1.0.0, que foi utilizada neste trabalho.

A lista de PRs gerada pelo *RefactoringMiner* foi disponibilizada em uma tabela *csv* [11]. Algumas colunas desta tabela já eram preenchidas, de forma a auxiliar os analistas. Estas colunas tinham os seguintes nome e seguintes informações:

- repo: Repositório Apache origem do PR.
- pr_number: Número de identificação do PR dentro do repositório.
- category: Indicação caso o PR seja RIPR ou nonRIPR.
- pr_url: Link para o PR.
- commit: *Secure Hash Algorithm* (SHA) [9] do commit dentro do repositório.
- commit_url: Link para o commit.
- initial_flag: Indicação caso o commit seja inicial ou não.
- refactoring_type: Tipo do refatoramento feito.
- refactoring_detail: Detalhes sobre o refatoramento (Arquivo específico e variável, método ou classe que foi modificado.)

A tabela continha também campos vazios, que deveriam ser preenchidos pelos analistas, e a partir do link para o PR, os analistas conseguiam ter acesso a todo o histórico de comentários e todas as mudanças ocorridas naquele PR, preenchendo os seguintes campos:

- confirmed_refactoring_flag: Indicação se o PR realmente ocorreu. Possíveis valores: VERDADEIRO e FALSO.
- covered_refactoring_flag: Indicação se houve algum comentário feito que induziu ao refatoramento. Possíveis valores: VERDADEIRO e FALSO.
- floss_refactoring_flag: Indicação se o autor do PR realizou mudanças no código além do refatoramento. Possíveis valores: VERDADEIRO e FALSO.
- notes: Esse campo foi destinado aos analistas escreverem comentários, caso houvesse algum comportamento anormal em algum *commit*.

467 repositórios foram selecionados para a geração das amostras, a escolha desses repositórios se deu por meio da busca textual no *github* utilizando o seguinte termo: *user:apache language:java archived:false*.

3.2 Equipe de análise

Para gerar um resultado mais imparcial e sem viés, foi formada uma equipe de três analistas. Os analistas realizavam o mesmo trabalho, individualmente, em cada rodada: analisar os mesmos PRs, preencher os campos da tabela de PRs, tirar suas próprias conclusões sobre análise e levar isso para uma discussão. O motivo dessa análise ser feita de forma manual se dá ao fato de que os comentários de revisão são feitos usando linguagem natural, e que precisamos entender a semântica desses textos, o que dificulta uma análise automática.

Para que essa análise fosse possível, foi composta uma equipe com 3 analistas. Todos possuem conhecimento sobre refatoramento, *pull requests*, *commit, git*, *github*, comentários, revisores, código-fonte, versionamento, e repositórios. Todos os analistas são estudantes da área de Computação e por vontade própria aceitaram realizar as análise

3.3 Análise das amostras

Os dados coletados na amostragem de dados foram analisados individualmente e manualmente pelos analistas, sendo que os mesmos PRs eram analisados por cada analista. Nessa análise, verificou-se cada PR, validando a ocorrência dos refatoramentos. Em seguida, analisou-se os comentários de revisão e discussões nos PRs. Após analisar todos os PRs de cada rodada os analistas se reuniam e discutiam sobre os resultados e sobre suas percepções individuais e assim decidiam se haveria alguma mudança na análise da próxima rodada. Ao todo foram analisados 122 PRs, sendo 63 RIPRs e 59 nonRIPRs. As rodadas serão melhor detalhadas a seguir:

3.3.1 Rodada 1

Na primeira rodada foram analisados 20 PRs, sendo 10 RIPRs e 10 nonRIPRs. Na discussão entre os analistas foi notado que em alguns PRs havia discussão entre o revisor e o autor, logo, a equipe tomou a decisão de inserir um campo a ser preenchido na próxima rodada de análise, o *discussion_flag*, que indica se houve discussão antes de algum commit, e seus possíveis valores eram VERDADEIRO e FALSO. Também foi percebido que as sugestões feitas por revisores podem ser feitas de forma direta, onde o comentário sugere um refatoramento, ou pode ser feito de forma indireta, onde o revisor sugere alguma mudança no código, não relacionado com refatoramento, mas que acarreta em um refatoramento por parte do autor. Por isso, mais um campo foi adicionado para ser preenchido na próxima rodada, o *direct_review_comment_flag*, que representa se houve alguma sugestão feita de forma direta, com os valores possíveis VERDADEIRO e FALSO.

3.3.2 Rodada 2

Na segunda rodada foram analisados 20 PRs, sendo 10 RIPRs e 10 nonRIPRs. Após análise e discussão dos analistas na rodada anterior foi decidido que na rodada 2 a amostra conteria PRs com apenas um *commit* que não seja inicial, para que fosse possível explorar PRs mais simples e poder comparar RIPRs com nonRIPRs.
Ao final da segunda rodada, foi decidido que a próxima amostra seria composta de PRs aleatórios, onde haveria a presença de refatoramentos relacionados a herança de código, e também PRs onde há mudança no design do código, para que fosse possível verificar se os padrões percebidos se repetiam.

3.3.3 Rodada 3

Na terceira rodada foram analisados 26 PRs, sendo 13 RIPRs e 13 nonRIPRs. Ao final da análise, os analistas perceberam que os revisores de código podem ou não justificar suas razões para que as mudanças sugeridas fossem feitas, assim decidiu-se adicionar mais um campo a ser analisado na próxima rodada, o rationale_flag, que indica se algum comentário contém ou não justificativas, e os possíveis valores são VERDADEIRO e FALSO.

3.3.4 Rodada 4

Na quarta e última rodada foram analisados 52 PRs, sendo 26 RIPRs e 26 nonRIPRs. Para essa análise, foram selecionados PRs que continham sequências de refatoramentos. Isso foi decidido devido ao interesse dos analistas em observar se os padrões percebidos até então se repetem em PRs mais complexos, onde há mais sugestões e mais *commits*.

3.5 Compilação de respostas

Depois de cada rodada de análise, cada um dos analistas respondeu individualmente às questões propostas no objetivo deste estudo de acordo com sua visão sobre os dados gerados e analisados, em busca da construção de respostas.

4. RESULTADOS

Após análise dos PRs, os analistas perceberam padrões que foram identificados em diversos PRs e em todas as amostras, e esses padrões nos ajudam a responder às questões apresentadas como objetivo:

Q1: Como revisores sugerem refatoramentos em comentários de review em PRs com refatoramento induzido?

Foram percebidas 4 características que definem uma sugestão de refatoramento em RIPRs:

1ª característica:
Revisores sempre apontam o arquivo, e linha do código-fonte, onde querem que alguma alteração seja feita, isso foi percebido em todos os PRs analisados. Um exemplo é o PR de número 4574 do repositório *apache/knox* analisado na rodada 4, no comentário anterior ao *commit* de id *c2ddcb9*. Na Figura 3 é possível perceber o arquivo e a função que deveria sofrer refatoramento.

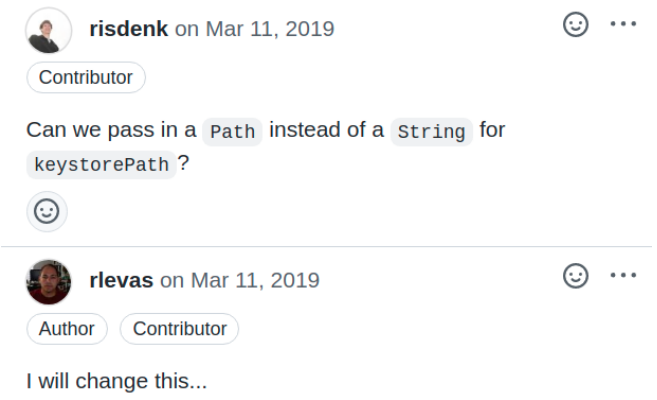

Figura 3. Um exemplo de sugestão de refatoramento. https://github.com/apache/knox/pull/69#discussion_r264268210

2ª característica:
Os comentários podem sugerir diretamente um refatoramento, ou podem sugerir alguma alteração no código que acarrete em um refatoramento. No repositório *apache/dubbo*, em um comentário feito antes do commit *6d1c122* representado na Figura 4, é possível ter um exemplo de uma sugestão de mudança de design de código que acarretou em um refatoramento de forma indireta:

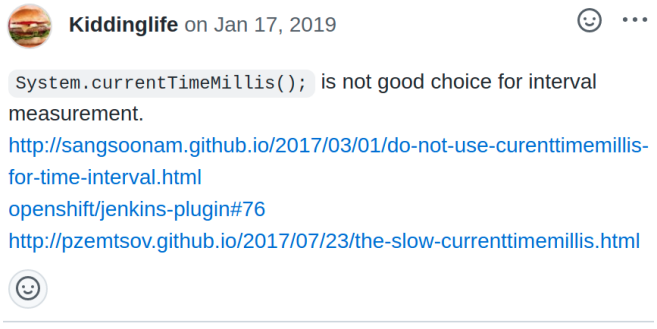

Figura 4. Exemplo de comentário que indiretamente induziu a um refatoramento. https://github.com/apache/dubbo/pull/3174#discussion_r248889145

3ª característica:
Os revisores podem justificar os motivos da sugestão feita, ou não. O mesmo PR pode conter comentários com ambos os tipos de sugestão, como por exemplo o 964 do repositório apache/brooklyn-server, apresentado na Figura 5 e na Figura 6, antes do *commit 49ee11d*, onde o revisor faz um comentário sem justificativa, e em seguida um comentário com justificativas:

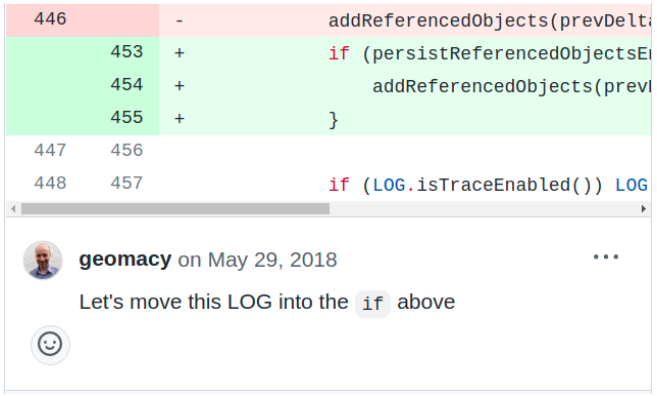

Figura 5. Exemplo de comentários sem justificativas. https://github.com/apache/brooklyn-server/pull/964#pullrequestreview-123932601

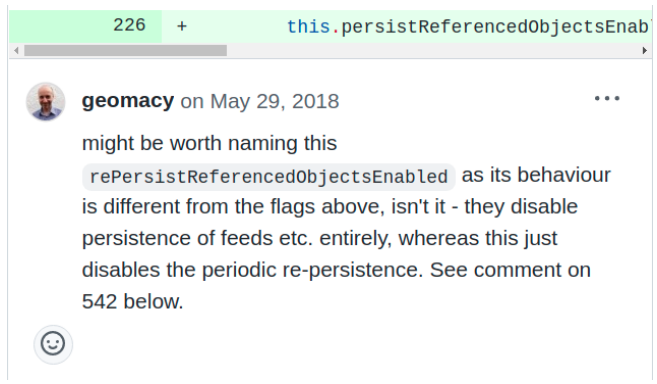

Figura 6. Exemplo de um comentário com justicativas

4º característica:
Os comentários que sugerem diretamente refatoramentos podem acontecer em forma de uma pergunta ao autor com a sugestão de mudanças, mas também pode acontecer com um comentário direto falando exatamente sobre a alteração que deve ser realizada.
A Figura 7 exemplifica comentários feitos por revisores em forma de sugestão para o autor e de forma direta:

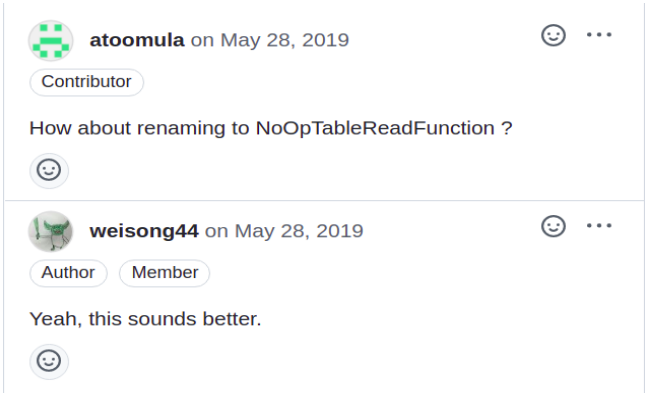

Figura 7. Exemplo de comentário feito em forma de sugestão. https://github.com/apache/samza/pull/1051#discussion_r287917720

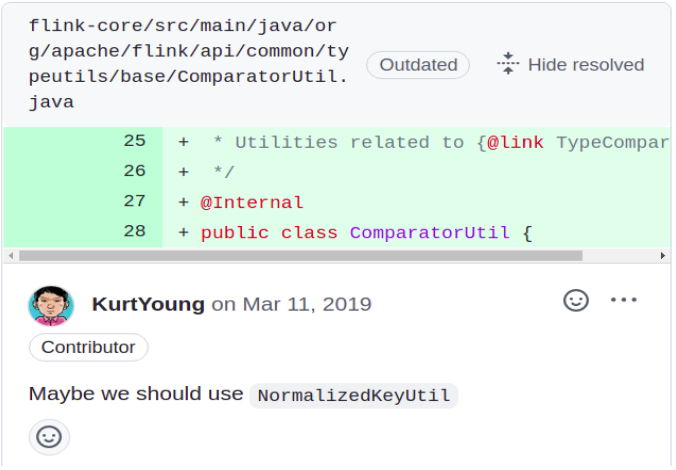

Figura 8. Exemplo de um comentário, no qual o revisor demandou a mudança ao autor. https://github.com/apache/flink/pull/7945#discussion_r264078776

Q2: Os revisores justificam suas razões?

Dentre os 195 commits com refatoramentos de PRs analisados na última rodada, 37 continham comentários com justificativas acerca das sugestões. Vale ressaltar que os commits iniciais de cada PR foram ignorados, tendo em vista que esses commits não possuem comentários anteriores à eles. Os commits com justificativas foram chamados de *rationale* e os que não contém, *not rationale*. A Tabela 1 exemplifica melhor essa relação.

	RIPRs		nonRIPRs	
	rationale	not rationale	rationale	not rationale
Commits	34	100	3	59

Tabela 1. Relação entre RIPRs e nonRIPRs no contexto das justificativas em comentários.

Através da Tabela 1, percebe-se que 25.37% dos *commits* em RIPRs houveram justificativas nos seus comentários, enquanto 4.83% dos *commits* em nonRIPRs houveram justificativas em seus comentários.
Nessa amostra houve uma maior porcentagem de realização de refatoramentos após comentários com justificativas, o que nos faz entender melhor a influência das justificativas nos comentários e abre espaço para novas observações serem feitas, como por exemplo se a presença de justificativa em comentários de sugestão tem maior chance de induzir a refatoramentos.
Também é possível a realização de um estudo que compara estes PRs com diferentes tipos de refatoramentos, já que muitas vezes alguns refatoramentos são óbvios, como por exemplo o comentário feito no repositório *apache/flink* a seguir, onde com apenas um termo comentado o revisor consegue indicar que um refatoramento do tipo *Rename Method* [12] deve ser aplicado:

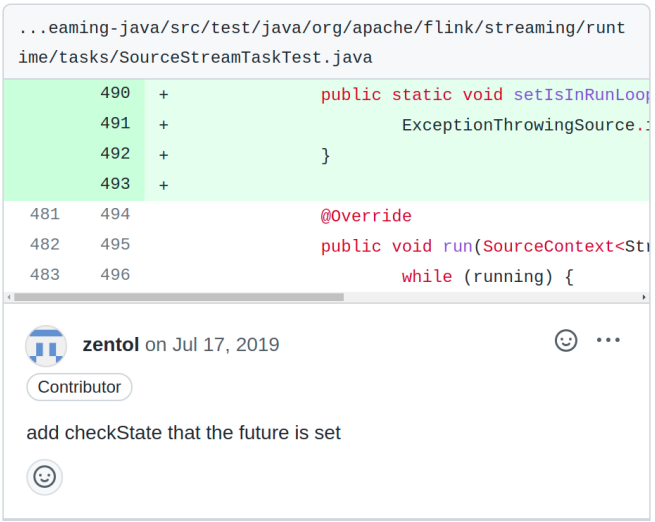

Figura 9. Comentário que não necessita justificativas. https://github.com/apache/flink/pull/9143#discussion_r304347335

Q3: Qual a relação entre recomendações de refatoramento e reais refatoramentos em PRs com refatoramento induzido?

Para entendermos a relação entre recomendações e realizações de refatoramentos em RIPRs, é necessário examinar os resultados obtidos ao analisar nonRIPRs, e com isso entender os motivos de refatoramentos não ocorrem mesmo com comentários. Os analistas definiram 2 motivos de não ocorrerem refatoramentos em nonRIPRs:

1º motivo:
Alguma discussão ocorre no comentário posterior a algum commit e, nessa discussão, o autor argumenta sobre não realizar a mudança sugerida e o revisor decide pela escolha do autor:

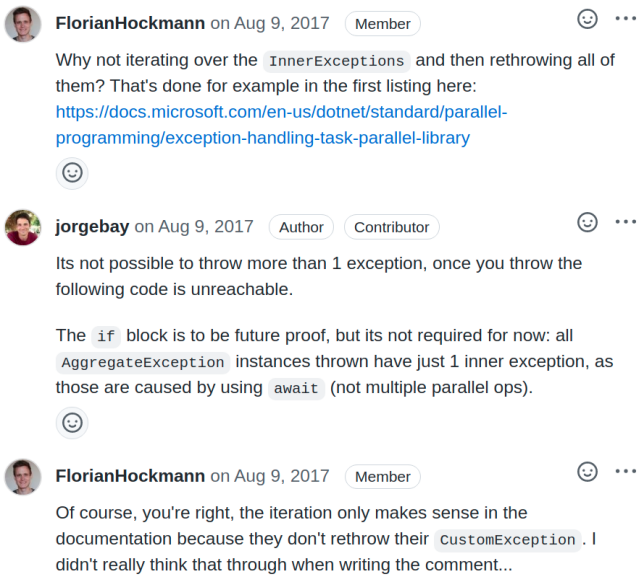

Figura 10. Exemplo de discussão em um nonRIPR. https://github.com/apache/tinkerpop/pull/690#discussion_r132150241

No comentário da Figura 10 é possível entender como uma sugestão que possivelmente acarretaria em um refatoramento não foi adotado pelo autor, após uma discussão.

2º motivo:
Os comentários podem sugerir mudanças que não necessitam de refatoramentos para que aconteçam.
Um exemplo é o PR 8140 [13] extraído do repositório *apache/beam*, onde o revisor realizou uma série de comentários com sugestões de design de código que não necessitam de refatoramentos para serem acatadas pelo autor.
A Tabela 2 representa a quantidade de ocorrências de cada motivo em nonRIPRs:

	Motivo 1	Motivo 2
1ª rodada	2 PRs	8 PRs
2ª rodada	3 PRs	7 PRs
3ª rodada	1 PRs	12 PRs
4ª rodada	3 PRs	23 PRs
Total	9 PRs	49 PRs

Tabela 2. Quantidade de nonRIPRs em que ocorreram os motivos citados.

Na Tabela 2 percebe-se que, em nonRIPRs, existe uma predominância de comentários com sugestões que não acarretam em refatoramentos.
Sendo assim, podemos notar que quando os motivos não ocorrem, um refatoramento é feito no PR após algum comentário.

5. CONCLUSÕES

Neste artigo foi feita uma análise de PRs extraídos de projetos *open-source* escritos na linguagem *Java*. Dentre esses PRs, uma análise foi feita com objetivo de caracterizar, definir e entender PRs que contém comentários que podem induzir a refatoramentos.

A partir disso, respondemos 3 questões:
Como revisores sugerem refatoramentos em comentários de review em RIPRs?
Apontado o local exato no código onde o refatoramento deve ser executado. Esse comentário pode ser direto, sugerindo o refatoramento exato que deve ser realizado, ou pode ser apenas alguma mudança no comportamento do código que cause um refatoramento. Os revisores podem justificar suas razões para a sugestão, como também podem apenas indicar o que deve ser modificado sem explicações. Os comentários também podem ser feitos em forma de pergunta, onde o revisor pergunta educadamente ao autor se ele pode realizar tais mudanças, mas também podem ser feitos de forma retórica, com foco no que deve ser modificado.
Os revisores justificam suas razões?
Dos commits analisados, a menor parte deles continha sugestões com justificativas.
Q3: Qual a relação entre recomendações de refatoramento e reais refatoramentos em RIPRs.
Para que um refatoramento não ocorra, o PR deve conter um comentário com sugestões que não gerem refatoramentos, ou um comentário que gere uma discussão entre o autor e o revisor, e ao ouvir os argumentos do autor, o revisor opta por não realizar tais sugestões.

6. RECONHECIMENTOS

Este trabalho foi possível, primeiramente graças à ferramenta *RefactoringMiner* desenvolvido principalmente por Nikolaos Tsantalis [16] que nos possibilitou coletar todos os dados necessários para análise. Graças também ao trabalho de Flávia Coelho de caracterização [17] e definição [18] de RIPRs, que teve importante contribuição para entendermos como selecionar e analisar os PRs.

7. REFERÊNCIAS

[1] FOWLER, Martin. Refactoring.com. https://refactoring.com/. acessado em 23/05/2021.

[2] Long Method. https://refactoring.guru/pt-br/smells/long-method. acessado em 23/05/2021.

[3] Extract Method. https://refactoring.guru/pt-br/extract-method. acessado em 23/05/2021.

[4]. BRITO, Rodrigo. VALENTE, Marco. “RefDiff4Go: Detecting Refactorings in Go”. https://homepages.dcc.ufmg.br/~mtov/pub/2020-sbcars.pdf. acessado em 23/0

[5] Everything you need to know about Code Smells. https://www.codegrip.tech/productivity/everything-you-need-to-know-about-code-smells/. acessado em 06/10/2021.

[6] About Github. https://github.com/about. acessado em 06/10/2021.

[7] What is a Pull Request in Git?. https://www.gitkraken.com/learn/git/tutorials/what-is-a-pull-request-in-git. acessado em 07/10/2021.

[8] RefactoringMiner. https://github.com/tsantalis/RefactoringMiner. acessado em 07/10/2021.

[9] What Is SHA-1 and How Is It Used for Data Verification?. https://www.lifewire.com/what-is-sha-1-2626011. acessado em 07/10/2021

[10] Nikolaos Tsantalis, Ameya Ketkar, and Danny Dig. RefactoringMiner 2.0. IEEE Transactions on Software Engineering, 2020.

[11] CSV file: Definition. https://support.google.com/google-ads/answer/9004364?hl=en. acessado em 10/10/2021.

[12] Rename Method. https://refactoring.guru/pt-br/rename-method. acessado em 12/10/2021.

[13] [BEAM-6822] Added Kafka Kubernetes cluster config files and setup script #8140. https://github.com/apache/beam/pull/8140. acessado em 12/10/2021.

[14] Sumaira Nazir, Nargis Fatima, Suriayati Chuprat. Modern Code Review Benefits-Primary Findings of A Systematic Literature Review. ICSIM '20: Proceedings of the 3rd International Conference on Software Engineering and Information Management. Janeiro de 2020. Páginas 210–215. https://doi.org/10.1145/3378936.3378954.

[15] DA COSTA DAVILA, Nicole. Modern Code Review: From Foundational Studies to Proposed Approaches and their Evaluation. Fevereiro de 2020. https://lume.ufrgs.br/handle/10183/211266.

[16] Nikolaos Tsantalis, Matin Mansouri, Laleh M. Eshkevari, Davood Mazinanian, Danny Dig. Accurate and Efficient Refactoring Detection in Commit History. 2018. http://users.encs.concordia.ca/~nikolaos/publications/ICSE_2018.pdf.

[17] COELHO, Flávia. Characterizing Refactoring-Inducing Pull Requests. Fevereiro de 2021. http://users.encs.concordia.ca/~nikolaos/publications/ICSE_2018.pdf.

[18] COELHO, Flávia. An Empirical Study on Refactoring-Inducing Pull Requests. Agosto de 2021. http://users.encs.concordia.ca/~nikolaos/publications/ICSE_2018.pdf.

[19] The Apache® Software Foundation projects statistics. https://projects.apache.org/statistics.html. Novembro 2020. acessado em: 13/10/2021.