UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCUS VINICIUS BERTONCELLO

Pesquisa exploratória sobre o perfil e comportamento dos contribuidores casuais no GitHub

Maringá
2019

MARCUS VINICIUS BERTONCELLO

Pesquisa exploratória sobre o perfil e comportamento dos contribuidores casuais no GitHub

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Igor Fabio Steinmacher
Coorientador: Prof. Dr. Edson Alves de Oliveira Junior

Maringá
2019

Bertoncello, Marcus Vinicius
B547p Pesquisa exploratória sobre o perfil e comportamento dos contribuidores casuais no GitHub. / Marcus Vinicius Bertoncello. – Maringá, 2019.
115 f.

Orientador: Prof. Dr. Igor Fabio Steinmacher
Coorientador: Prof. Dr. Edson Alves de Oliveira Junior
Dissertação (Mestrado) – Universidade Estadual de Maringá, Departamento de Informática, Pós-Graduação em Ciência da Computação.

1. Software Livre. 2. GitHub. 3. Contribuidor casual. I. Steinmacher, Igor Fabio. II. Oliveira Junior, Edson Alves de. III. Título.

CDD: 005.3

César Augusto Galvão Fernandes Conde – CRB 9/1747

FOLHA DE APROVAÇÃO

MARCUS VINICIUS BERTONCELLO

Pesquisa exploratória sobre o perfil e comportamento dos contribuidores casuais no GitHub

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Departamento de Informática, Centro de Tecnologia da Universidade Estadual de Maringá, como requisito parcial para obtenção do título de Mestre em Ciência da Computação pela Banca Examinadora composta pelos membros:

BANCA EXAMINADORA

Prof. Dr. Igor Fabio Steinmacher
Universidade Tecnológica Federal do Paraná – DACOM/UTFPR-CM
participação remota

Profa. Dra. Thelma Elita Colanzi Lopes
Universidade Estadual de Maringá – DIN/UEM

Prof. Dr. Wesley Klewerton Guêz Assunção
Universidade Tecnológica Federal do Paraná – COTSI/UTFPR-TD

Aprovada em: 13 de novembro de 2019.
Local da defesa: Sala 101, Bloco C56, *campus* da Universidade Estadual de Maringá.

AGRADECIMENTOS

Agradeço primeiramente a Deus por ter me abençoado nesta caminhada e as pessoas que contribuíram na realização deste trabalho.

Ao meu Orientador Igor Fabio Steinmacher por toda ajuda e paciência que teve comigo durante esta caminhada, este trabalho só foi possível de ser realizado devido a sua orientação e conhecimentos passados durante esse período.

À minha família por todo apoio e carinho que sempre estiveram comigo, me incentivando para sempre melhorar.

As novas amizades que o mestrado na UEM me proporcionou. Se eu cheguei até aqui, foi graças ao apoio incondicional de todos os meus colegas. Em especial agradeço ao meu grande amigo Osvaldo Fernando Cossa, pela amizade e pelas horas de estudos.

Aos meu amigos e colegas da UFPR - *campus* avançado em Jandaia do Sul, pelo suporte durante o meu afastamento. Aos diretores do campus pelo incentivo e apoio para a realização dos meus estudos. Em especial aos meus colegas de unidade, Charles Masaharu Sakai e Fabio Antonio Gabriel. Ao professor Alexandre Prusch Züge por ter dedicado o seu tempo para me auxiliar.

Aos professores do DIN, pelos ensinamentos passados durante esse tempo, em especial a professora Thelma Elita Colanzi Lopes por me permitir realizar o estágio em docência na sua disciplina. Ao professor Edson Alves de Oliveira Junior pelo apoio. À Maria Inês Davanço Laccort por todo carinho e assistência prestada a todos os alunos do programa.

Aos professores Igor Wiese e Gustavo Pinto pelo auxílio na elaboração deste trabalho e nos artigos.

E a todos cujos nomes não foram citados, mas que contribuíram para que este trabalho fosse realizado.

Pesquisa exploratória sobre o perfil e comportamento dos contribuidores casuais no GitHub

RESUMO

O desenvolvimento de software livre está em constante crescimento devido aos seus benefícios, tanto para quem desenvolve quanto para quem o utiliza. Para que um software livre continue evoluindo é necessário o esforço de contribuidores engajados com o seu desenvolvimento e manutenção. Neste trabalho foi explorado um tipo específico de contribuidor, conhecido como contribuidor casual, que é aquele que realiza uma única contribuição no projeto. O objetivo deste trabalho foi explorar o comportamento deste contribuidor dentro do projeto no qual ele fez a sua contribuição. Para alcançar os objetivos do trabalho, a primeira etapa foi analisar dois métodos para selecionar os contribuidores casuais. Além disso, foi analisado a acurácia do método escolhido para este trabalho. Após selecionar os contribuidores casuais foi analisado o relacionamento do mesmo com o projeto, para isso, foi observado o comportamento em relação as redes sociais do GitHub (estrelas, seguidores, observadores e *forks*), e a participação em outras atividades do projeto (comentários, novas tentativas de contribuição e criação de novas *issues*). Pode-se observar que, em geral, os contribuidores casuais não participam do projeto ativamente. Poucos contribuidores participam de outras atividades do projeto, e quando o fazem, realizam as atividades antes de contribuir. Em relação às contribuições realizadas, o contribuidor participa (68,7%) com código-fonte do projeto, e em sua grande maioria utilizando a linguagem de programação C. Por fim, foram utilizados três métodos para agrupar os contribuidores casuais de acordo com as características analisadas. Para isso foram utilizados dois algoritmos de agrupamento *(K-means e Model-based clustering)* e um método manual. Os dois algoritmos não tiveram resultados satisfatórios, levando a utilizar um agrupamento manual dos contribuidores. Foi utilizado o método de regressão logística multinomial para testar o modelo proposto, porém a qualidade de ajuste foi insatisfatória. Os resultados encontrados neste trabalho podem servir como ponto de partida para novas pesquisas sobre o comportamento dos contribuidores casuais, além de aprimorar o estado da arte. A comunidade de Software Livre pode se beneficiar pensando em modos de manter contribuidor interessado no projeto e explorar novos focos de interesses. Outros métodos para elaborar o perfil do contribuidor podem ser elaborados tendo em vista que os métodos propostos neste trabalho não obtiveram resultados satisfatórios.

Palavras-chave: Software Livre. GitHub. Contribuidor casual.

Exploratory research on the profile and behavior of casuals on GitHub

ABSTRACT

Open Source Software development is constantly growing due to the benefits offered by this model. Open Source growth and sustainability requires the effort of contributors engaged in its development and maintenance. This work explores a specific type of contributor, known as a casual contributor, who makes a single contribution to a project and do not return. The purpose of this thesis was to explore the behavior of this contributor within the project in which he made his contribution. To reach the objectives of the work the first step was to analyze two methods to select casual contributors. In addition, the accuracy of the method chosen for this work was analyzed. After selecting the casual contributors, their relationship with the project was analyzed, and their behavior in relation to GitHub's social networks (stars, followers, observers and forks, and participation in other project activities (comments, new attempts to contribute and create new issues It can be seen that, in general, casual contributors do not actively participate in the project. Few contributors participate in other project activities, and when they do, carry out the activities before contributing. contributions made, the contributor participates (68.7%) with project source code, and mostly using the C programming language. Finally, three methods were used to group the casual contributor according to their characteristics analyzed. For this we used two clustering algorithms (K-means and Model-based clustering) and a manual method. Both algorithms did not yield satisfactory results, leading to the use of a manual grouping of contributors. The multinomial logistic regression method was used to test the proposed model, but the quality of fit was unsatisfactory. The results found in this paper can serve as a starting point for further research on the behavior of casual contributors, as well as improving the state of the art of casual contributors. The OSS community can benefit by thinking about ways to keep contributors interested in the project and to explore new focuses of interest. Other methods for profiling the contributor can be elaborated considering that the methods proposed in this paper have not obtained satisfactory results.

Keywords: Open Source. GitHub. Casual conbtributors.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

SL: Software Livre
FLOSS: *Free Libre Open Source Software*
API: *Application Programming Interface*
URL: *Uniform Resource Locator*
PR: *Pull Requests*
JSON: *JavaScript Object Notation*
OSS: *Open Source Software*
SHA: *Secure Hash Algorithm*
UI: *User Interface*

SUMÁRIO

1

Introdução

Um software é considerado livre quando ele obedece a quatro liberdades: o direito de usar; copiar; estudar; distribuir e aperfeiçoar o software (Gnu, 2018). Para que um Software Livre (SL) possa ser modificado ou usado por qualquer pessoa, é importante que tenha seu código-fonte disponibilizado publicamente sob uma licença que esteja de acordo com as liberdades mencionadas. O fenômeno do SL produziu uma impressão positiva, não só da indústria de software, mas também das organizações (Boehmke e Hazen, 2017). Um exemplo disso é o uso de SL nos governos. De acordo com Câmara e Fonseca (2007), o uso de SL pode ajudar países em desenvolvimento a dominar a tecnologia de desenvolvimento de software e permitir aplicações que alavanquem o crescimento local.

Um SL é criado e mantido por uma comunidade de contribuidores motivados, que contribuem coletivamente com o desenvolvimento (Sheoran et al., 2014). Para que a comunidade de SL seja capaz de se manter, ela deve ser capaz de atrair e reter novos contribuidores (Lee et al., 2017). Um exemplo de um SL que atrai e recebe contribuidores de forma positiva é o NodeJS (Haff, 2018), que atualmente conta com mais de 2.000[1] contribuidores fazendo com que o projeto continue a existir de maneira sustentável.

Parte dos contribuidores participam de projetos de SL de forma voluntária e episódica, podendo engajar-se ao projeto a qualquer momento, assim como podem sair sem nenhum tipo de comprometimento (Balestra et al., 2017; Barcomb, 2016). Mesmo sabendo que a sustentabilidade e sobrevivência de projetos de software livre dependem da entrada de novos membros, muitas dificuldades são encontradas para que contribuidores entrem e se mantenham ativos em um projeto (Steinmacher et al., 2014). Muitos novatos em um

[1] `https://tableless.com.br/nodejs-umbler`

projeto encontram barreiras para fazer a primeira contribuição, o que os leva algumas vezes, a abandonar o projeto. Essas barreiras são forças contrárias à motivação que os novatos possuem para se engajar em projetos de SL, como afirma Hannebauer et al. (2017).

Sabe-se ainda que, em média, os contribuidores de SL deixam de contribuir para um projeto após 1 ano (Shah, 2006), porém muitos contribuidores deixam o projeto antes desse período. Uma possível explicação para isso é que, de acordo com Zhou e Mockus (2015), a motivação inicial de um contribuidor afeta significantemente a chance dele permanecer no projeto a longo prazo. O que significa que nem todos os contribuidores tem a intenção de entrar no projeto e permanecer por muito tempo. Nesse contexto, pesquisas recentes (Lee et al., 2017; Pinto et al., 2016; Ramos et al., 2015) têm analisado um fenômeno cada vez mais frequente em projetos de SL: os contribuidores casuais. Contribuidores casuais são aqueles que não tem interesse em se tornar membros de longo prazo no projeto, contribuem uma única vez para o projeto e não realizam novas contribuições (Pinto et al., 2016). Lee et al. (2017) afirmam que, em muitos casos, estes contribuidores não se tornam membros ativos na comunidade pois encontram muitas barreiras para contribuir mais vezes.

Ainda com relação aos contribuidores casuais, Barcomb (2016) afirma que pouco se sabe sobre esses contribuidores, como por exemplo, a forma como eles são vistos dentro da comunidade, quais contribuições eles realizam, como eles são gerenciados e quão comum eles são. De acordo com Pham et al. (2013), a razão para o crescimento deste tipo de comportamento foi o surgimento e consolidação do modelo de desenvolvimento *"pull-based"*, introduzido por ambientes como GitHub e GitLab. Neste tipo de modelo, o usuário faz uma solicitação de alteração do código para os membros do projeto por meio de um *"pull request"* (PR). Este modelo facilita a contribuição em projetos de software, por reduzir diversas barreiras para entrada de novatos, pois simplifica e unifica o processo de contribuição (Pinto et al., 2016).

Tendo em vista a necessidade de novos contribuidores para a sustentabilidade de projetos de SL, o fenômeno dos contribuidores casuais, e o contexto social em que estão inseridos, o objetivo deste trabalho é *explorar o comportamento de contribuidores casuais na plataforma do GitHub*[2]*, visando identificar características comuns dos contribuidores casuais*. Para isso, foram analisados os pull requests realizados por contribuidores casuais, a participação em outras tarefas no projeto, bem como a sua rede de relacionamentos no contexto da plataforma.

[2] `http://www.github.com`

Para atingir o objetivo do trabalho, os dados do GitHub foram coletados por meio de sua API pública[3] em conjunto com o conjunto de dados disponibilizado pelo GHTorrent (Gousios, 2013), incluindo informações de contribuições, projetos e contribuidores que se encaixam no perfil de casual: aqueles que possuem apenas uma contribuição aceita em um projeto (Pinto et al., 2016), por heurísticas ou por atribuição da própria plataforma.

Compreendendo as características de contribuidores casuais torna-se possível beneficiar as comunidades, apontando, por exemplo, o que esperar em termos de comprometimento e tipo de contribuição. Este trabalho ajuda ainda a entender a importância desse tipo de contribuidor em um projeto e de suas contribuições. Isto pode apoiar a criação de novas formas de interação de projetos e contribuidores, o que pode vir a desencadear maneiras diferentes de atrair e reter contribuidores, bem como levar a novas técnicas de manutenção e evolução de projetos de SL.

1.1 Contribuições

Neste trabalho analisou-se o comportamento dos contribuidores casuais na plataforma do GitHub. As contribuições deste trabalho incluem:

- **Definição de contribuidores casuais utilizando heurísticas**: foram comparados dois métodos para selecionar contribuidores casuais, um método aplicando heurísticas de desambiguação de PRs (Gousios et al., 2014) e outro método já conhecido e estudado (Pinto et al., 2016). Os dois métodos foram comparados para saber qual obteve os melhores resultados nos projetos selecionados neste trabalho. Utilizando PRs é possível encontrar um número menor de falso-negativos de contribuidores casuais em relação aos *commits*.

- **Análise das heurísticas propostas por Gousios et al. (2014)**: As heurísticas foram analisadas com objetivo de saber sua eficácia na identificação de contribuidores casuais, evidenciando as vantagens e desvantagens. Embora as heurísticas apresentem falhas, a combinação delas aumenta a acurácia do método para identificação de PRs aceitos e não aceitos.

- **Relacionamento dos contribuidores casuais com o projeto ao qual eles contribuíram**: Foi analisado o comportamento do contribuidor casual em 3 redes sociais do GitHub (estrelas, observações e seguir membros/donos do projeto). Observou-se também como o contribuidor se comporta com a cópia do projeto

[3] `https://developer.github.com/v3/`

(*fork*) utilizada para a contribuição (se ele manteve alterações na cópia do projeto). Analisou-se também a criação de novas *issues* no projeto, comentários em outras *issues* ou PRs e se o contribuidor casual tentou realizar novas contribuições (PR). Todas as análises foram realizadas em relação ao período da contribuição, assim, analisou-se o comportamento do contribuidor antes e após seu PR ser aceito no projeto. O contribuidor casual pouco se envolve com o projeto onde realiza a sua contribuição.

- **Especificação do tipo de contribuição**: Foram analisados os arquivos modificados pelas contribuições dos contribuidores casuais. Os arquivos foram categorizados de acordo com as heurísticas propostas por Vasilescu et al. (2014). Os contribuidores casuais colaboram com o projeto em sua maioria, com modificações em arquivos de código-fonte. Além disso descobriu-se que a linguagem mais utilizada por eles é C++ seguido por Javascript.

- **Métodos para analisar o perfil do contribuidor casual**: Foram utilizados 3 métodos de agrupamento para definir o perfil do contribuidor casual. Os métodos foram baseados em relação às características das contribuição analisadas neste trabalho. Dois métodos foram baseados em algoritmos de clusterização (*K-means* e *Model-based clustering*) e outro método foi reproduzido manualmente. Dos três métodos utilizados, nenhum deles se mostrou efetivo ao separar os contribuidores.

1.1.1 Outras contribuições

Além das contribuições citadas anteriormente, foram desenvolvidas algumas ferramentas que auxiliam a mineração de dados do GitHub.

- Desenvolvimento de *scripts* utilizando a linguagem de programação Python para consumir a API pública do GitHub, que está disponível para download no endereço `https://github.com/markaumvb/casuais`

- Dump da estrutura do banco de Dados com tabelas e visões.

2

Revisão da Literatura

Este capítulo aborda o que já se sabe sobre contribuidores casuais e o contexto no qual eles se encaixam, o capítulo está dividido como segue: a seção 2.1 aborda o que é um software livre bem como suas vantagens. A seção 2.2 descreve as motivações e as barreiras encontradas por novatos para contribuir em projetos de SL. A seção 2.3 descreve como a plataforma do GitHub e o modo de contribuidor conhecido como *pull-based* ajuda os projetos a terem mais contribuições e visibilidade e por fim, a seção 2.4 aborda o que já se sabe na literatura sobre os contribuidores casuais.

2.1 Software Livre

Um software para ser considerado livre deve ser disponibilizado sob uma licença que permita a inspeção, uso, modificação e redistribuição do código-fonte do software (Crowston et al., 2012). Para o desenvolvimento de um software livre é importante que ele esteja disponível a qualquer pessoa que tenha interesse em ajudar ou apenas usar o software. O uso de SL ajuda países como o Brasil a ter reduções em custos de softwares proprietários e ganhar autonomia e controle sobre seus softwares (Torvalds, 2018).

O SL está passando por um período de renascimento, devido ao surgimento de plataformas e fluxos de trabalhos modernos para desenvolver e manter o projeto. Como resultado, desenvolvedores estão criando softwares em código aberto mais rápido do que antes (Coelho e Valente, 2017). Atualmente, muitos usuários finais utilizam software livre, como é o caso do navegador Mozilla Firefox, sendo o segundo mais utilizado no Brasil no ano de 2017 (Statcounter, 2018). Seguindo este modelo de desenvolvimento aberto,

empresas como a Microsoft estão se beneficiando e criando projetos abertos para que a comunidade possa contribuir. Atualmente a Microsoft conta com mais de 1.000 projetos abertos e hospedados no GitHub[1].

Além da Microsoft, outras empresas de software conhecidas por seus produtos de software de código fechado estão começando a divulgar parte do seu trabalho usando o SL e também promovendo tal atividade. Por exemplo, em uma pesquisa recente, Pinto et al. (2018) afirmam que nos últimos anos a Google lançou mais de 900 projetos de código aberto. Além da Google, outras empresas como a Apple lançaram alguns de seus produtos com o código fonte aberto. Um exemplo disso é o Swift[2] que é uma linguagem de programação de sua plataforma móvel (Pinto et al., 2018). Ainda de acordo com Pinto et al. (2018), os benefícios de fornecer o código-fonte de um software fechado são, promover contribuições externas, criar ideias novas e possivelmente aumentar o ritmo de mudança.

O desenvolvimento de SL tem a capacidade de competir com sucesso e, talvez, em muitos casos, substituir métodos de desenvolvimento comercial tradicionais (Mockus et al., 2000). De acordo com Mockus et al. (2000), as principais diferenças em desenvolver um software livre são: SL são desenvolvidos por vários contribuidores voluntários; voluntários são livres para escolher qual trabalho vão realizar; não existe um *design* ou projeto detalhado a nível do sistema e não existem planos nem cronogramas (prazos de entrega) dos projetos. Além de todas essas diferenças existe ainda a distribuição geográfica dos contribuidores, onde os desenvolvedores moram em lugares distantes e dificilmente terão contatos face-a-face. Os autores ainda argumentam que mesmo com essa distância entre os colaboradores do projetos, os resultados são equivalentes ou superiores aos outros tipos de software desenvolvidos de forma mais tradicional. Ainda sobre os benefícios do SL, Mockus et al. (2000) alegam que, por exemplo, os defeitos são encontrados e corrigidos muito rapidamente porque existem "muitos globos oculares" procurando pelos problemas. O código é escrito com mais cuidado e criatividade, porque os desenvolvedores estão trabalhando apenas em coisas para as quais eles têm uma verdadeira paixão.

Os potenciais benefícios do SL, se alcançados, não vem sem custos (Pinto et al., 2018). Abrir o código-fonte significa criar, manter e promover uma comunidade em torno da tecnologia; por exemplo, atrair novos desenvolvedores que estão ansiosos para implementar novos recursos (Alexander Hars, 2002; Dabbish et al., 2012; Pinto et al., 2018; Wang e Sarma, 2011). Quando uma empresa de software abre um projeto de software, a equipe de desenvolvimento de software pode enfrentar sobrecarga adicional devido aos custos não triviais de: (1) refatorar a base de código para ser compreensível; (2) criação de

[1] `https://github.com/Microsoft`
[2] `https://github.com/apple/swift`

um site de desenvolvimento e canais de comunicação; e (3) escrever documentação para recém-chegados (Fogel, 2005; Pinto et al., 2018).

2.2 Novatos em Software Livre

Grande parte dos projetos de software livre dependem da comunidade ou são baseados em comunidades Steinmacher (2015). Estes projetos, assim como outras atividades de desenvolvimento de software, são atividades colaborativas Vasilescu (2014). Os participantes podem ter motivações intrínsecas como por exemplo altruísmo e a diversão, ou motivações extrínsecas, como por exemplo melhorar sua habilidade ou melhorar o seu status para participar de uma equipe de desenvolvimento. Em seu estudo, Lee et al. (2017) descobriram que as motivações são baseadas em necessidades extrínsecas para a maioria dos contribuidores. Para que um SL sobreviva e tenha sucesso, as comunidades precisam motivar e reter novos contribuidores para que continue sustentável (Balali et al., 2018). Entretanto, um contribuidor pode se engajar em um projeto de SL sem ter nenhum comprometimento de ajudar com o desenvolvimento do projeto, assim como, também pode abandonar o projeto sem motivos (Balestra et al., 2017; Barcomb, 2016).

Steinmacher et al. (2014) observaram que existem quatro forças diferentes para que um contribuidor se junte a um projeto. Enquanto as forças de motivação e de atratividade, são forças que atraem um desconhecido para um projeto, outras forças como barreiras e retenção influenciam na permanência do projeto. Antes de um contribuidor realizar uma contribuição com um projeto, ele precisa aprender aspectos sociais e técnicos (Steinmacher et al., 2014). Durante esse tempo de aprendizado os novatos podem encontrar barreiras que fazem com que abandonem o projeto. Em sua pesquisa, Steinmacher (2015) identificou 58 barreiras que dificultam essa contribuição, como por exemplo, a dificuldade de configurar um ambiente de produção, falta de conhecimento técnico, interação social com outros membros do projeto, etc. Essas barreiras identificadas foram divididas em 6 categorias: diferenças culturais, características dos novatos, orientações, obstáculos técnicos, problemas de documentação e recepção de questões.

Mesmo encontrando essas barreiras, muitos contribuidores perseveram e conseguem ter o sucesso em contribuir em um projeto de SL (Lee et al., 2017). A literatura mostra que grande parte dos contribuidores que iniciam em um projeto de SL começam com a correção de erros (Lee et al., 2017). A correção de erro é um dos motivos mais comuns que atraem novos contribuidores para o desenvolvimento de SL. Esta motivação está relacionada diretamente à necessidade pessoal do usuário que está sendo afetado pelo erro a ser corrigido.

Em sua pesquisa Lee et al. (2017) investigaram o porquê de muitos contribuidores não permanecerem contribuindo com um projeto, descobriram que um dos principais motivos é a falta de tempo, seguido pelo motivo de que não existe mais a necessidade de contribuir. Além disso, Steinmacher et al. (2018) dizem que muitos novatos em um projeto enviam suas contribuições que não são incorporadas ao projeto e acabam desistindo.

Algumas das barreiras encontradas foram mapeadas por (Hannebauer et al., 2011), que apresentou potenciais soluções às barreiras em uma linguagem de padrão de projetos. Esses padrões ajudam a diminuir as barreiras de contribuição encontradas por Steinmacher (2015). Um exemplo de um padrão criado chama-se *Bazaar Architecture*, que sugere que a arquitetura do software seja desenvolvida em uma hierarquia plana de módulos. Nesse padrão modularizado os novatos podem adicionar novas características no sistema com menos tempo, pois eles devem apenas aprender a interface na qual eles irão implementar novos recursos.

De acordo com Steinmacher (2015), os contribuidores mencionaram que encontram poucas barreiras ao contribuir em projetos que estão hospedados no GitHub, isso se deve ao paradigma de codificação social introduzido pelo GitHub, que oferece uma série de recursos sociais.

2.3 Ambiente social de codificação

Tradicionalmente, as comunidades de SL são apenas baseadas em aspectos técnicos, sem a devida ênfase ao aspecto social do desenvolvimento de software. Assim, para que um contribuidor pudesse contribuir com um SL ele precisava monitorar uma lista de e-mails para assim então depois realizar sua contribuição (Dabbish et al., 2012), seguindo processos que, por muitas vezes, era diferentes entre os projetos. Quando as tecnologias de computação social são usadas em um contexto de desenvolvimento de software, há uma oportunidade para alavancar redes sociais articuladas e observar atividades relacionadas a códigos (Dabbish et al., 2012).

Com o crescimento de redes sociais e novas formas de controlar versões em sistemas distribuídos, muitos SL passaram a ter uma transparência maior para os contribuidores (Dabbish et al., 2012). Com as redes sociais os contribuidores puderam ter um relacionamento mais próximo, facilitando a interação. Para os novatos que procuram projetos para contribuir, é possível investigar as contribuições realizadas anteriormente, contatar os contribuidores por meio de discussões nas tarefas em um ambiente mais transparente.

A transparência do ambiente e a visibilidade das relações dos usuários podem também ter um efeito importante na decisão das contribuições (Dabbish et al., 2012). Esses

ambientes, chamados de plataformas de codificação social, expõem as relações dos contribuidores com outros e também informações sobre o projeto, assim como algumas ferramentas que auxiliam no gerenciamento do projeto como por exemplo o rastreamento de erros.

Existem várias plataformas de codificação social, como *Bitbucket*[3], *Gitlab*[4] etc. Porém, foi com o surgimento do GitHub que o crescimento foi maior e mais rápido (Metz, 2015). O Google que há alguns anos tinha seu próprio ambiente de codificação social (o Google Code), mantém projetos hospedados no GitHub assim como outras grandes empresas. O crescimento da plataforma se deve a facilidade que contribuidores encontram para colaborar com os projetos.

O GitHub é uma plataforma social de codificação que permite hospedar projetos de forma pública, facilitando a exposição do projeto e do seu código-fonte para qualquer pessoa que se cadastre em seu site. Atualmente em números o GitHub possui mais de 100 milhões de projetos hospedados (GitHub, 2018b). Para Pinto et al. (2018) esses ambientes de codificação social não estão apenas diminuindo as barreiras que os contribuidores encontram, mas também estão fazendo com que as contribuições em SL estão se tornando mais visíveis.

O GitHub oferece também uma série de possibilidades de se conectar socialmente, como por exemplo por meio de *watchers*, *follows* e *forks* (Yu et al., 2014). Essas redes fazem com que os contribuidores possam observar as atividades de um projeto assim como os membros do projeto. Sheoran et al. (2014) afirmam que os contribuidores que entram em uma rede de observação do GitHub tendem a permanecer por mais tempo no projeto do que os que não participam. Também afirma que uma das primeiras redes na qual o desenvolvedor entra é a rede de observadores (*watchers*), no qual o contribuidor apenas observa o projeto, sendo notificado de todas atividades envolvidas no projeto.

Muitos usuários que entram em uma rede social do GitHub tendem a seguir as pessoas com grande reputação, como é o caso de Linus Torvalds, que tem mais de 13.000 usuários que o seguem no GitHub (Yu et al., 2014). Além deste comportamento de seguir um líder, os usuários têm o comportamento de rebanho que significa que estão no GitHub porque estão cercados de contribuidores que falam sobre o GitHub (Yu et al., 2014). Algumas pesquisas apontam que a quantidade de contribuidores observando o projeto através da rede de observação, mede a qualidade de um software (Dabbish et al., 2012; Sheoran et al., 2014). Em seu trabalho, Borges et al. (2016) observou que os repositórios hospedados

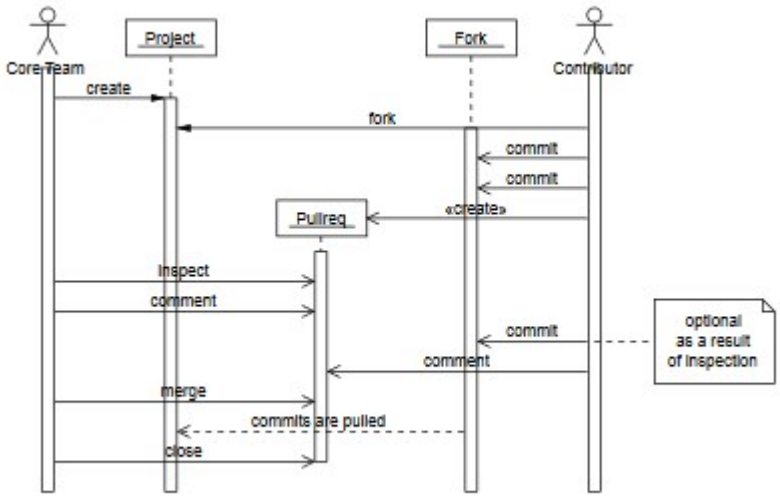
[3] `https://bitbucket.org/`
[4] `https://about.gitlab.com/`

no GitHub por organizações são mais populares do que aqueles que são hospedados por indivíduos.

2.3.1 Desenvolvimento baseado em pulls

Com essa crescente comunidade de SL, a forma de contribuir com um projeto tem se tornado mais fácil, graças ao modelo de desenvolvimento "*pull-based*" (ou baseado em *pulls*)(Pham et al., 2013), introduzido por ambientes como *GitHub* e *GitLab*. Neste modelo de contribuição, o contribuidor faz um *fork* (clone) do projeto, tendo em suas mãos o projeto na íntegra para poder modificar. O usuário então submete suas alterações em forma de *pull-requests*, onde o contribuidor envia uma solicitação de junção (*merge*) do seu código modificado com o código original do projeto. As alterações podem passar por revisões ou sugestões de melhorias antes de serem integradas ao projeto. Qualquer usuário do GitHub que tenha uma conta pode realizar comentários e revisões de código nas submissões de alteração. Em alguns casos o contribuidor pode realizar novas alterações em um *pull-request* caso seja necessário, assim como outros membros também podem interagir sugerindo modificações no código-fonte enviado (ver Figura - 2.1).

Figura 2.1: Diagrama de sequência para realizar uma contribuição em um projeto



Fonte: Gousios (2014)

Muitos SL estão migrando da sua forma antiga de contribuição (lista de e-mails) para o modelo *pull-based* (Gousios et al., 2014). Esse modelo possibilita maior exposição, potencialmente aumentando o número de contribuidores (Gousios et al., 2014; Pham et

al., 2013). Gousios et al. (2016) observaram que em janeiro de 2016, 135.000 projetos hospedados no GitHub receberam mais de 600.000 *pull requests*.

Rebouças et al. (2017) afirmam que essas ferramentas, além de ajudar no controle de versão do sistema, também ajudam em outros processos como por exemplo, a abertura de chamados, facilitando o processo de criar, manter e contribuir com o software. Além disso, esse modelo facilita a forma com a qual os contribuidores podem discutir sobre as mudanças a serem aplicadas no projeto, através de discussões e revisões de código, antes de aceitar ou rejeitar a contribuição (Gousios et al., 2014; Pham et al., 2013).

2.4 Contribuidores casuais em Software livre

De acordo com Barcomb (2016) contribuidores casuais também são conhecidos na literatura como voluntários episódicos na comunidade de projetos *FLOSS*. Esses contribuidores são aqueles que contribuem de alguma forma no projeto, mas não se mantêm por muito tempo, pois preferem ter tarefas de curto prazo (Bryen e Madden, 2006). Esse "novo" fenômeno (voluntários episódicos), já era conhecido, mas voltou a ser discutido recentemente, possivelmente devido às vidas cada vez mais agitadas das pessoas e ao profissionalismo da força de trabalho sem fins lucrativos (Bryen e Madden, 2006). De acordo com Barcomb (2016) as organizações estão se adaptando para fazer o uso desses contribuidores, porém ainda não se tem muitas informações de como esses contribuidores podem ser gerenciados e inspirados dentro do contexto de SL.

Em sua pesquisa de Gousios et al. (2014), onde afirmaram que 7% dos *pull requests* feitos no GitHub em 2012 foram feitos por contribuidores casuais, e também 3,5% dos "*forks*" realizados são com a intenção de realizar uma única contribuição para o projeto. Pham et al. (2013) mencionaram que o custo para fazer um *fork* de um projeto no GitHub é insignificante (54% dos repositórios são *forks*) pois é comum para os desenvolvedores usar os repositórios para executar apenas *commits* casuais.

As contribuições realizadas por contribuidores casuais são feitas, em geral, por motivações extrínsecas (Lee et al., 2017; Pinto et al., 2016). De acordo com Lee et al. (2017) as contribuições realizadas por contribuidores periféricos ao projeto se dão por motivos que incluem motivos profissionais, como por exemplo o uso da ferramenta para o seu trabalho ou serem pagos por algum empregador, ou outros motivos como por exemplo aumentar sua reputação ou desenvolver novas características para o projeto. Pinto et al. (2016) também apresentam este tipo de motivação como a mais recorrente para este tipo de contribuidor.

As contribuições que os contribuidores realizam em projetos envolvem qualquer tipo de mudança de arquivo, seja adicionar um novo arquivo, remover um arquivo, ou diretamente no arquivo do código-fonte. Lu et al. (2017) analisaram a qualidade do código feita por contribuidores casuais com outros tipos de contribuidores, em diferentes projetos. Os contribuidores que em um determinado projeto participaram de forma casual, mas em outros projetos participaram de uma forma mais ativa, possuindo mais de uma única contribuição aceita, neste caso não houve nenhuma diferença estatisticamente significante em seus códigos, utilizando a métrica de qualidade de código. Além disso os autores notaram que os contribuidores casuais que possuem poucos projetos com marcação de estrelas[5] introduziram mais CQI - (*Code Quality Issue*) nos projetos, do que aqueles que possuem mais projetos com estrelas.

Para Zhou e Mockus (2015) para que um projeto SL seja mantido e tenha sucesso, o projeto precisa de contribuidores capazes de executar qualquer tarefa demandada, porém algumas tarefas exigem contribuidores experientes. As contribuições de um contribuidor casual podem ser tão importantes quanto a de um contribuidor regular (não casual). Contribuidores regulares estão participando ativamente do projeto, e definindo os rumos no qual o projeto irá seguir, e são capazes de realizar tarefas mais complexas, por contribuírem mais de uma vez com o projeto.

Na pesquisa realizada por Pinto et al. (2016) analisaram a presença do contribuidor casual em 275 projetos hospedados no GitHub, em um dos projetos analisados, como o projeto `django`[6] (10 anos de histórico de código) possui um total de 61,57% de contribuidores casuais em seu projeto, enquanto projetos mais antigos como por exemplo o `Linux`[7] com mais de 20 anos de histórico de código, possui 39,28%. Porém os pesquisadores descobriram que esse valor não reflete na quantidade de contribuições, na realidade 1,73% das contribuições dos projetos observados são realizadas por casuais. Ainda de acordo com Pinto et al. (2016) as contribuições realizadas por contribuidores casuais estão longes de ser triviais. Concluíram que os contribuidores casuais contribuem com correção de erros (30,2%), adicionam novos recursos (18,75%) e também refatoram o código (8,85%). Os casuais, embora contribuam com uma pequena parte do projeto (1,73% dos *commits*), trazem benefícios reconhecidos pelos mantenedores.

Em resumo, o contribuidor casual tem um importante papel no desenvolvimento de SL, o perfil de um contribuidor casual pode mudar o modo de contribuir. Um caso de sucesso é o `NodeJS`[8] que aceita todos os tipos de contribuições, sendo que a maior parte

[5] `https://help.github.com/en/articles/saving-repositories-with-stars`
[6] `https://github.com/django/django`
[7] `https://github.com/torvalds/linux`
[8] `https://nodejs.org/en/`

do código-fonte é escrita por contribuidores casuais (Haff, 2018), e muito ainda se precisa entender sobre o perfil desse tipo de contribuidor e suas contribuições, a fim de beneficiar contribuidores e comunidades SL.

3

Metodologia de Pesquisa

O Objetivo deste trabalho é *explorar o comportamento de contribuidores casuais na plataforma GitHub*. O GitHub foi escolhido neste trabalho por ser na atualidade a plataforma mais conhecida e utilizada pela comunidade de software livre. O trabalho visa analisar o tipo de relacionamento que os contribuidores casuais têm com os projetos. Até o momento algumas pesquisas desenvolvidas sobre o assunto tratam de forma superficial os contribuidores casuais (Lee et al., 2017; Pinto et al., 2016; Ramos et al., 2015). Este estudo sobre contribuidores casuais ajuda a entender a forma com que eles contribuem com o software, até o momento não se sabe como eles se comportam na comunidade de software livre.

Para guiar a pesquisa em direção ao objetivo proposto, foram definidas questões de pesquisas mais específicas a serem respondidas por este trabalho:

QP1: Como se compara o uso de *commits* e *pull requests* como unidade de análise na identificação de contribuidores casuais?

O primeiro passo deste trabalho foi realizar uma comparação entre dois métodos para encontrar os contribuidores casuais. A comparação foi necessária para determinar qual método de selecionar os contribuidores casuais é mais eficiente. Foram analisados os modelos propostos por Pinto et al. (2016) que seleciona os contribuidores pelo número de *commits* e (Gousios et al., 2014) que utiliza o PR e algumas heurísticas para fazer tal seleção. Além disso, as heurísticas foram comparadas para saber a acurácia, combinando-as ou de modo exclusivo. Os resultados relativos a esta questão estão na seção 4.3.4.

QP2: Como é o relacionamento dos contribuidores casuais com os projetos com os quais eles contribuem?

O relacionamento entre o contribuidor casual e o projeto foi analisado através da interação do contribuidor casual com as redes sociais da plataforma: seguir membros, marcar projeto com estrela, observar o projeto e realizar o *fork* do projeto. Também foi analisado se o contribuidor participou de outras atividades do projeto como: criar novas *issues*, tentar novas contribuições através de novos PRs, comentar em *issues* ou *pull requests* e participar de comentários em outros PRs. Por fim, o tipo de contribuição realizada pelo contribuidor, utilizando a classificação dos arquivos sugeridas por Vasilescu et al. (2014). Os resultados relativos a esta questão estão na seção 5.4.

QP3: Qual o perfil do contribuidor casual através da sua interação com o projeto?

Agrupar os contribuidores casuais com base nas características analisadas utilizando algoritmos de *clustering* e manualmente. Os algoritmos utilizados foram o *Model-based clustering* em seguida o *K-means*. Manualmente os contribuidores foram separados em 4 grupos baseado nas tentativas de outras contribuições antes ou após o PR aceito (*Antes*: tentaram contribuir somente antes; *Depois*: tentaram somente após; *Nenhum*: só realizaram uma única contribuição; e *Antes/Depois*: tentaram contribuir antes e após o PR aceito). Para avaliar a eficácia do modelo proposto, foi aplicado a técnica da regressão logística multinomial. Os resultados relativos a esta questão estão na seção 5.4.

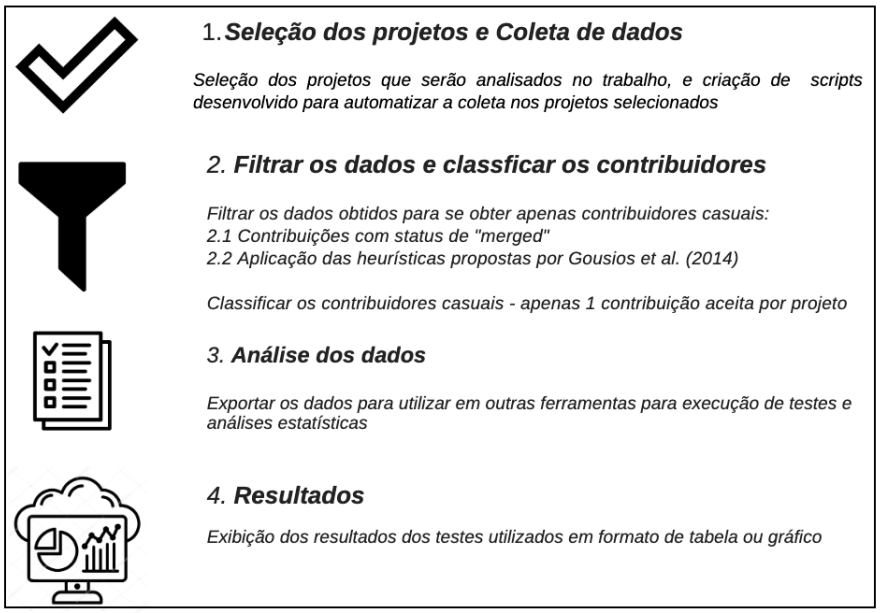
3.1 Metodologia

A metodologia utilizada neste trabalho é baseada no método utilizado por (Ray et al., 2017) e (Pinto et al., 2016). A metodologia proposta é explicada detalhadamente na Figura - 3.1.

A primeira etapa foi selecionar quais são os projetos que podem ser utilizados como objeto de estudo para essa pesquisa. Os projetos selecionados foram escolhidos pela sua popularidade, de acordo com Ray et al. (2017). A popularidade de um projeto é medida através da quantidade de estrelas (*stars*) que um projeto possui no GitHub (GitHub, 2019g). Foram escolhidas 16 linguagens de programação para se ter uma amostra diversificada de projetos (C, C++, Clojure, CoffeeScript, Erlang, Go, Haskell, Java, JavaScript, Objective-C, Perl, PHP, Python, Ruby, Scala, e TypeScript). Os projetos escolhidos de cada linguagem de programação foram selecionados pelo critério de quantidade de estrelas dentro da plataforma do GitHub.

Para cada linguagem de programação, foram selecionados 20 projetos, totalizando assim 320 projetos de software, porém alguns projetos não foram coletados devido a mudança do dono do repositório e/ou mudança do nome do projeto.

Figura 3.1: Etapas da metodologia



Algumas restrições foram impostas na seleção dos projetos, como a exclusão de linguagens que não são consideradas como linguagem de programação, como por exemplo CSS e HTML (Ray et al., 2017). Após a exclusão de projetos escritos nestas linguagens, foram realizados filtros para os projetos selecionados:

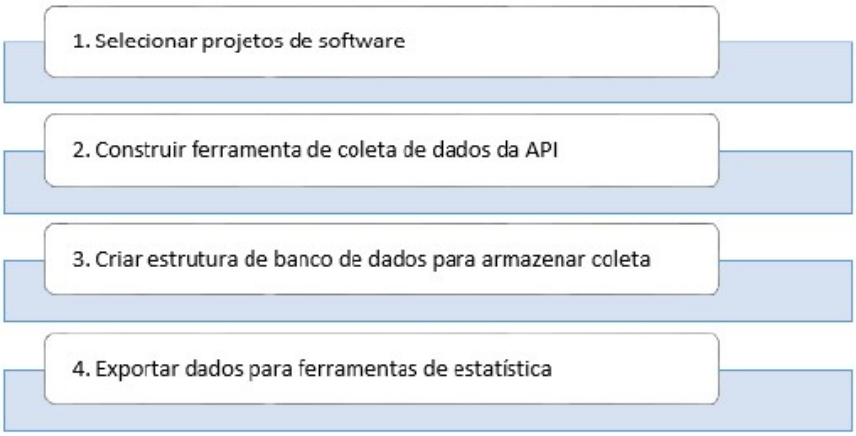
- poucos *commits* - projetos com menos de 28 *commits*, onde 28 é o primeiro quartil da quantidade de *commits* de todos os projetos;
- não serem projetos de software - alguns projetos hospedados no GitHub são livros didáticos, como por exemplo `clojure-cookbook`[1], ou *bookmarks*. A importância deste trabalho é manter o foco em projetos que são softwares, uma característica importante da linguagem de programação e processo de contribuição pode influenciar não só a codificação processo, mas também o envolvimento da comunidade Pinto et al. (2016);
- projetos que não são colaborativos - projetos que são mantidos apenas por um contribuidor.

3.1.1 Coleta de dados

A coleta de dados foi dividida em 4 etapas, conforme apresentado na Figura - 3.2.

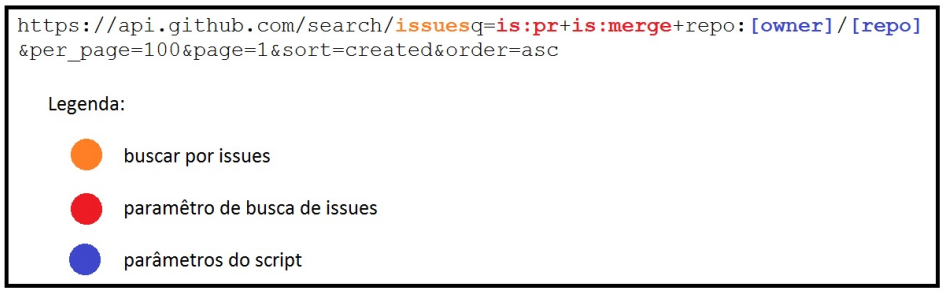
[1] `https://github.com/clojure-cookbook/clojure-cookbook`

Figura 3.2: Etapas da coleta de dados



Na plataforma de desenvolvimento do GitHub[2] está disponível uma API pública, onde é possível obter várias informações sobre projetos, usuários, contribuições, etc. Foi utilizada a API para obter os dados das contribuições realizadas nos projetos propostos. Sendo assim foram criadas URLs[3] para se obter todos os *pull requests* dos projetos. Uma das principais fontes utilizadas nos estudos é o *pull request* (PR) (Cosentino et al., 2016). Os contribuidores podem criar e enviar um conjunto de alterações para um projeto de software por meio de PRs. Cada PR pode ser discutido por qualquer usuário do GitHub, após isso ele pode ser aceito ou rejeitado pelos mantenedores do projeto de software (Gousios et al., 2014; Pham et al., 2013). Quando um PR é aceito por um membro ele ganha o status de *merged*. Esse status garante que o PR foi aceito e incorporado ao repositório do projeto.

Figura 3.3: URL utilizada para obter os *pull requests* através da API do GitHub



[2] `https://developer.github.com/v3/`
[3] *URL - Uniform Resource Locators*

A Figura - 3.3, exibe a URL utilizada para obter os PRs de um projeto. O `[owner]` é o parâmetro para informar o dono do repositório e `[repo]` é o nome do repositório do projeto. Foi passado como parâmetro o tipo de busca a ser feito utilizando o parâmetro `q=is:pr`, esse parâmetro é utilizado para filtrar apenas as *issues* que são do tipo PR. Além disso, utilizou-se um filtro para identificar se o PR foi *merged* (aceito) ou *unmerged* (não aceito).

A API responde à requisição em formato de arquivo JSON *(JavaScript Object Notation)*. Para fazer a leitura desse arquivo e automatizar a tarefa de coleta de dados, foram criados *scripts* em Python para percorrer todos os PRs do projeto informado na URL. Após percorrer cada item do arquivo JSON, os dados foram armazenados em um banco de dados. Os PRs coletados, de todos projetos, foram coletados até a data de criação: 10 de Setembro de 2017.

Além dos PRs coletados, utilizou-se também os dados do projeto GHTorrent GHTorrent (2019b). O GHTorrent monitora os eventos públicos de um repositório do GitHub e com isso armazena todas as informações obtidas. Essas informações estão disponíveis para download (GHTorrent, 2019a), para este trabalho utilizou-se o *database dump* de Outubro de 2017 do MySQL[4].

Após coletar os dados dos projetos, foi verificado que alguns projetos não poderiam fazer parte do estudo, isso ocorreu por 3 motivos:

1. **Alguns projetos possuíam menos que 5 *pull requests***: Vários projetos são populares no GitHub, porém possuem poucos PRs, como consequência, poucos contribuidores. Um exemplo disso são os projetos `lulzlabs/AirChat`,[5] `teijo/jquery-bracket`,[6] and `wg/wrk`[7].

2. **Projetos inativos**: Alguns projetos foram abandonados pelos seus mantenedores. Um exemplo disso é o projeto `ostinelli/misultin`[8] e `icefox/arora`[9]. Esses projetos possuem o seu último *commit* realizado no repositório a mais de sete anos atrás. Foram encontrados projetos que estão arquivados, como por exemplo `coolwanglu/vim.js`[10]).

[4] `http://ghtorrent-downloads.ewi.tudelft.nl/mysql/mysql-2017-10-01.tar.gz`
[5] `https://github.com/lulzlabs/AirChat`
[6] `https://github.com/teijo/jquery-bracket`
[7] `https://github.com/wg/wrk`
[8] `https://github.com/ostinelli/misultin`
[9] `https://github.com/icefox/arora`
[10] `https://github.com/coolwanglu/vim.js`

3. **Projetos que não aceitam *pull request***: Por fim, alguns projetos foram excluídos por não utilizarem o sistema de PR, dentro da plataforma do GitHub. Um exemplo de projeto é o `Linux`[11], que possui um *mirror* no GitHub, mas o seu desenvolvimento acontece em outras plataformas de codificação. Como consequência, nenhum PR criado no GitHub é aceito. Outro projeto que acontece o mesmo caso, é o projeto `clojure/clojurescript`[12], no qual os PRs são fechados porém informam aos contribuidores a forma correta de realizar a contribuição.

Além de coletar todos os PRs dos projetos, foram coletadas outras informações:

- **Estrelas**: Foi necessário coletar as informação de estrelas nos projetos, tendo em vista que essa informação é essencial para o trabalho e a mesma não se encontra no banco de dados do GHTorrent. Assim foram coletadas todos os usuários que marcaram o projeto com uma "estrela".

- **Comentários**: Foram coletados todos os comentários realizados nos PRs coletados. Os comentários coletados são importantes para a utilização das heurísticas no trabalho.

- ***Commits***: Foram coletados todos os *commits* realizados no "master branch" dos projetos. Após modificar o projeto é necessário salvar essas alterações para futuramente ser enviada em um PR, esse processo é chamado de *commit* Git (2019a). Os *commits* foram coletados com o propósito de serem utilizados nas heurísticas.

3.1.2 Classificação dos contribuidores e aplicação de heurísticas

O primeiro passo para classificar o contribuidor casual é estabelecer um requisito que determine que ele seja casual. O requisito adotado neste trabalho é: o contribuidor deverá ter um único PR aceito no projeto. Para isso, foram filtradas todos os PRs que possuíam o status de *"merged"*. Os PRs que possuem esse status são a certeza de que a contribuição foi aceita no tronco principal do projeto. Depois de investigar manualmente alguns PRs coletados, notou-se um comportamento interessante entre os PRs *"unmerged"*. Embora os PRs pareçam *"unmerged"*, é possível que eles tenham sido, de fato, *merged*.

Este comportamento ocorreu, basicamente, em três ocasiões: os *commits* foram *cherry-picked*[13]; os *commits* foram *squashed*[14] em um *commit* diferente; ou o conteúdo

[11] `https://github.com/torvalds/linux`
[12] `https://github.com/clojure/clojurescript`
[13] https://git-scm.com/docs/git-cherry-pick
[14] https://gist.github.com/patik/b8a9dc5cd356f9f6f980

do *patch* foi copiado para um *commit* separado. Esses falso-negativos são conhecidos na literatura de mineração de repositórios, que também propõe abordagens para lidar com elas (Gousios et al., 2014; Kalliamvakou et al., 2016). Decidiu-se aplicar uma versão conservadora da heurística do conjunto introduzido por Gousious e colegas (Gousios et al., 2014), para minimizar o efeito desses falso-negativos (ou seja, aqueles que foram aceitos no repositório do projeto, mas foram marcados como *unmerged*). As heurísticas são as seguintes:

1. **Palavra-chave**: A primeira heurística é baseada nos comentários realizados no PR. Para utilizar esta heurística coletou-se todos os comentários realizados nos PRs que estavam marcados como *unmerged*. Filtrou-se a seleção para todos os comentários que foram realizados até a data final da coleta (2017-09-10 23:59:59), e selecionou-se apenas os PRs que estavam marcados como *"closed"* (fechado). Para cada comentário coletado, armazenou-se informações como o nome do usuário e associação do usuário com o repositório[15], o corpo da mensagem e a data do comentário.

   Alguns PRs não possuíam comentários. Isto ocorre devido ao fato de que alguns contribuidores submetem seus PRs e nenhum outro contribuidor realiza algum comentário. Com isso alguns PRs não fizeram parte desta heurística. Ao todo obteve-se um total de 12.708 PRs que possuem comentários.

   Para aplicar a heurística, os dados foram filtrados da seguinte forma:

   - comentários realizados no intervalo de 1 dia antes ou depois da data de encerramento do PR. Foi descartado as primeiras mensagens realizadas no PR, considerando que as primeiras mensagens são de correções ou observações da contribuição, assim não sendo possível definir se a contribuição havia de fato aceita. Alguns PRs tiveram mais de 1 mensagem positiva antes ou após o encerramento do PR dentro desse intervalo. Mensagens positivas são comentários que possuem alguma palavra-chave presente. Foi selecionado apenas o comentário mais recente (mais próximo ao encerramento).
   - comentários que continham palavras-chaves presentes de acordo com a Tabela - 3.1. Essa lista possui um conjunto de palavras que foram utilizadas e induz a possibilidade da aceitação da contribuição. As palavras-chaves definidas foram baseadas na heurística proposta por Gousios et al. (2014), como por exemplo, ("pulled", "integrated", "pushed"). Uma análise manual foi realizada para

[15]https://developer.github.com/v4/enum/commentauthorassociation/

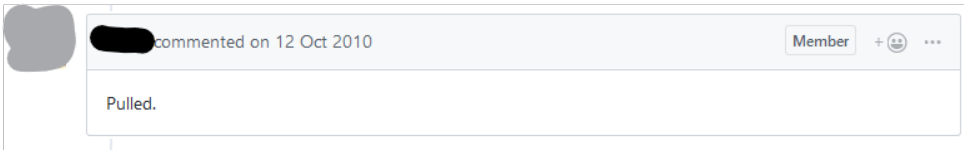
encontrar novas palavras, e algumas novas foram incluídas, como por exemplo, *"landed"*. As palavras-chaves selecionadas são palavras que pressupõe que o PR ou parte (algum *commit*) dele tenha sido aceito. As palavras utilizadas e suas ocorrências podem ser observadas na tabela Tabela - 3.1.

Tabela 3.1: Palavras-chaves utilizadas na heurística

Palavra-chave	**TOTAL de ocorrências**
integrating	42
applying	89
Looks great	109
integrated	164
cherry picked	172
pushing	334
pulling	725
Looks good	1.026
applied	1.063
pulled	1.164
pushed	2.752
LGTM	7.776
land	10.520
merg	35.817

A Figura - 3.4 exibe um PR que possui um status de "unmerged" mas após o uso da heurística foi possível determinar que o mesmo estava com o status incorreto.

Figura 3.4: Exemplo de comentário selecionado pela heurística Palavra-chave



2. **Presença do SHA nos últimos comentários**: Cada *commit* realizado no repositório é identificado por um valor único, este valor é conhecido como SHA (*Secure Hash Algorithm*). Em alguns comentários encontrou-se menções ao SHA do *commit* aceito no PR. O SHA é utilizado pelo GitHub para manter um histórico de mudanças no código, como quem fez a mudança e quando foi feita, entre outras informações (GitHub, 2019f).

   Nos comentários coletados encontrou-se o SHA-1, que é um número hexadecimal de 40 caracteres. Em alguns comentários encontrou-se esse valor de forma reduzida. O

GitHub utiliza uma forma mais simples de referenciar um SHA em um comentário chamado de *short-link* (GitHub, 2019d). O short-link transforma o SHA com 40 caracteres em um link relacionado de 7 caracteres, sendo que os 7 primeiros caracteres do short-link são os mesmos do SHA-1 com 40 caracteres.

Para verificar se o *commit* mencionado nos últimos comentários pertencia ao projeto, os SHAs obtidos foram cruzados com os dados da coleta realizada anteriormente. Conforme mencionado, os *commits* coletados foram apenas aqueles que pertenciam ao *branch* "master" do projeto. Além disso, os dados foram cruzados com os *commits* da tabela do GHTorrent.

Observou-se que, em alguns casos, os SHAs estavam presentes no evento *closed* do PR. Este é um dos tipos de evento que o GitHub armazena no contexto de uma *issue*. Tais eventos podem vir acompanhados de comentários, e, nesse caso específico, das informações de commits. Alguns membros ao encerrar o PR, informavam em sua mensagem de encerramento qual *commit* foi aceito. Todos os PRs que estavam marcados como *unmerged*, tiveram seus eventos coletados. Após essa coleta os dados foram novamente filtrados, afim de se obter apenas os eventos do tipo *closed*. Após aplicar o primeiro filtro, foi realizado um segundo filtro para obter apenas os eventos *closed* que possuíam o identificador do SHA.

Para a aplicação da heurística foram obedecidos os seguintes critérios:

- Os SHAs foram separados em 2 tipos, aqueles que foram *short-link* (7 caracteres) e também o SHA-1 (40 caracteres). Os comentários foram selecionados sem distinção do papel do usuário, ou seja, comentários realizados por membros, contribuidores, colaboradores, ou usuários que não possuem nenhum status no projeto. Comentários realizados no intervalo de 1 dia (antes ou depois) da data do fechamento do PR.
- Buscou-se dentro dos últimos comentários somente aqueles que possuíam uma cadeia de caracteres que podem representar um SHA. Para saber qual comentário tinha o essa cadeia, filtrou-se as mensagens utilizando a expressão regular no MySQL, **REGEXP \'b[0-9a-f] {7,39}\b'** para os short-link. A outra expressão regular utilizada foi: **REGEXP \'b[0-9a-f]{40,42}\b'** para o SHA-1 (40 caracteres).

Após obter todos os SHAs encontrados nos comentários, foi necessário verificar se o SHA encontrado no comentário pertencia a algum *commit* do projeto. Para isso, foram analisadas 3 fontes de dados:

(a) **Tabela de commits (GHTorrent)**: a tabela possui todos os *commits* realizados no repositório, em todas ramificações.

(b) **Commits por pull request (Coleta própria–API)**: todos os *commits* realizados na ramificação principal (*master-branch*) do projeto.

(c) **Eventos dos pull requests (Coleta própria–API)**: *commits* obtidos através da coleta dos eventos do PR. Para esta tabela foram criados os filtros para obter apenas os eventos do tipo *closed* e que possuem o identificador do SHA.

Alguns SHAs foram encontrados em mais de uma tabela, entretanto essa duplicidade não interfere nos resultados, pois o cruzamento das informações é aplicado para verificar a existência ou a falta do *commit* no projeto. A Figura - 3.5 exibe um exemplo de PR que originalmente estava como *unmerged*, mas com a utilização da heurística "SHA", foi considerado como contribuição aceita. É possível notar que o SHA aceito da contribuição esta presente no evento de encerramento do PR.

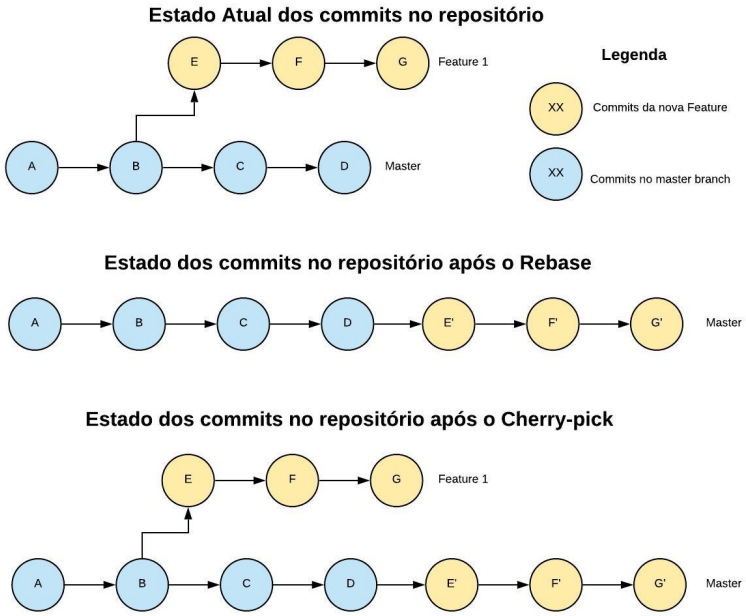
Figura 3.5: Exemplo de PR capturado pela heurística SHA (evento de encerramento)

3. **Commits no master-branch**: Esta heurística tem como objetivo verificar se algum *commit* presente no PR marcado como *"unmerged"*, está presente nos *commits* que foram aceitos no projeto na ramificação principal. Os projetos podem ter várias ramificações do seu código-fonte. Essas ramificações são chamadas de *branch* (GitHub, 2019h). Os branches são utilizados para que novas características possam

ser testadas no projeto sem comprometer o código-fonte da ramificação principal (*master branch*). Um PR pode ser criado em qualquer *branch* (ramificação). Quando os mantenedores do projeto resolvem aproveitar as contribuições realizadas em outras ramificações para ramificação principal do projeto, eles o fazem utilizando 2 métodos: *"rebased"* ou *"Cherry-picked"* (Git, 2019b).

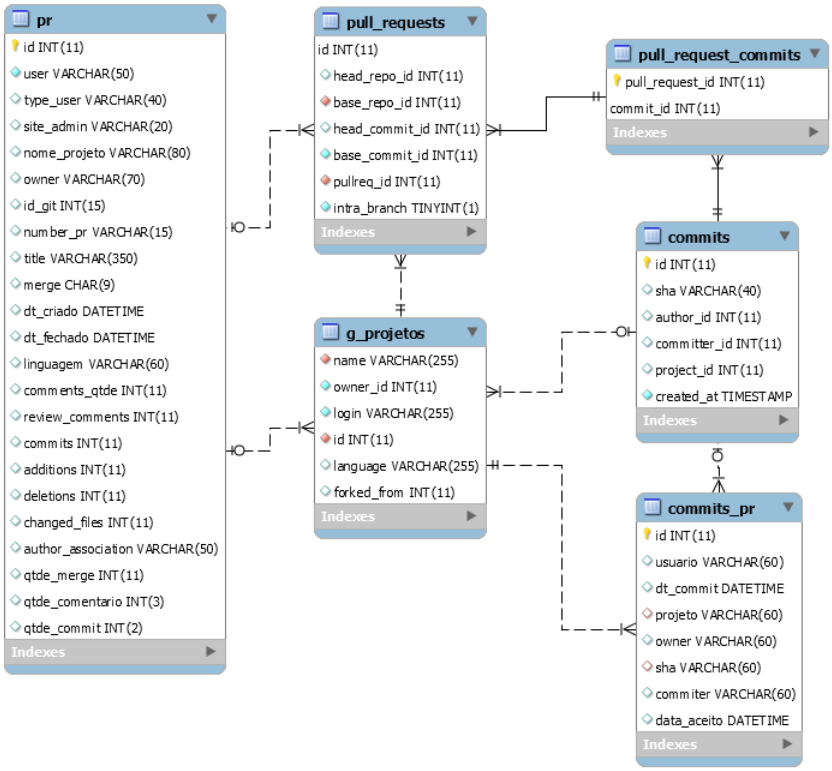
A Figura - 3.6 demonstra a diferença entre fazer o *rebase* e o *cherry-pick*. Quando o comando *rebase* é utilizado, os *commits* que pertencem a ramificação informada, são aplicados no *master branch*, registrando um novo *commit*. Neste caso o SHA é alterado, mas as informações do autor do *commit* é mantido. Por outro lado, quando utiliza-se o *cherry-pick* as mudanças aplicadas em um *branch* são aplicadas ao *branch master*, porém, mantendo todo o histórico dos *commits*.

Figura 3.6: Diferença entre realizar o *Cherry-pick* e o *Rebase*



A Figura - 3.7 ilustra os relacionamentos entre as entidades utilizadas para a heurística. A tabela "pr" presente no diagrama, é a tabela que foi criada na primeira coleta, conforme a seção 3.1.1. Nesta tabela existem todos os PRs *merged* e *unmerged* de todos os projetos coletados. O campo "numero_pr" representa o número de identificação do PR no projeto. A tabela "commits_pr" contém os *commits* coletados na ramificação principal do projeto (*master-branch*). Foi criado a tabela "g_projetos" para auxiliar nas consultas, essa tabela possui informações de todos os projetos utilizados neste trabalho (263 projetos no total). As tabelas

Figura 3.7: Diagrama de Entidade-Relacionamento



"pull_requests", "pull_request_commits" e "commits" são tabelas que pertencem ao GHTorrent[16] e foram utilizadas para realizar as consultas necessárias.

Para os casos em que foi possível identificar as contribuições *unmerged* (61.529 de 143.099 PRs), os dados foram marcados com a heurística que ajudou a identificar o falso negativo. Em seguida, foram selecionados todos os PRs *merged* que eram conhecidos. Foram selecionados os contribuidores que tiveram um único PR aceito, sendo o PR marcado como "merged" originalmente, ou capturado pela heurística.

3.1.3 Análise dos dados

Após a coleta dos dados (conforme a seção 3.1.1) e filtragem para obter apenas os contribuidores casuais nos projetos, a primeira observação realizada é sobre os dados obtidos.

A Tabela - 3.2 fornece informações numéricas sobre os dados coletados. Os dados usados neste trabalho estão disponíveis para fins de replicação no site complementar[17].

[16] `http://ghtorrent.org/relational.html`
[17] `https://github.com/markaumvb/casuais/replicacao`

Tabela 3.2: Informação sobre os dados coletados (por linguagem)

	# Projetos	# Pull Requests			# Commits
		Merged	Unmerged	Total	
C	15	16.793	5.937	22.730	185.738
C++	18	40.253	10.620	50.873	263.603
Clojure	18	2.914	657	3.571	25.966
CoffeeScript	18	7.449	2.515	9.964	45.583
Erlang	13	3.126	787	3.913	41.700
Go	16	16.604	3.228	19.832	68.867
Haskell	17	6.049	1.068	7.117	50.936
Java	17	23.245	5.601	28.846	118.020
JavaScript	19	33.997	14.458	48.455	131.512
Objective-C	19	6.813	2.385	9.198	37.923
Perl	13	2.218	768	2.986	28.965
PHP	18	24.357	8.268	32.625	142.307
Python	13	21.324	6.724	28.048	118.406
Ruby	17	64.101	14.549	78.650	228.909
Scala	19	17.772	2.923	20.695	88.670
Typescript	13	6.277	956	7.233	71.675

Foram coletados 374.736 PRs de 263 projetos para este trabalho. Deste total, 293.292 PRs tinham o status *merged* (78,2%) e 81.444 *unmerged* (21.8%).

Para a primeira pergunta do trabalho (QP1), a análise esta descrita na seção 4.3.1, para a segunda questão (QP2) a análise é realizada da seguinte forma:

- *Follow:* O contribuidor casual pode seguir outros contribuidores que participam do projeto. Nos casos em que o dono do projeto é um usuário, é possível que um contribuidor casual siga o dono do projeto no GitHub (exemplo, `ArnoldC`[18], no qual o dono do projeto é o usuário "lhartikk"). Quando um projeto pertence a uma organização, o contribuidor pode seguir os membros da organização e membros do projeto, como por exemplo o projeto `Activeadmin`[19]. Na plataforma do GitHub é possível obter informações sobre quem os contribuidores seguem dentro da rede conhecida como *followers*. Nesse caso específico, verificou se o contribuidor casual que esta sendo analisado segue (*follow*) algum deles.

- Estrela no projeto: os contribuidores podem marcar um repositório do GitHub com uma estrela[20], com intenção de manifestar o interesse ou a satisfação com o projeto

[18] `https://github.com/lhartikk/ArnoldC`
[19] `https://github.com/activeadmin/activeadmin`
[20] `https://help.github.com/articles/about-stars/`

(Borges et al., 2016). Neste trabalho analisou-se em qual momento os contribuidores casuais marcaram o projeto em que contribuíram casualmente com uma estrela (se antes ou após a sua contribuição ser aceita) ou se não estrelaram o projeto. Analisou-se tal informação como sendo *proxy* para o interesse do contribuidor casual no projeto.

- Observadores do projeto: foi analisado se o contribuidor casual marcou o projeto para ser observado (*watch*), e se isso foi realizado antes ou depois de sua contribuição. Quando um contribuidor marca o projeto para observar ele recebe todas as notificações de *issues*, PRs e comentários feitos por outros contribuidores (GitHub, 2018c). Observar um repositório sinaliza o interesse na atividade do repositório e um potencial interesse em contribuir (Sheoran et al., 2014).

- *Fork* no projeto: um *fork* é a cópia de um projeto para outra conta no GitHub (GitHub, 2018a). Para realizar uma contribuição, o contribuidor precisa realizar uma cópia do projeto e fazer suas modificações nesta cópia. Após fazer as modificações e enviar ao projeto, o contribuidor pode se desfazer dessa cópia ou mantê-la em seus repositórios. Foi analisado se a última cópia do projeto através de um *fork* foi realizada antes ou depois da sua contribuição. Além disso, analisou se o contribuidor manteve a cópia do repositório com novas modificações.

- Criação de uma nova *issue*: para verificar outros tipos de atividades dos contribuidores, observou se o contribuidor casual criou alguma *issue*, antes e/ou após a sua contribuição, além do PR do próprio contribuidor. Uma *issue* é utilizada para reportar bugs encontrados no sistema ou solicitar uma nova funcionalidade. Além disso a *issue* é utilizada para receber *feedback* do software GitHub (2019a).

- Comentário em outras *issues*: analisar se o contribuidor casual participa do projeto com comentários em *issues* ou PRs criados por outros contribuidores no projeto. Isso pode demonstrar o interesse do contribuidor com o projeto, contribuindo com o projeto com discussões. Essa atitude pode demonstrar o seu comprometimento com projeto.

- Submissão de outros *pull requests* não aceitos: Analisar a quantidade de tentativas de novas contribuições com o projeto que não foram aceitas. Essa questão é importante pois demonstra o interesse do contribuidor em tentar submeter outras contribuições. Foi analisado o comportamento do contribuidor antes e após a sua contribuição aceita, com isso é possível saber se o contribuidor persistiu nas

contribuições até ter uma aceita, ou se o mesmo tentou realizar novas contribuições após ter uma aceita.

- Analisar todos os *arquivos modificados* pelo contribuidor através dos *commits* realizados no PR aceito. Para isso coletou-se a informação de todos os arquivos modificados pelos contribuidores e aplicou-se as heurísticas baseadas no trabalho de Vasilescu et al. (2014). Para definir o tipo de contribuição realizada, observou-se a extensão do arquivo ou o diretório do arquivo modificado, por exemplo, caso o arquivo possua uma extensão `*.java`, atribui-se que a contribuição foi em um arquivo de código-fonte da linguagem de programação Java, caso o arquivo modificado possui uma extensão desconhecida, verificou-se qual o diretório ele esta, por exemplo, diretório "*.*/src", indica que o arquivo possivelmente seja um arquivo de código-fonte, porém sem definir qual a linguagem utilizada.

Por fim, para analisar a última questão deste trabalho (QP3), foi utilizado algoritmos de agrupamento conhecidos como *Model-based clustering* e *K-means* para agrupar os contribuidores casuais e determinar o perfil do grupo através de suas características. Além disso, os contribuidores casuais também foram agrupados manualmente, utilizando como variável dependente a tentativa de novas contribuições. Utilizar a técnica de regressão logística multinomial para saber se o modelo proposto é efetivo ou não. Os resultados para cada questão deste trabalho, se encontram nas seções QP1 - 4.3.4, QP2 e QP3- 5.4.

4

Replicação de estudo

Para identificar os contribuidores casuais nos projetos coletados, foram analisados dois métodos. Alguns trabalhos utilizam o número de *commits* para determinar os tipos de contribuidores, enquanto outros utilizam PRs. A replicação do trabalho de Pinto et al. (2016), busca comparar o método proposto por Pinto e seus colegas, utilizando o número de *commits*, pelo método utilizado porGousios et al. (2014), para selecionar os contribuidores casuais.

4.1 Replicação

Nenhum trabalho até o presente momento comparou os 2 métodos citados anteriormente. Através da replicação, é possível comparar um método com o outro e os seus resultados afim de se obter os melhores resultados.

4.1.1 Por que replicar um experimento?

Estudos de replicação estão atraindo a comunidade de engenharia de software e este tipo de pesquisa empírica está aumentando ao longo dos últimos anos (Bezerra et al., 2015a). Campbell (1963) argumenta que experimentos precisam ser replicados em diferentes contextos, em diferentes momentos e sob diferentes condições antes que eles possam produzir conhecimento generalizável. Assim, as replicações podem ajudar a melhorar a compreensão de um fenômeno, uma vez que os resultados relatados por um estudo nem sempre podem ser transferidos diretamente para outros contextos (Dinh-Trong e Bieman, 2005).

Shull et al. (2008) identificaram dois tipos de replicações: (i) replicações exatas, quando os pesquisadores aplicam os mesmos procedimentos para responder as mesmas questões de pesquisa o mais próximo possível; e (ii) replicações conceituais,isso acontece quando os pesquisadores investigam a mesma questão de pesquisa usando um procedimento experimental diferente.

Pode-se também classificar uma replicação como interna ou externa. As replicações internas são aquelas realizadas pelos mesmos pesquisadores do estudo original, enquanto as replicações externas são aquelas realizadas por um grupo diferente de pesquisadores (Bezerra et al., 2015b). Apesar dos tipos de replicação escolhidos pelos pesquisadores, vale mencionar que a ideia principal é melhorar o estado da arte discutindo novos métodos e novos resultados descobertos na replicação (Chen et al., 2019). Neste trabalho foi realizado uma replicação conceitual e externa.

4.2 Estudo Original

No estudo original foram propostas 3 questões:

QP1. Quão comum são os contribuidores casuais em projetos de OSS (Software Livre)?

QP2. Quais são as características de uma contribuição casual?

QP3. Como contribuidores casuais e mantenedores de projetos percebem as contribuições casuais?

O estudo original teve como objetivo obter uma compreensão aprofundada dos contribuidores casuais, bem como os benefícios e problemas por trás dele. A **QP1** forneceu uma visão geral da existência de contribuidores casuais no conjunto de projetos estudados. Na **QP2**, os autores realizaram uma inspeção manual para entender a intenção do contribuidor casual ao enviar sua contribuição. Resultados de uma pesquisa realizada com contribuidores e mantenedores casuais foi apresentada na **QP3**, na qual eles investigaram as motivações, benefícios e problemas por trás da presença dos contribuidores casuais.

Na replicação, a **QP3** não foi replicada. Esta decisão deu-se devido ao foco em uma melhor compreensão dos métodos utilizados para identificar os contribuidores casuais que podem afetar os principais resultados do trabalho original.

4.2.1 Design

No estudo original os autores conduziram uma análise quantitativa e qualitativa dos dados dos projetos de SL. Para selecionar projetos representativos de SL, os autores consultaram

o GitHubArchive[1] para encontrar os projetos mais populares, em termos do número de estrelas. Os autores clonaram os projetos em uma máquina local e usaram os comandos `git` para extrair os logs de *commits* e o conteúdo dos arquivos. Eles também categorizaram manualmente 384 contribuições casuais analisando os *commits* feitos por contribuidores casuais.

4.2.2 Participantes

Foram selecionados os 20 projetos mais populares em 16 linguagens de programação. Para isso utilizaram a quantidade de estrelas GitHub (2019g) que cada projeto possui. As linguagens utilizadas foram: C, C++, Clojure, CoffeeScript, Erlang, Go, Haskell, Java, JavaScript, Objective-C, Perl, PHP, Python, Ruby, Scala, e TypeScript. Depois de remover alguns projetos que não eram projetos de software ou estavam inativos, obtiveram um total de 275 projetos do GitHub analisados. Esses projetos tiveram um total de 73.960 contribuidores que realizaram 2.039.376 contribuições.

4.2.3 Artefatos

O estudo original compartilhou todos os dados coletados. No entanto, o conjunto de dados apresenta apenas dados de *commits* coletados via `git log`, impossibilitando a análise de PRs. Neste estudo de replicação, criou-se um novo conjunto de dados, contando com dados obtidos diretamente por meio da API do GitHub e do GHTorrent[2], incluindo dados de PRs e *commits*, para comparar essas duas abordagens diferentes. Na seção 3.1.1, é possível notar que algumas modificações nas heurísticas foram realizadas, além de algumas modificações na metodologia do estudo original para que seja possível analisar os *commits* e os PRs.

4.3 Replicação

Os dados utilizados para a replicação, são os mesmos dados que foram obtidos pela coleta de dados na seção 3.1.1

[1] `https://www.gharchive.org/`
[2] `http://ghtorrent.org`

4.3.1 Análise dos dados

1. **Análise Manual**: Uma análise manual dos dados obtidos também foi realizada. Para responder à QP2, o conteúdo das contribuições feitas por contribuidores casuais foram analisados. Para isso, foi selecionado uma amostra estatisticamente significativa de 384 contribuições casuais (95% de confiança e 5% de margem de erro) realizadas em 263 diferentes projetos. Para cada contribuição, estudou-se o título do PR, a descrição e a discussão do PR, além da mensagem de *commits*, e o código alterado. Utilizou-se as categorias de Pinto do estudo (Pinto et al., 2016) original para classificar os PRs.

   Comparou-se a eficácia dessas duas abordagens (*commits* e PRs). Para tanto, foi investigada uma amostra de 442 contribuidores (95% de confiança e 5% de margem de erro) da amostra. Os contribuidores foram classificados manualmente em contribuidores casuais (quando identificou-se apenas uma única contribuição), regular (quando o contribuidor tinha mais de uma contribuição mesclada ao repositório) ou quasi-contribuidor (quando nenhuma contribuição foi aceita). Para analisar as contribuições, foram investigados cuidadosamente os *commits* feitos pelo usuário na ramificação *master*) do projeto, os PRs enviados e as discussões realizadas no PR para assegurar se a contribuição foi aceita ou não. Depois de classificar manualmente esse conjunto, foi utilizada a classificação para comparar com os resultados obtidos usando *commits* e PRs.

   Dois pesquisadores realizaram ambas as análises manuais. Trabalharam independentemente e, quando necessário, as discrepâncias foram resolvidas após uma reunião de discussão com um terceiro pesquisador.

2. **Análise estatística**: Foram realizados testes estatísticos para comparar a porcentagem de contribuidores casuais identificados usando PRs e *commits*. Depois de confirmar que os dados não seguiam uma distribuição normal para qualquer projeto – aplicando o teste de normalidade de `Shapiro-Wilk` – foi realizado o teste de *Wilcoxon signed-rank*(Wilcoxon, 1945). Também foi aplicado o delta (Grissom e Kim, 2005) de *Effect size*(tamanho de efeito), para saber a distância média entre as amostras. Os resultados são interpretados usando os limiares fornecidos em Romano et al. (Romano et al., 2006), isto é, $delta < 0,147$ (*insignificante*), $delta < 0,33$ (*pequena*), $delta < 0,474$ (textit média), e $delta >= 0,474$ (*grande*).

Os resultados dos testes podem ser observados na tabela Tabela - 4.1.

Tabela 4.1: Resultados dos testes e *effect size*

Linguagem	**p-value**	***Cliff's Delta***	***Effect size***
C	0,06	0,52	grande
C++	<0,001	0,69	grande
Clojure	<0,001	0,92	grande
CoffeeScript	<0,001	0,77	grande
Erlang	<0,001	0,64	grande
Go	0,03	0,87	grande
Haskell	0,01	0,78	grande
Java	0,01	0,81	grande
Javascript	<0,001	0,69	grande
Objective-C	<0,001	0,82	grande
Perl	<0,001	0,62	grande
PHP	<0,01	0,82	grande
Python	<0,001	0,88	grande
Ruby	<0,001	0,61	grande
Scala	<0,001	0,54	grande
Typescript	<0,001	1,00	grande

4.3.2 Estudo dos Resultados

QP1. Quão comuns são os contribuidores casuais em projetos de SL?

Como no estudo original, foi analisado o número de contribuições feitas por contribuidores nos projetos coletados. Os histogramas apresentados na Figura - 4.1 mostram o quadro geral do número de PRs feitos pelos contribuidores, considerando todos os projetos da amostra (contribuições por contribuidor: mediana = 1, média = 4,6, Q3 = 2 , desvio padrão = 33,27). A primeira descoberta é que, independentemente da linguagem de programação, poucos contribuidores são responsáveis pela maioria dos PRs, enquanto a maioria dos contribuidores realizam poucos PRs.

Usando PRs, encontrou-se não apenas um número (relativo) maior de contribuidores, mas também de contribuições casuais, em comparação com o estudo original. Por exemplo, para cada projeto analisado, escrito principalmente em TypeScript, encontrou-se mais contribuidores casuais usando o PR comparado com o uso de *commits* (veja a Figura - 4.2). Em relação às contribuições, os projetos de linguagem de programação Clojure tiveram um aumento de mais de 25% das contribuições feitas pelos programas casuais (veja a Figura - 4.3).

Figura 4.1: Distribuição de contribuições e contribuidores nos projetos analisados. (*Outliers* foram removidos para uma visualização mais fácil).

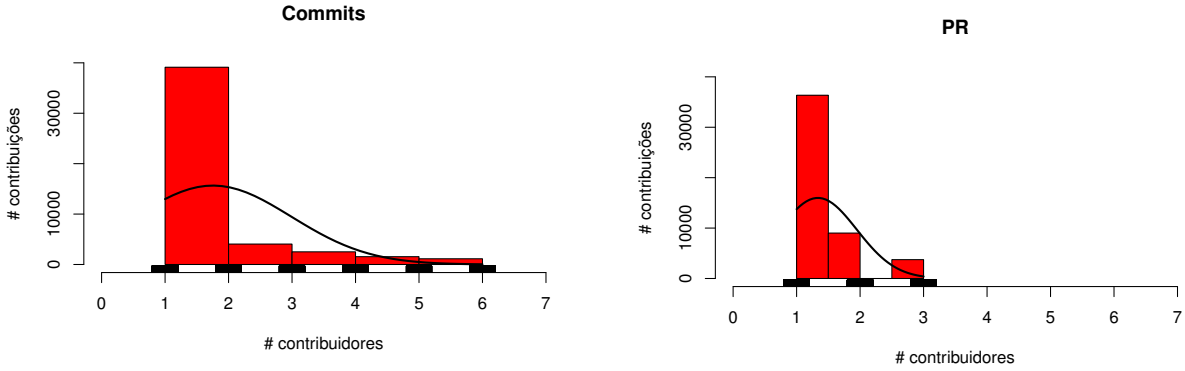


Figura 4.2: Percentual dos contribuidores casuais por linguagem de programação

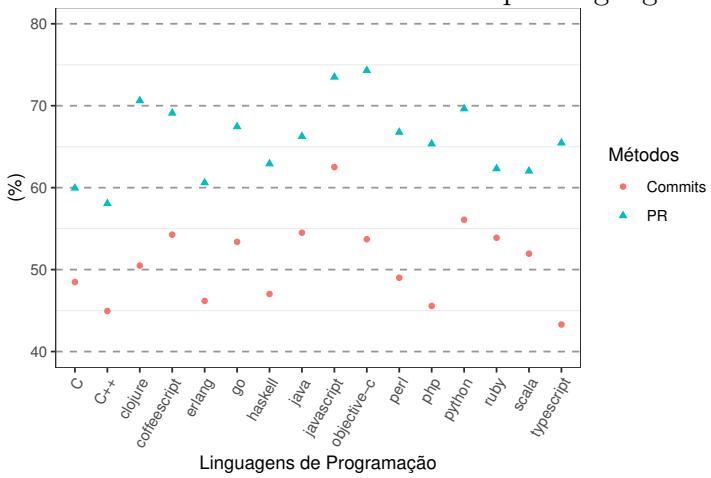
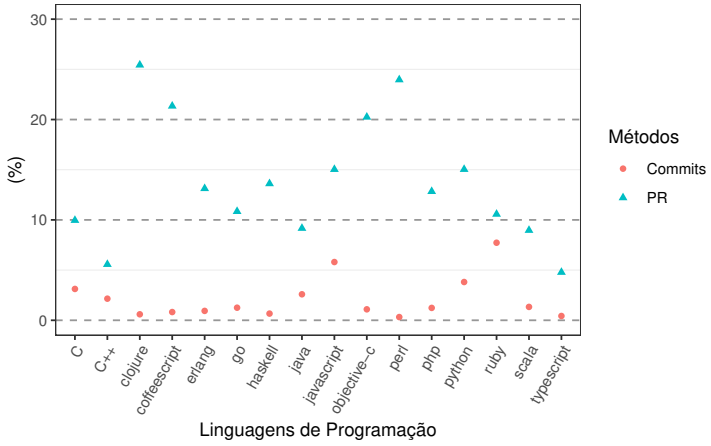


Figura 4.3: Percentual das contribuições casuais por linguagem de programação



As Figuras: Figura - 4.4 e Figura - 4.5 apresentam uma visão geral dos contribuidores casuais por projeto na forma de boxplots (removendo os *outliers* para facilitar a visualização). Cada boxplot representa os dados dos contribuidores de todos os projetos analisados, agrupados por sua principal linguagem de programação. Uma das primeiras observações notáveis desta figura é que, usando PRs, a proporção de contribuidores casuais é maior, independentemente da linguagem de programação. A diferença pode ser tão pequena quanto 8,43% (Ruby), mas pode ser tão alta quanto 22,16%, que é o caso de projetos escritos em TypeScript. Para cada projeto nessa linguagem específica, identificou-se uma porcentagem maior de contribuidores casuais ao usar PR do que ao usar *commits*. Ou seja, a abordagem usada para minerar dados de contribuição tem uma influência não trivial nos resultados.

No geral, 66,02% dos contribuidores dos projetos analisados tiveram um único PR *merged* (para um projeto); assim, foi classificado como contribuidor casual. Em comparação com o número apresentado no estudo original (Pinto et al., 2016) (48,98%), isso representa um aumento de ∼35%. Para fazer uma comparação justa, a porcentagem de contribuidores casuais encontrados analisando os *commits* para a mesma amostra e período usado para a análise de PR (até outubro de 2017) foi de 52,88% no nível de *commit*. Assim, a diferença de usar PR está longe de ser insignificante (∼25%).

Analisando mais profundamente—em nível de projeto—os extremos das amostras, em nove projetos mais de 90% dos contribuidores foram identificados como casuais. Por outro lado, 18 projetos esse número caiu para menos de 50% (dois deles com menos de 35%). Essa descoberta é particularmente interessante: os projetos que tem o menor número de contribuidores casuais ainda tem uma quantidade não desprezível deles. No entanto, oito dos 18 projetos com menos contribuidores casuais foram escritos em C ou C++ (4 cada) e outros 4 em Scala. Isso talvez explique as iniciativas do Scala Center para atrair mais contribuidores[3].

Projetos entre os 20% com menor variação em termos de % de casuais (variando entre ∼7% to 10%) incluem projetos grandes e consolidados como `node/node-js`, `bitcoin/bitcoin`, `spree/spree`, `elastic/elasticsearch`, e `homebrew/homebrew-cask`. Por outro lado, para os projetos entre os 20% com maior variação (variando de 24% a 84%), é possível notar duas coisas interessantes: (i) eles compreendem projetos novos / menos populares (como como `Engelberg/instaparse`, `MojoJolo/textteaser`, `winjs/winjs`); (ii) 12 projetos escritos em Clojure e 7 escritos em Typescript. Isso pode indicar que projetos mais recentes e aqueles desenvolvidos usando idiomas novos ou de tendência, são mais aderentes

[3] `https://github.com/scalacenter/sprees`

ao modelo *pull-based*, fazendo uso mais extensivo de PR como unidade composta de muitos *commits* que representam uma única contribuição.

Figura 4.4: Distribuição dos contribuidores casuais por projeto (agrupado por linguagem de programação). Boxplot da esquerda significa *commits* enquanto o da direita significa *pull request*.

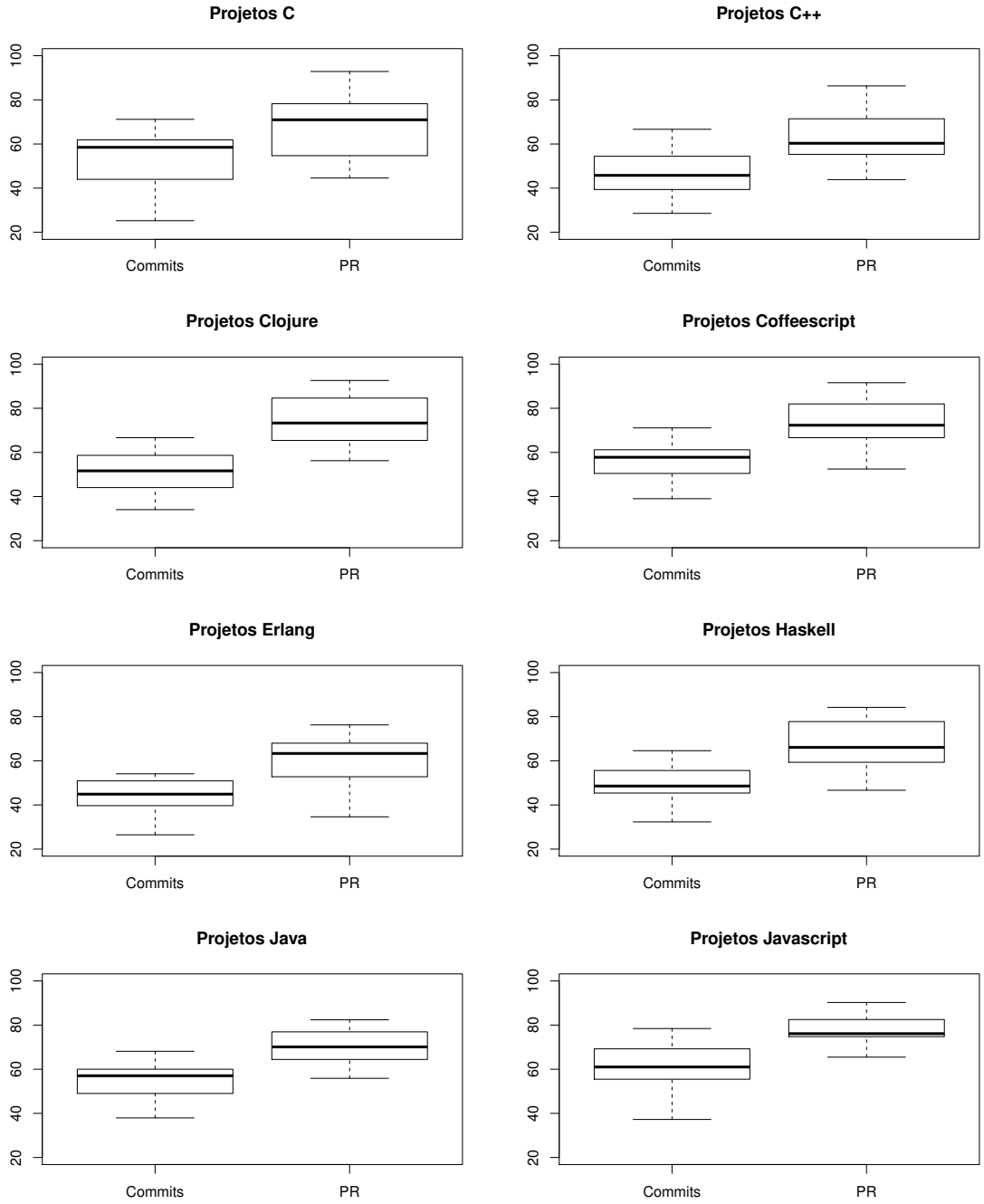
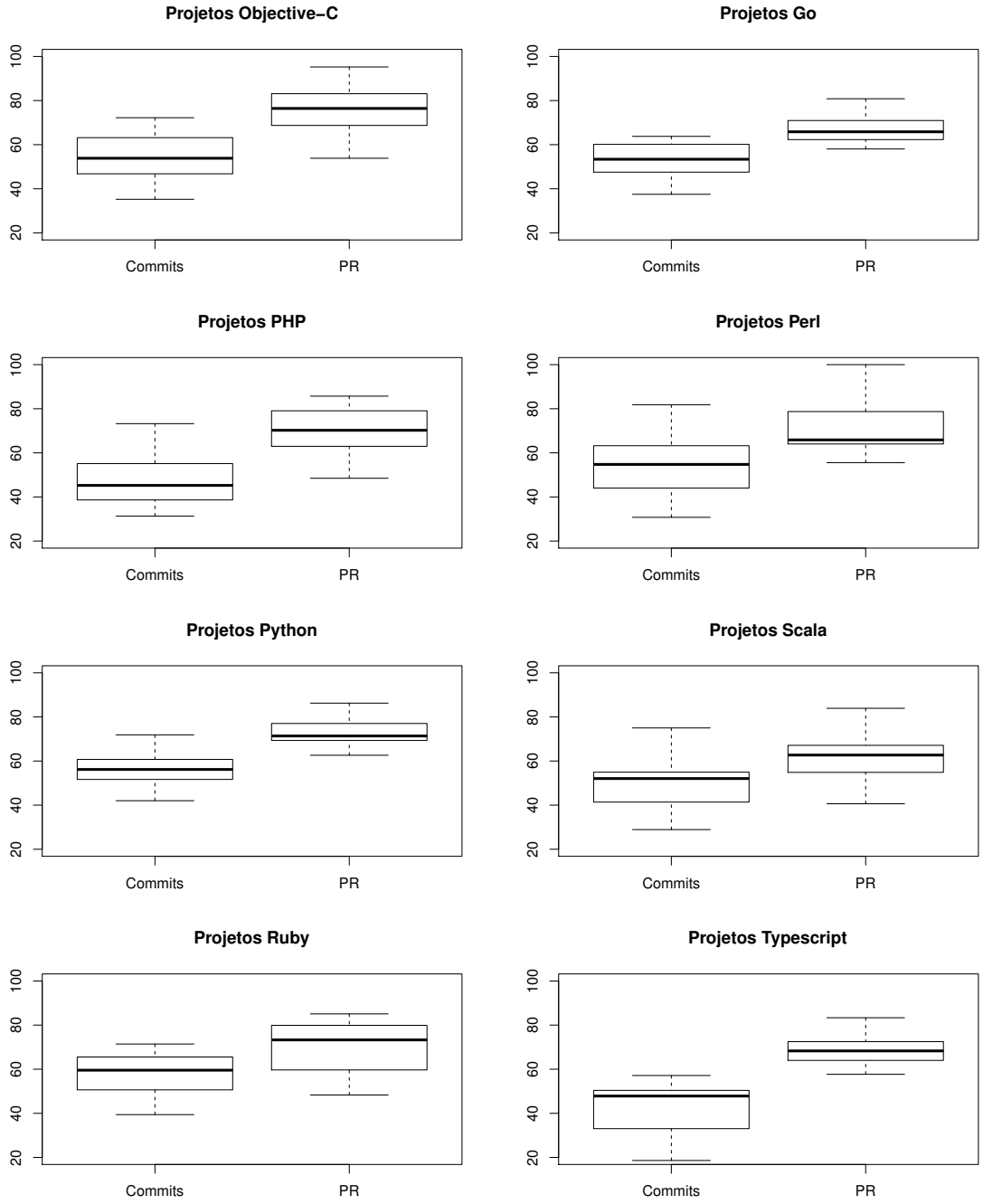


Figura 4.5: Distribuição dos contribuidores casuais por projeto (agrupado por linguagem de programação). Boxplot da esquerda significa *commits* enquanto o da direita significa *pull request*.



Analisando o número de PRs, nota-se que os contribuidores casuais realizaram 12,5% do total de PR *merged* dos projetos analisados. Esse número é maior (de longe) do que a proporção apresentada no estudo original (Pinto et al., 2016) (1,73%). O resultado

usando os dados de *commits* coletados para essa replicação (mesmo período que os dados usados para os PRs), mostrou um resultado próximo ao reportado no estudo original (contribuidores com um único *commit* enviaram 1,83% do total de *commits*). A porcentagem mais alta de "contribuições" na forma de PR não é uma surpresa, pois: (i) há uma porcentagem maior de contribuidores casuais identificados quando usando PR (Fig. Figura - 4.4 e Figura - 4.5) e (ii) o número de PRs é, em geral, menor que o número de *commits* (veja Tabela - 3.2) porque os PRs podem conter vários *commits*.

Ao analisar a porcentagem de PRs feitas por contribuidores casuais por projeto, é possível notar uma distribuição esparsa. Embora 13 projetos tenham contribuidores casuais responsáveis por mais de 65% dos PRs, outros 15 projetos com menos de 5% de PR de contribuidores casuais. Os projetos `facebook/Shimmer` e `weavejester/compojure` receberam respectivamente 90,91% e 86,21% de suas contribuições de contribuidores casuais, enquanto em `scala-js/scala-js` e `mozilla/shumway` os contribuidores casuais representam 1,82% e 2,21%.

QP2. Quais são as características de uma contribuição casual?

Foi analisado o número de arquivos alterados e o número de adições e exclusões realizadas pelos contribuidores casuais em seus PRs. Cada PR tem, em **média**, 1.804 linhas adicionadas (Q1 = 2; mediana = 11; Q3 = 55), 890 linhas excluídas (Q1 = 1; mediana = 3; Q3 = 16), 16,8 arquivos alterados (Q1 = 1; mediana = 2; Q3 = 4) e 14,3 cometer (Q1 = 1; mediana = 1; Q3 = 2).

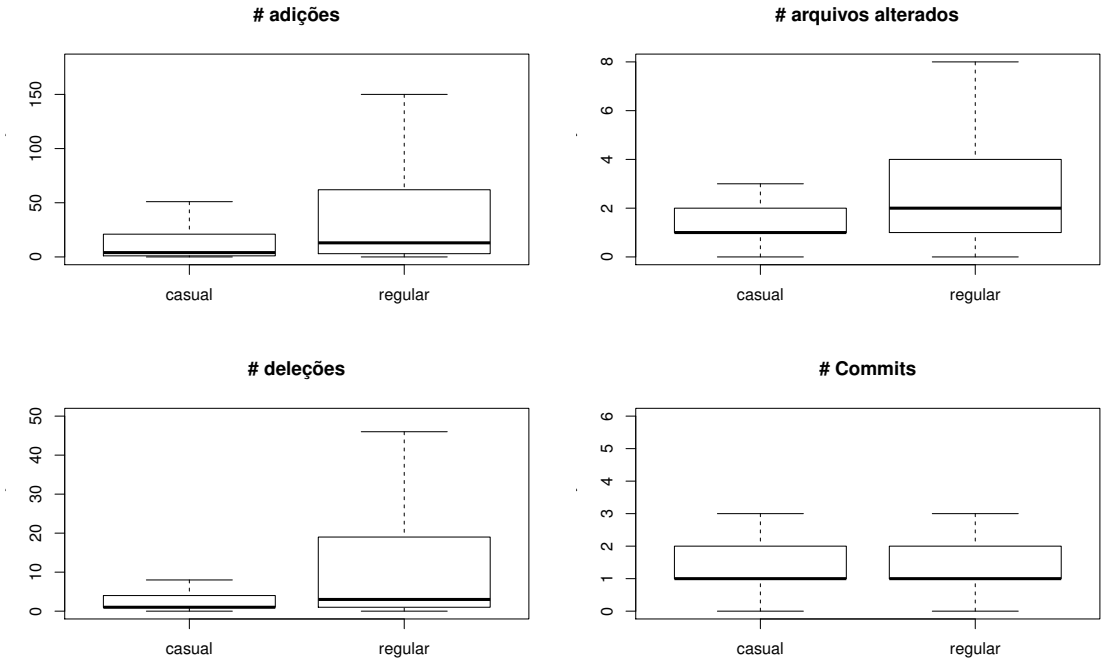
Observando os dados por linguagens de programação, os contribuidores casuais modificam, em **média**, mais arquivos na linguagem de programação C++ (35,2) e menos arquivos no Clojure (3,2). Em termos de número de *commits*, os projetos que possuem uma maior quantidade são feitos em C++ (35,75) e com menos em projetos da linguagem Clojure (2,5). Em termos de adições, os contribuidores casuais adicionam mais linhas de código, em **média**, em projetos C++ (5227,3) e menos em projetos Ruby (86,4). Finalmente, em termos de exclusão, os projetos de C têm mais exclusões de linha, em média (1.562,8), e os projetos de Perl têm o mínimo (27,6).

A Figura - 4.6 apresenta a distribuição do número de adições, exclusões, arquivos alterados e *commits* por PR — considerando todos os 263 projetos — comparando contribuidores casuais e regulares. Para facilitar a visualização, os *outliers* foram removidos das figuras. Os boxplots mostram alguns padrões interessantes. Primeiro, as contribuições dos casuais são significativamente menores (ou seja, elas têm menos adições e exclusões e alteram menos arquivos).

Segundo, como apontado no estudo original (Pinto et al., 2016), uma parte não insignificante das contribuições apresentam vários arquivos alterados, com múltiplas

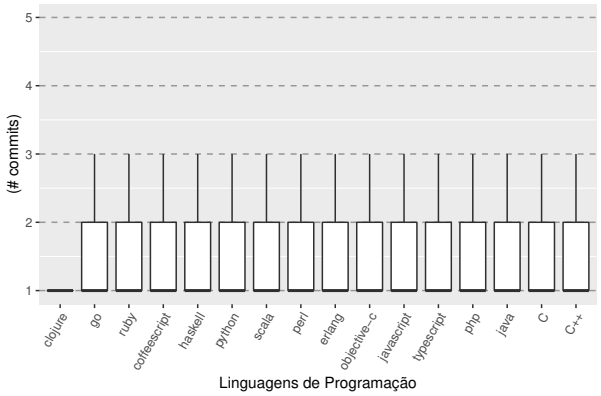
adições e exclusões. Além disso, ambos os grupos (casual e regular) executam um número muito semelhante de *commits*. Em particular, o número mediano de *commits* por PR é um (dois é o terceiro quartil e três é o máximo). Embora este resultado esteja particularmente alinhado com a descoberta de Gousios, que observou que a maioria dos PRs vem junto com um único *commit* (Gousios et al., 2014), o que sugere que estes PRs podem não ser o objetivo de um minucioso ciclo de revisão de código (que podem ter mais *commits* adicionais para melhorar as alterações solicitadas). Em uma análise manual de uma pequena amostra de 20 PRs feitos por contribuidores casuais com apenas um *commits*, observou que apenas três deles tiveram revisões de código. A Figura - 4.7 mostra de forma mais detalhada a quantidade de *commits* por PRs realizados por contribuidores casuais. No entanto, ao olhar mais de perto os PRs *outliers*, observa-se que 3.930 (∼9%) dos 42.006 PRs têm mais de três *commits*, o que corresponde a 112.056 de 160.970 (69,61%) *commits* dos PRs dos contribuidores casuais.

Figura 4.6: Características dos *Pull requests*



Para investigar melhor o objetivo das contribuições casuais, foi analisado manualmente uma amostra de 384 PRs realizados por contribuidores casuais. A Tabela - 4.2 resume os resultados da análise manual e a comparação com os resultados originais do estudo. Além de algumas pequenas diferenças, é possível notar que os tipos de contribuições feitas pelos contribuidores casuais estão de acordo com os encontrados no estudo original. Uma

Figura 4.7: Quantidade de *commits* por *pull request* feitos por contribuidores casuais.



notável semelhança na primeira categoria de intenções do grupo: "correção de BUGs". Nesta categoria, os números absoluto e percentual coincidem com os encontrados no estudo original. Também semelhante ao estudo original, algumas contribuições casuais que vão desde simples erros de digitação[4] até correções que exigem conhecimento em determinados tópicos, como essa correção enviada para o projeto XBMC[5] que corrige uma falha de vídeo ao ajustar o volume do controle remoto do XBMC Android, que requer conhecimento sobre cache e kernel do Linux.

Quanto às diferenças, foi encontrado 12 contribuições que não puderam ser classificadas de acordo com as categorias originais, foram classificadas como "Outras". Exemplos de "Outras" incluem, por exemplo, a criação de um utilitário de script para criar um arquivo rpm[6], aprimorando a configuração de Integração Contínua[7], e aumentando os *commits merged* em uma determinada ramificação da versão[8].

4.3.3 Comparando os resultados

De acordo com Carver (2010), um dos principais valores de um estudo de replicação é a comparação de seus resultados com os resultados do estudo original. A Tabela - 4.3 resume as principais descobertas de ambos os estudos.

Resultados consistentes: Algumas das descobertas que estão de acordo com o estudo original incluem: (i) há um número não desprezível de contribuidores casuais

[4] `https://github.com/thoughtbot/paperclip/pull/822`
[5] `https://github.com/xbmc/xbmc/pull/2065`
[6] `https://github.com/ariya/phantomjs/pull/342`
[7] `https://github.com/tornadoweb/tornado/pull/539`
[8] `https://github.com/trinitycore/TrinityCore/pull/13703`

Tabela 4.2: Categorização dos contribuidores casuais.

Categoria	*Pull requests* #	*Pull requests* %	Estudo Original #	Estudo Original %
Correção de BUGs	116	30,20%	116	30,20%
Documentação	99	25,78%	110	28,64%
Adicionar novo recurso	39	10,10%	72	18,75%
Refatoração	46	11,98%	34	8,85%
Atualizar versão/dependências	37	9,64%	25	6,51%
Melhorar o erro/mensagens de ajudas	19	4,95%	14	3,64%
Melhorar o uso de recursos	5	1,30%	8	2,08%
Adicionar/Corrigir casos de teste	12	3,12%	5	1,30%
Outros	12	3,12%	–	–

- comparando o número de casuais obtidos, no nível do projeto, 259 dos projetos em que o número de contribuidores casuais foram maiores usando PR em comparação com o uso de *commits*, apenas quatro projetos apresentaram mais contribuidores casuais ao usar a abordagem de *commits*, proporcionalmente. Um exemplo disso é o projeto `ninenines/cowboy`, em que selecionando contribuidores casuais utilizando os *commits*, 70,33% dos contribuidores eram casuais; enquanto pelo método PR 68,00% eram casuais. (ii) Contribuições realizadas por contribuidores casuais não são triviais. Alguns exigem um conhecimento profundo do código-fonte e das tecnologias envolvidas. Além disso, muitas contribuições modificam arquivos diferentes e lidam com melhorias no uso de recursos e refatoração. (iii) Embora muitas contribuições estejam relacionadas à documentação (correção de erros/gramática, adição de links, alteração de cabeçalhos), a maioria das contribuições está relacionada ao código (correção de *BUGs*, adição de novos recursos, refatoração).

Resultados diferentes: Existem diferenças entre os dois estudos. Como é possível perceber os resultados, a principal diferença dos resultados do estudo original e dessa replicação gira em torno do tamanho da população de contribuidores casuais. Usando PRs, a porcentagem média de contribuidores casuais aumentou em mais de 30%. É possível afirmar isso através do teste *Wilcoxon signed-rank* para cada linguagem de programação, seguido por um teste de delta de *Cliff* para avaliar o tamanho do efeito (*effect size*). Todas as diferenças são, de fato, significativas (todos os *p-value*<0,05), exceto para os projetos escritos em C, que eram marginalmente significativos (*p-value* = 0,06). Para todos os casos, o tamanho de efeito foi grande de acordo com o delta de *Cliff* (todos>0,5).

Tabela 4.3: Sumário dos resultados dos estudos

	Estudo Original	**Replicação**
Método	*Commits*	*Pull requests*
Tamanho da amostra	275	263
Descobertas	**QP1**: 48,98% dos contribuidores que foram analisados são contribuidores casuais. No entanto, esses contribuidores são responsáveis por apenas 1,73% do total de contribuições no conjunto de projetos analisados. **QP2**: Após uma inspeção manual da amostra das contribuições casuais, eles encontraram que 28,64% delas são relacionadas a erros de digitação , 30,20% correção de BUGs, 18,75% propor novas características, e 8,85% refatoração de código.	**QP1**: o total de contribuidores casuais é 66% na amostra (atingindo mais de 90% em alguns projetos). Além disso, os contribuidores casuais são responsáveis por 12,5% do total de PR nos projetos analisados. **QP2**: Em uma análise manual das contribuições casuais PRs, descobriu-se que 25,78% delas estão relacionadas à documentação, 30,20% correção de BUGs, 11,98% refatoração de código, e 10,10% adicionar novas características. Descobriu-se também que PRs aprimorando a configuração e o volume de Integração Contínua previamente mesclados com *commits* em outras ramificações.

4.3.4 Discussão dos resultados e lições aprendidas

Foram encontrados cerca de 30% mais contribuidores casuais ao usar o método PR comparado aos *commits*. Entretanto, isso não significa que a replicação supere o estudo original. Para possibilitar uma comparação justa, para cada abordagem foram investigados manualmente 442 contribuidores que foram identificados com PRs aceitos. Foram encontrados 50 casos (11,3%) de usuários classificados de forma incorreta utilizando PR, em comparação com 116 (26,2%) casos ao usar *commits*. Os resultados da análise manual são apresentados na tabela Tabela - 4.4.

Tabela 4.4: Análise Manual dos contribuidores casuais. As células hachuradas representam os contribuidores classificados corretamente - As colunas representam a análise manual

Classificação	***Pull requests***		***Commits***		
Manual	**Casual**	**Regular**	**Casual**	**Regular**	N/A
Casual	251	15	176	30	60
Regular	8	141	4	123	22
Quasi Contribuidor	26	1	–	–	27
Total	418	24	299	34	109

Contribuidores do GitHub sem *commits*: A primeira observação interessante desta análise foi que 109 dos contribuidores analisados manualmente (24,7%) não tiveram *commits* criados ou identificados por seus nomes de usuários do GitHub. De fato, 82 deles tiveram alguma contribuição aceita. Analisando os *commits* sob os PRs, nota-se que a causa raiz é que os contribuidores fizeram *commits* usando um endereço de e-mail diferente daquele vinculado à sua conta do GitHub — o que leva a um usuário desconhecido no sistema GitHub — ou incluiu confirmações de terceiros.

***Commits* podem perder contribuidores casuais (porque PR agrupam os *commits*)**: O método centrado em *commits* teve uma perda de 30 contribuidores casuais (6,8% da amostra). Isso aconteceu porque esses contribuidores tinham mais de um *commit* em um único PR. Essa é uma desvantagem esperada de utilizar *commits*, já que as PRs são projetados para agrupar os *commits* relacionados à mesma contribuição — reduzindo os problemas com diferentes hábitos de *commit* (Herzig et al., 2016). Isso fica claro em casos como o exemplo identificado no projeto iPython[9]. Nesses casos, os PRs capturam melhor o fenômeno de contribuidores casuais do que os *commits*.

As heurísticas dos PRs também apresentam falhas: Embora as heurísticas tenham ajudado a descobrir 61.529 de 143.099 (43%) PRs marcados como *"unmerged"*—que são potencialmente "merged"—alguns erros de classificação foram encontrados. Um exemplo disso é que 15 contribuidores casuais reais (3,4%) foram classificados como regulares ao usar as PRs. Além disso, 27 contribuidores (6,1%) sem nenhum *commit* ou PR (quasi-contribuidores). Isso aconteceu porque as heurísticas aplicadas aos PRs *"unmerged"* identificaram alguns falso-positivos (por exemplo, `https://github.com/spree/spree/pull/938`). Embora as heurísticas usadas possam levar a falso-positivos, o uso das mesmas é altamente encorajado. Mas isso também sugere que ainda há espaço para melhorar as heurísticas.

Os desenvolvedores podem usar *commits e PRs*: Foram identificados oito falso-positivos que foram classificados como contribuidores casuais usando PRs mas, na verdade, tiveram mais de uma contribuição aceita. A principal razão para isso é que os desenvolvedores tinham *commits* enviados diretamente para o repositório, sem criar um PR. Isso aconteceu, ao menos, nos seguintes cenários: (i) contribuições feitas via `git` que ignoravam o fluxo do PR[10]; (ii) *commits* feitos antes da migração do projeto para o GitHub[11].

O `git` pode atrapalhar as coisas: Ao realizar a análise manual, percebe-se que os mantenedores do projeto empregam comandos `git` como *rebase*(reescreve o histórico de *commit*) e *cherry-pick*(ou seja, não altera o histórico existente; em vez disso, adiciona ao histórico) que alteram o estrutura de uma contribuição. Essas operações têm o potencial de excluir *commits* (com *rebase*) ou ignorar *commits* (ao copiar manualmente as contribuições), o que, por sua vez, impede a análise de contribuição casual. Muitos

[9] `https://github.com/ipython/ipython/pull/4302`
[10] `https://github.com/cocos2d/cocos2d-x/commits?author=zhukaixy`
[11] `https://github.com/fzaninotto/Faker/commits?author=paulvalla`

desses casos foram capturados pelas heurísticas aplicadas para atender PRs originalmente classificados como *unmerged*.

Os desenvolvedores podem atrapalhar: Um exemplo em que um desenvolvedor teve um PR criado na ramificação principal e foi solicitado a abertura de outro PR visando a ramificação da *release* estável atual (que também foi *merged*). Portanto, o mesmo *commit* foi submetido a dois PRs; a partir de então, quando analisado os *commits*, encontra-se apenas um, mas analisando o número de PRs *merged*, encontra-se dois. Também encontra-se contribuições copiadas em um *commit* diferente e adição manual via linha de comando.

4.4 Avaliação das heurísticas

De maneira complementar à análise de PR, foi realizada a avaliação da eficiência das heurísticas propostas por (Gousios et al., 2014), que foram utilizadas na subseção 3.1.2. A análise foi necessária para compreender qual heurística pode trazer melhores resultados par classificar os PRs. Os resultados podem contribuir para as próximas pesquisas na área.

Nesta seção é apresentada a avaliação das três heurísticas utilizadas neste trabalho: SHAs, Palavras-chave e *commits - master branch*, de forma independente e também em conjunto (combinação de heurísticas).

Os dados utilizados para avaliar as heurísticas foram os mesmos coletados na seção 3. A Tabela - 4.5 apresenta os resultados obtidos após a aplicação das heurísticas nos PRs separados pelas linguagens de programação. Observando os resultados pode-se afirmar que a heurística das palavras-chave consegue obter uma maior quantidade de falso-positivos, como por exemplo, na linguagem Javascript 4.127 PRs foram selecionados pela heurística palavras-chave. Por outro lado a heurística de *commits no master branch* obtêm poucos falso-positivos. Além disso, evidencia-se que a aplicação das heurísticas no PRs com status `unmerged` diminuem a quantidade de falso-negativos. Observando os dados da Tabela - 4.5 nota-se que os projetos desenvolvidos na linguagem de programação C++ foram encontrados mais casos com a aplicação da heurística *SHA*, enquanto os projetos desenvolvidos nas outras linguagem a heurística *palavras-chave* encontrou mais casos potencialmente classificados de forma incorreta (*unmerged*).

A heurística *commits master branch* representa um total de 6,20% dos falso-positivos encontrados, enquanto a heurística palavras-chave e a SHA representam 54,40% e 39,40% respectivamente. Apesar de apresentarem resultados que impactaram o número de PRs

Tabela 4.5: Quantidade de PRs obtidos por heurística

	# SHA	#Palavras-Chave	# Commits master branch
C	383	1.752	218
C++	3.483	1.721	247
Clojure	45	109	12
Coffeescript	147	306	47
Erlang	46	193	40
Go	170	486	112
Haskell	70	139	40
Java	1.775	2.391	91
Javascript	1.599	4.127	312
Objective-C	221	411	84
Perl	107	460	13
PHP	1.547	1.896	355
Python	1.068	968	74
Ruby	3.415	2.921	150
Scala	242	627	100
Typescript	14	103	62
TOTAL	14.332	18.610	1.957

aceitos, até esse ponto as heurísticas não foram avaliadas para saber a acurácia da sua aplicação. Como verificou-se na identificação dos casuais (Seção 4.3.4), as heurísticas apresentam falhas.

4.4.1 Análise dos Dados

Para verificar a acurácia das heurísticas utilizadas neste trabalho, foi conduzida uma segunda análise manual dos PRs, considerando os resultados de cada heurística. Primeiramente, foram selecionados todos os PRs *unmerged* que foram identificados como potenciais falso-negativos por cada heurística (exclusivamente). Com isso, as populações analisadas foram de 18.310 PRs para a heurística palavras-chave, 13.264 e 2.087 para as heurísticas SHA e *commits* no *master branch*, respectivamente. Foram amostrados PRs para cada heurística. Para todas as amostras foi utilizado o mesmo cálculo amostral, 90% de confiança e 5% de margem de erro.

Também foram analisadas as combinações das heurísticas, cominadas par-a-par e a combinação das três heurísticas. Para a análise da combinação de duas heurísticas, os dados foram filtrados com o objetivo de analisar as heurísticas encontradas em ambos os

casos. Por fim foram selecionadas todas as contribuições que foram encontradas nas três heurísticas e realizado uma análise na amostra.

4.4.2 Resultados

A análise manual relativa à heurística "palavras-chave" foi realizada em uma amostra de 269 PRs. Dentre esses, 137 PR (51%) que possuíam o status de *unmerged* foram na verdade aceitos. Um exemplo disso é o PR `https://github.com/scikit-learn/scikit-learn/pull/659`, o PR foi aceito, realizando o método *rebase*. Por outro lado 132 (49%) PRs foram classificados erroneamente pela heurística. Esses casos de falso-positivos ocorrem principalmente pelo uso de algumas palavras, como por exemplo, a palavra *"merg"*, que pode remeter a aceitação do PR, no PR `https://github.com/Homebrew/homebrew-cask/pull/11478` a palavra-chave *"merge"* presente no penúltimo comentário, indica uma possível aceitação, porém se encontra em outro contexto, onde um membro do projeto orienta o contribuidor a criar um novo PR. Essa grande quantidade de falso-positivos explicam o motivo da relativa baixa taxa de acerto da heurística, quando utilizada de forma isolada.

Na heurística SHA, foi conduzida a análise de 268 PRs que foram identificados apenas por esta heurística. Foram encontrados, 195 PRs (72%) que apresentavam o status de *unmerged*, mas na realidade, foram aceitos no projeto. Em comparação com a heurística da palavras-chave a heurística do SHA se demonstrou mais efetiva. Esse resultado indica (i) nem todos os *commits* presentes no PR são aceitos nos projetos, porém um único *commit*[12] pode ser aceito e mesclado ao projeto; (ii) mencionar o SHA nos comentários é mais efetivo do que utilizar palavras que induzem a aceitação; e (iii) o PR pode conter *commits* de outros contribuidores, como por exemplo o PR `https://github.com/TrinityCore/TrinityCore/pull/16469/commits` que foi criado por um usuário, porém o *commit* foi realizado por outro contribuidor. Os outros 72 PRs (28%) foram classificados de forma errada, pois o SHA mencionado no comentário pode pertencer a outro PR, ou pode ter sido aceito em linha de comando `git`, fora da plataforma do GitHub[13]. Nesse último caso, o dono do repositório participa dos comentários do PR, e informa o contribuidor a utilização de outro *commit* para corrigir o problema informado.

Por fim, para a heurística *commits master branch* analisou-se uma amostra de 241 PRs. Embora poucos PRs foram selecionados por esta heurística (1.957), a acurácia dela é 100%. Como a heurística determina que ao menos um *commit* presente no PR esteja

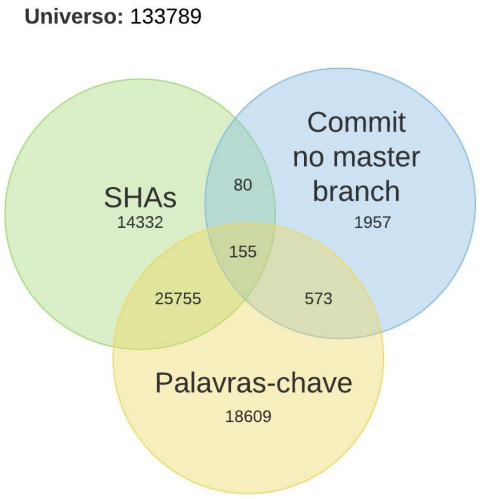
[12] `https://github.com/Homebrew/homebrew-cask/pull/21109`
[13] `https://github.com/ggreer/the_silver_searcher/pull/190`

na ramificação principal do projeto (*master branch*), o PR foi definido como contribuição aceita, considerando em nível de PR. Em um PR é possível que existam vários *commits* de vários contribuidores, como por exemplo em `https://github.com/liuliu/ccv/pull/205`, em que existem 929 *commits*[14] realizados por outros contribuidores, e o *commit* mesclado no *master branch* não foi criado pelo autor do PR. Ainda sobre este PR, é possível verificar que o autor do PR não possui nenhum *commit* no *master branch*[15]. Em outros casos, como no PR `https://github.com/php/php-src/pull/606`, o autor que criou o PR, também é o autor do *commit*[16] que foi aceito no *master branch* do projeto.

Quando analisadas as 3 heurísticas de forma independente, não há dúvidas que a heurística *commit no master branch* é a que possui melhores resultados (100%), em relação às demais. Enquanto a heurística palavras-chave apresenta uma baixa eficiência (51%), ela consegue capturar a maior quantidade absoluta de contribuições realizadas por PR.

As heurísticas também foram combinadas em pares e todas juntas para uma análise apurada, com o objetivo de saber a acurácia das combinações. A Figura - 4.8 exibe os resultados de cada conjunto e suas interseções. Observando o diagrama, nota-se que dos 133.789 PR considerados inicialmente como *unmerged*, 72.328 (54%) não foram nem selecionados por nenhuma heurística. Essas contribuições são conhecidas como quasi-contribuições. As quasi-contribuições são contribuições que foram enviadas ao projeto, mas não foram aceitas.

Figura 4.8: Diagrama de Venn dos resultados das heurísticas (Quantidade de PR)

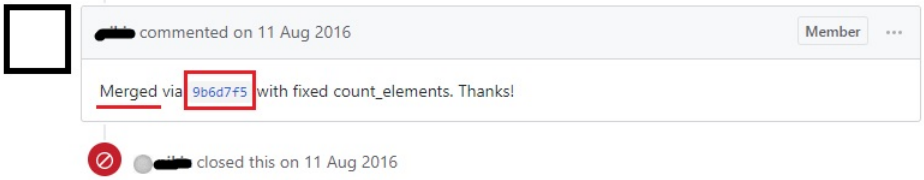


[14] `https://github.com/liuliu/ccv/pull/205/commits`
[15] `https://github.com/liuliu/ccv/commits?author=prataprish`
[16] `https://github.com/php/php-src/commits?author=DaveRandom`

Observando a Figura - 4.8 percebe-se que a combinação das heurísticas SHAs e palavras-chave capturam uma grande fatia dos PRs (25.755 de 61.461). Isso significa que por muitas vezes o SHA vem acompanhado com alguma palavra-chave (ver Figura - 4.9). Foi realizado uma análise da combinação entre as heurísticas SHA e palavra-chave. Em uma amostra de 270 PRs (5% de erro e 90% de confiabilidade), 93,4% foram classificados de maneira correta. Essa taxa de acerto, se deve a *commits* que sofreram *"rebase"*, *"cherry-picked"* ou *"squash commit"*, como por exemplo a contribuição `https://github.com/netty/netty/pull/3617` que teve seu *commit* aproveitado em outra ramificação do projeto (*Cherry-picked*). Além disso notou-se que das contribuições classificadas de forma correta (252 PRs), 82,5% apresentavam na mensagem a palavra-chave e o SHA, 16,7% apresentaram a palavra-chave na mensagem e o SHA presente no encerramento do PR e 0,8% uma mensagem com a palavra-chave e outra mensagem contendo o SHA. Enquanto os PRs classificados erroneamente, apresentaram o *commit* na mensagem que pertence a outro PR ou outro *commit*.

Combinando a heurística palavras-chave com *commits master branch* percebe-se que dos 18.690 PRs capturados pela exclusivamente pela heurística palavras-chave, 573 foram aceitos e mesclados no *master branch* do projeto (3,1%). Essa informação é importante pois demonstra que a utilização da heurística palavra-chave consegue ser efetiva em alguns casos[17], mesmo quando outras heurísticas não capturam o erro de classificação.

Figura 4.9: Exemplo da combinação das heurísticas SHA e palavras-chave



Ao observar a combinação do conjunto dos PRs identificados por meio das heurísticas SHA e *commit master branch* nota-se uma pequena quantidade de PRs (80). Isso se deve ao fato da heurística *commits master branch* identificar poucos PRs, pois a sua aplicação requer um *commit* aceito na ramificação principal do projeto (*master branch*). Dos 14.412 PR capturados pela heurística SHA, 0,55% foram identificados também pela heurística *commit master branch*. A quantidade, apesar de ser muito baixa, pode indicar que os SHAs utilizados nos comentários são provenientes do método *rebase*, *squash*, ou indicar que o *commit* foi refeito ou aproveitado em outro PR.

[17] `https://github.com/antirez/redis/pull/682`

Por fim, a combinação das três heurísticas resultou em 155 PRs identificados, esse valor representa 0,11% dos PRs detectados pelas heurísticas. Essa representação é desprezível. Um exemplo de PR capturado pelas três heurísticas é, o PR `https://github.com/spree/spree/pull/844`, em que foi utilizada a palavra-chave *"pushed"* no comentário anterior ao fechamento do PR, o SHA presente no evento de encerramento do PR, e os 2 *commits* realizados no PR foram mesclados no *master branch*. Outro exemplo é o PR `https://github.com/Homebrew/homebrew-cask/pull/8056`, no qual foi realizado *"merge"* manualmente, a palavra encontrada pela heurística foi *"merged"*, o SHA (*short link*) do *commit* esta presente na mensagem, e teve um dos seus *commits* presentes no *master branch*[18] - algo a ser ressaltado é que o PR foi aceito, porém o *commit* foi realizado por outro contribuidor[19]. Ao combinar as três heurísticas propostas por Gousios et al. (2014), a probabilidade de acerto é de 100% na identificação das contribuições.

4.4.3 Discussão

A utilização das heurísticas propostas por Gousios et al. (2014) potencialmente reduz a classificação errônea de PRs—neste caso aqueles que originalmente possuem o status de *"unmerged"* pelo GitHub. Entretanto, em alguns casos, as heurísticas também podem cometer erros. Na análise manual, percebeu-se que esses erros ocorrem principalmente pelos seguintes motivos:

- **Sobreposição**: Alguns PRs tiveram suas soluções resolvidas em outro PR como por exemplo, o PR `https://github.com/cocos2d/cocos2d-x/pull/468`, onde o problema foi corrigido no PR #474[20], ou um novo PR foi criado para resolver a *issue*, exexmplo: `https://github.com/akka/akka/pull/2128`. Além disso o PR pode ser duplicado, isso ocorre quando dois contribuidores enviam soluções para o mesmo problema. Nesses casos apenas uma solução é aceita, assim o *commit* esta presente em outro PR, como por exemplo: `https://github.com/elastic/elasticsearch/pull/8411` onde o PR # 8494 continha a mesma solução. A contribuição em um *branch* pode ser reaproveitada em outro, através do (*Cherry-pick* ou *rebased*). Algumas contribuições já haviam sido realizadas e aceitas em outros *branches* - `https://github.com/pedestal/pedestal/pull/276`, podem suprimir o PR, desse modo a contribuição aceita também pode ser utilizada em outros *branches*. Realizar

[18] `https://github.com/Homebrew/homebrew-cask/commits/master?after=b1532f4e4e451cdc7d81250e6f3ba1a091c4ec9e+69&author=tapeinosyne`

[19] `https://github.com/Homebrew/homebrew-cask/pull/8056/commits/552cb2cacd18b01d57307a6a3e9765e47e7c9788`

[20] `https://github.com/cocos2d/cocos2d-x/pull/474`

o *cherry-pick* ou o *rebase* de um *commit* agiliza na correção de problemas, pois assim o teste do código pode ser realizado apenas uma única vez. Os *commits* que passaram pelos testes podem ser utilizados em outros *branches*, de maneira segura e sem a necessidade de realizar novos testes.

- **Falta de atividade**: O contribuidor apresenta uma falta de atividade no PR. O autor do PR por muitas vezes acaba abandonando a sua solicitação e não interagindo nas conversações que envolvem o seu PR, portanto a solicitação acaba sendo encerrada por falta de atividade, um exemplo disso é: `https://github.com/matomo-org/matomo/pull/8782`, onde após um longo período de inatividade, um membro do projeto encerrou o PR. Além disso, alguns contribuidores enviam suas contribuições sem nenhuma mensagem sobre do que se trata o seu PR, e quando são solicitados por outros contribuidores, não respondem as dúvidas que surgem dos membros do projeto.

- **Diferentes visões**: Diferentes opiniões surgem quanto a solução do problema encontrado no projeto. Em geral, alguns PRs possuem uma grande troca de mensagens quanto a aceitação da contribuição ou as ideias propostas pelo contribuidor. As contribuições podem não ser aceitas de imediato, ou até que a ideia seja concretizada durante a conversação, como por exemplo o PR, `https://github.com/jquery/jquery/pull/436`, onde os contribuidores discutem uma modificação para corrigir um BUG existente[21] e encerram o PR pois é necessário pensar mais sobre a solução do problema. Muitas vezes as contribuições não são aceitas, porém, servem de base para novas contribuições. Na maioria dos casos, uma contribuição que possui divergências quanto a aceitação, geram um novo PR.

- **Erros ao enviar a contribuição**: Alguns erros são encontrados quando o contribuidor tenta depositar a sua contribuição no projeto. O problema mais comum encontrado é utilizar a ramificação errada do projeto para implantar o seu PR.

- **Problemas com o código**: Alguns projetos solicitam que os contribuidores enviem junto de suas contribuições arquivos de teste[22]. Muitos contribuidores acabam deixando de enviar esse arquivo ou até resultados de *benchmark*. O envio de código-fonte sem realizar teste pode gerar um novo BUG no projeto, isso foi notado em alguns casos, no qual o *commit* enviado acabou gerando uma nova *issue*

[21] `https://bugs.jquery.com/ticket/9381`
[22] `https://github.com/fzaninotto/Faker/pull/846`

ex: `https://github.com/scikit-learn/scikit-learn/pull/3244`. Além disso muitas contribuições diminuíram a performance do projeto[23].

- **Projetos de terceiros ou obsoletos**: A contribuição não foi aceita ou ficou em *stand-by* aguardando outras atualizações, como por exemplo, a atualização da linguagem de programação utilizada no projeto ou de ferramentas, como por exemplo *plugins*. Além disso a contribuição pode vir a se tornar obsoleta como por exemplo: `https://github.com/select2/select2/pull/2943`, onde a contribuição foi realizada em uma ramificação antiga do projeto.

Importante ressaltar que, em alguns casos, o *commit* aceito pela contribuição não pertencia ao autor que criou o PR. Porém, é possível notar que o *commit* aceito utiliza o código criado pelo autor do PR. Esses casos são mais observados na heurística *commits master branch* onde os *commits* aceitos na ramificação principal do projeto, podem ser de outros contribuidores que não seja o criador do PR.

4.4.4 Recomendações

A utilização das heurísticas propostas por (Gousios et al., 2014) e utilizadas neste trabalho, produzem um grande impacto, tendo em vista que ao utilizar as heurísticas novos contribuidores e novas contribuições por consequência (45,4%) foram e encontrados. Embora as heurística apresentem falhas, a utilização das heurísticas para capturar novas contribuições são altamente encorajadas, e algumas recomendações enquanto as suas utilizações podem ser feitas:

- **Palavras-Chave**: A heurística utiliza uma lista de palavras-chave para determinar a aceitação da contribuição, foi a que obteve o menor índice de acerto (51%). Esse baixa taxa de acerto, se deu principalmente pela palavra *"merge"*, onde a mesma pode representar a aceitação forma positiva, ou de forma negativa em alguns casos, dependendo do contexto da aplicação da palavra. Uma proposta para a melhoria desta heurística seria a aplicação de outras palavras, ou a negação de algumas palavras como por exemplo, não conter a palavra-chave *"not"*, ou outras palavras que remetem a negação da ação.

- **SHA**: Quando se compara o SHA presente no comentário a heurística obteve uma acurácia de 72%, mas ainda apresentou alguns casos onde a heurística falhou. Para uma melhor aplicação da heurística uma solução seria, considerar os SHAs presentes

[23] `http://github.com/AFNetworking/AFNetworking/pull/3260`

no evento de encerramento do PR dentro de um intervalo de tempo. Em alguns casos o SHA presente no encerramento do PR, é antigo, ou seja podendo ser de outro PR. Se for aplicado um intervalo de tempo, seria possível capturar menos PR, mas com menos falhas.

- ***Master branch***: A melhor heurística a ser utilizada, com acurácia de 100%, não apresenta falhas. Um detalhe importante nesta heurística é que alguns *commits* que foram aceitos no *master branch*, não foram criados pelo autor do PR, e sim por outros contribuidores. Dessa forma, levando em consideração o PR, pode-se afirmar que o mesmo foi aceito, contudo se for considerado o *commit* é possível dizer que a contribuição foi realizada por outro contribuidor.

- **Combinações**: combinando as duas heurísticas potencialmente mais fracas, utilizando a interseção entre SHA e palavras-chave, a taxa de acerto avaliada foi de 93,4%. Tal taxa é muito superior à taxa para as heurísticas SHA e palavras-chave separadamente, 72% e 51% respectivamente. Pode-se afirmar que a utilização da combinação das heurísticas possuem uma efetividade maior.

4.4.5 Ameaças à validade

Primeiro, o estudo está limitado a projetos de SL hospedados no GitHub. Embora foram estudados 263, não foi explorado todos os possíveis projetos de SL disponíveis online que usam o modelo *pull-based*. Além disso, tentou-se utilizar os mesmos projetos investigados no estudo original. Porém, alguns desses projetos tornaram-se inativos ou não foram encontrados. Como consequência, o número de projetos estudados difere um pouco do estudo original (275 no estudo original e 263 em desta pesquisa).

O estudo original baseou-se em dados coletados através do utilitário `git log` para conduzir sua análise. Neste trabalho, optou-se por usar a API do GitHub e o GHTorrent, que é mais confiável na identificação dos contribuidores de desambiguação. No entanto, esse benefício vem com um desafio: ao buscar dados da API do GitHub, observa-se que alguns *commits* não tinham a identificação do autor. Essa ameaça em particular aconteceu porque um contribuidor do GitHub pode fazer *commit* usando uma configuração desconhecida do `git` (o que não ajuda o GitHub a saber qual é o usuário que executou os *commits*). Para atenuar essa ameaça, foi removido os *commits* sem a identificação do usuário. Ainda com relação ao uso da API do GitHub, pode-se argumentar, no entanto, que essa decisão pode introduzir algum ruído nos dados de *commit* quando comparados aos resultados do estudo original. Porém, os resultados usando a abordagem de *commits*

estão em uma semelhança acentuada com o estudo original. Por exemplo, a porcentagem de contribuições feitas por contribuidores casuais foi de 1,73% no estudo original, e 1,83% usando a API do GitHub com dados de dois anos depois. Isso fortalece a confiança nos resultados do estudo original.

Semelhante aos projetos que se tornam inativos, os usuários do GitHub também podem se tornar inativos ou alterar seus nomes de usuário, o que pode trazer vieses para análise. Da mesma forma, antigos contribuidores podem aparecer com novas contribuições casuais. Outra ameaça está relacionada a contribuidores que abrem um PR com um usuário mas realizam o *commit* com outra conta de usuário (este problema foi reportado como "contribuidores do GitHub sem *commits*" na seção 4.3.4). Devido à escala do trabalho (ou seja, mais de 293.292 PRs e 1.028.172 *commits* foram analisados), não foi possível verificar e fornecer uma correção para essas preocupações. Ainda sobre as PRs, nota-se que, embora um PR pode ser encontrado na API, ele pode ser excluído do repositório, criando inconsistências nos dados. Por fim, os dados foram baixados até meados de setembro de 2017. Como os projetos estudados ainda estão em evolução, o número total de PRs pode aumentar durante a janela de tempo de 2018–2019. Porém, não se espera, mudanças no comportamento do contribuidor.

4.4.6 Conclusões

Para reproduzir a replicação, os mesmos objetivos e métodos foram usados para selecionar os contribuidores casuais (o método de *commits*). Além disso, foi utilizado um segundo método, o PR. O objetivo em utilizar o PR é entender se diferentes métodos (usados para o mesmo propósito) poderiam produzir os mesmos resultados. Entre os achados, é possível dizer que uso do PR é um método para encontrar contribuidores causais, onde o número de contribuidores casuais é maior do que aqueles relatados com o método de *commits*. Além disso, analisando manualmente um conjunto de contribuidores, pode-se evidenciar que o uso de PRs resultam em um número menor de falso-negativos, tornando-os mais eficiente do que os *commits* para capturar o fenômeno dos contribuidores casuais. Pode-se confirmar que o número de contribuidores casuais é não negligenciável, com uma média de 66% dos projetos analisados e atingindo mais de 90% em alguns projetos. Além disso, o número de PRs feitos pelos contribuidores casuais é de 12,5% e atinge mais de 50% em vários projetos. Comparando com a abordagem do nível de *commit* (Pinto et al., 2016), é notável um aumento tanto em termos de contribuidores quanto de número de *patches* submetidas ao analisar o fenômeno no nível de PR. Os resultados obtidos foram semelhantes ao estudo original quando analisado as categorias

das contribuições dos contribuidores casuais: a maioria das contribuições são correções de *BUGs* e documentação. Os PRs de contribuidores casuais são, em geral, menores do que os contribuidores regulares, em termos de linhas adicionadas, removidas e arquivos alterados. No entanto, os contribuidores casuais também fazem contribuições significativas.

A utilização das heurísticas são fortemente encorajadas, mesmo apresentando algumas falhas. A análise das heurísticas mostra que utilizando-as de forma separada, o seu poder efetivo é relativamente baixo nas heurísticas SHA e palavra-chave, e alto na heurística *commit no master branch*, porém, esta heurística seleciona poucas contribuições (1.957 de 133.789). A combinação das heurísticas *SHJA* e palavras-chave podem por sua vez, aumentar a acurácia e assim, obter mais contribuições de forma correta. Por fim, a combinação as três heurísticas obteve uma efetividade de 100%, devido a utilização da heurística *commit no master branch*, dessa forma é possível utilizar as heurísticas em 2 opções, (i) combinar as heurísticas em pares para obter um maior número de contribuições porém, com uma efetividade menor, ou, (ii) utilizar as 3 heurísticas combinadas e assim ter uma acurácia de 100% nas contribuições originalmente marcadas como *unmerged*.

5

Comportamento dos contribuidores casuais

Este capítulo apresenta a análise do comportamento dos contribuidores casuais em relação a sua participação nas redes sociais e atividades no projeto. Para a segunda questão deste trabalho (**"Como é o relacionamento dos contribuidores casuais com os projetos com os quais eles contribuem?"**) analisou-se o comportamento do contribuidor casual em relação ao projeto no qual ele realizou sua contribuição. Para a análise dos dados, foi utilizada estatística descritiva, por meio da qual foram analisados todos arquivos enviados nas contribuições e os tipos de mudanças mais realizadas pelos contribuidores casuais. E por fim foram analisados 3 modelos para agrupar os contribuidores de acordo com suas características.

5.1 Relacionamento com o projeto

Foram analisadas algumas características do contribuidor casual em relação ao projeto no qual ele realizou a contribuição. As características analisadas foram baseadas nas redes sociais do projeto, e as atividades que foram desenvolvidas pelo contribuidor casual, além da contribuição via PR. Essas análises são importantes compreender o relacionamento e o interesse do contribuidor com o projeto.

5.1.1 Seguir membros do projeto

A primeira característica analisada sobre o comportamento do contribuidor em relação ao projeto no qual ele contribuiu, é a participação na rede de seguidores do GitHub. Para isso, foi analisado se o contribuidor casual segue os mantenedores do projeto. O mantenedor do projeto é o usuário que é dono ou membro do projeto e integra as contribuições enviadas ao projeto.

Na plataforma do GitHub, alguns projetos são mantidos por usuários, como por exemplo, o projeto aleph[1] que é mantido pelo usuário "ztellman"; e outros são mantidos por organizações, como por exemplo, o Node[2]. Para fins de entendimento, as análises foram distintas para projetos que estão sob organizações e sob nomes de usuários.

As organizações são contas compartilhadas, nas quais empresas podem disponibilizar vários projetos (GitHub, 2019b). Contribuidores que são membros da organização são aqueles que pertencem à organização; já os membros do projeto são aqueles que são responsáveis por um projeto que é mantido por uma organização.

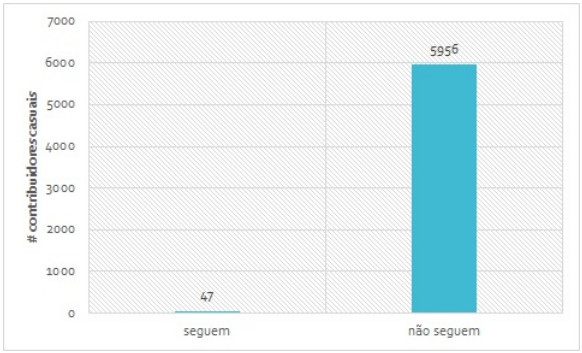
Foi analisado se o contribuidor segue algum membro do projeto/organização, ou o dono do projeto, e o tempo que o contribuidor levou para segui-los tendo como base a data da criação do PR aceito. Para isso, os contribuidores foram separados entre aqueles que contribuíram em projetos mantidos por usuários e projetos mantidos por organização. Nos projetos mantidos por organizações é possível observar se o contribuidor segue algum membro do projeto ou algum membro da organização que mantém o projeto.

A Figura - 5.1 ilustra o resultado da análise dos contribuidores casuais em relação a seguir (*followers*) os mantenedores dos projetos. Na grande maioria nota-se que os contribuidores não seguem os responsáveis pelos projetos, tanto por aqueles que são donos dos projetos (99,2%,) quanto por aqueles que fazem parte da organização ou são membros do projeto (95%).

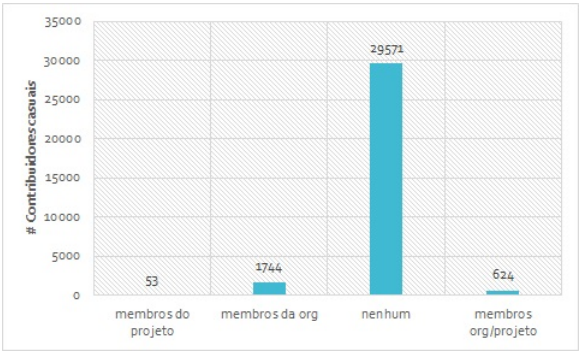
Ao todo 95% dos contribuidores não seguem o dono do projeto (ver Figura - 5.1(a)). Esse comportamento pode ser um indicador da falta de interesse do contribuidor casual com o projeto, demonstrado pela falta de interesse do contribuidor em se relacionar diretamente com o dono do projeto. O mesmo comportamento acontece com os contribuidores que seguem membros da organização/projeto (ver Figura - 5.1(b), ou seja, o contribuidor também demonstra a falta de interesse em projetos mantidos por organizações (92,44%). Poucos contribuidores seguem somente membros do projeto (0,16%), membros da organização (5,47%), ou ambos (1,95%).

[1] `https://github.com/ztellman/aleph`
[2] `https://github.com/nodejs/node`

Figura 5.1: Comportamento dos contribuidores casuais em relação a seguir mantenedores do projeto



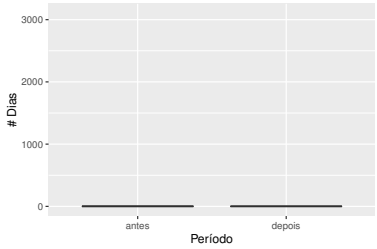
(a) Seguir o dono do projeto



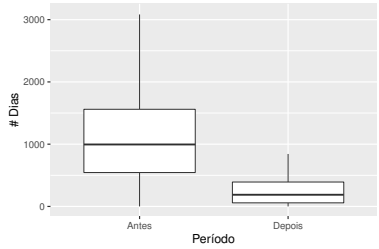
(b) Seguir os membros dos projetos

Os contribuidores analisados seguem mais membros que façam parte da organização (1.744) daqueles que são membros do projeto (53). Mais uma vez, isso pode indicar a falta de interesse no projeto. Ainda, nota-se um pequeno relacionamento com a organização, o que pode indicar que esses contribuidores podem ter laços com outros projetos da organização.

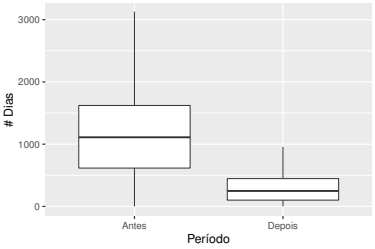
Figura 5.2: Tempo (em dias) para seguir os responsáveis pelos projetos - para uma fácil visualização os *outliers* foram removidos



(a) Seguir o dono do projeto



(b) Seguir os membros do projeto



(c) Seguir os membros da organização responsável pelo projeto

Observando os contribuidores que participaram de projetos mantidos por usuários (ver Figura - 5.2(a)), nota-se que estes começam a seguir o dono do projeto poucos dias antes da contribuição aceita (Média = 9, Mediana = 0 , Q1 = 0, Q3 = 1,5). Aqueles que começaram a seguir o dono após contribuírem, o fazem na média 21 dias após o aceite do PR (Média = 21, Mediana = 0,5, Q1=0, Q3 = 2). Percebe-se que, os poucos usuários

que seguem o dono do projeto, seguem nos dias que antecedem ou sucedem o PR, por diferença de horas (antes ou após a contribuição). Isto pode indicar baixo interesse de longo prazo pelo projeto, visto que o evento (seguir) ocorre próximo ao dia da aceitação da contribuição.

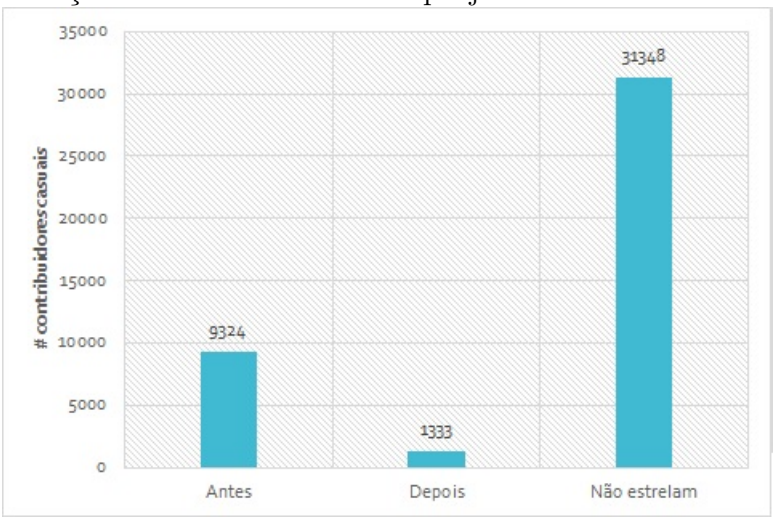
Em relação aos contribuidores que participaram de projetos mantidos por organizações, existe um comportamento semelhante entre seguir os membros do projeto e os membros da organização (ver Figura - 5.2(b) e Figura - 5.2(c)). O contribuidor casual tende a começar a seguir os membros do projeto (Média 1.096 dias) antes de seguir os membros da organização (Média 1.148 dias).

Com esses resultados pode-se afirmar que os contribuidores casuais utilizam pouco a rede social de *followers* para indicar interesse, e quando utilizam, tendem a participar antes da contribuição aceita. Um ponto interessante desta análise é que existem contribuidores casuais que demonstram interesse pelo projeto muito tempo antes da contribuição em si. Projetos poderiam usar tal tempo como indicador de interesse, e aumentar o foco em reter esses contribuidores após o primeiro PR.

5.1.2 Estrelas

A quantidade de estrelas é utilizada no GitHub como métrica para avaliar a popularidade do projeto na plataforma. Para o objetivo deste trabalho, analisou se o contribuidor casual marcou o projeto com uma estrela, e caso tenha marcado, o tempo que levou para fazer a marcação.

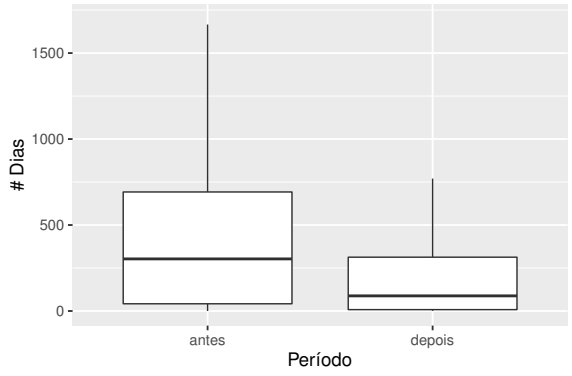
Figura 5.3: Comportamento dos contribuidores casuais nos projetos que contribuíram em relação a marcar a estrela no projeto



Observando a Figura - 5.3 nota-se que 74,6% dos contribuidores não marcaram o projeto com estrela. Essa taxa pode indicar a falta de interesse do contribuidor casual pelo projeto. Observando os contribuidores que marcaram o projeto com estrela (25,4%), notou-se que a grande maioria (87,5%) o fez antes de realizar a contribuição.

A Figura - 5.4 ilustra a distribuição em dias que o contribuidor casual levou para marcar o projeto com a estrela. Em média o contribuidor casual leva 460 dias para marcar a estrela no projeto (Mediana = 303, 1Q = 42, 3Q = 693). Essa descoberta demonstra que grande parte dos contribuidores casuais começam a se interessar pelo projeto, por períodos próximos de um ano antes da contribuição. Ao contrário do que indicam as redes de *followers*, as estrelas apontam para períodos mais longos de interesse pelo projeto antes da contribuição.

Figura 5.4: Tempo (em dias) para marcar o projeto com uma "estrela" - (*outliers* foram removidos para facilitar a visualização)

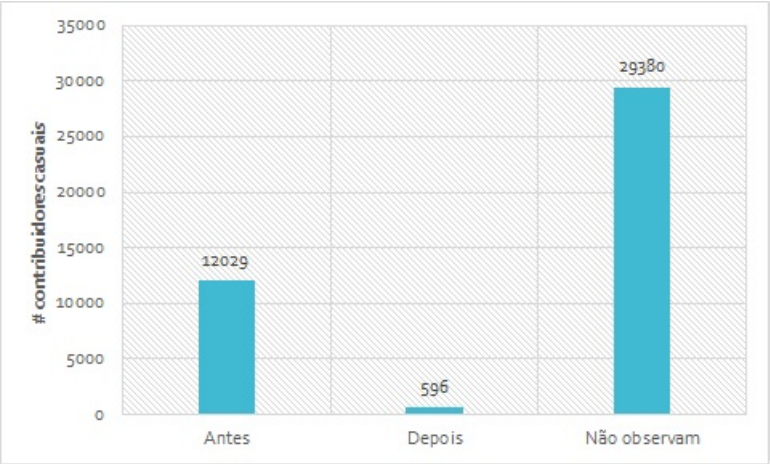


5.1.3 Observadores

Qualquer usuário da plataforma do GitHub pode observar o projeto, desde que possua um cadastro na plataforma. Quando um usuário observa o projeto, ele recebe todas as notificações do repositório, sem ter nenhum tipo de comprometimento com o projeto. Observar um repositório pode indicar interesse no projeto (GitHub, 2019e). De acordo com Sheoran et al. (2014) os contribuidores começam a observar o projeto antes de contribuir, sendo assim, a rede de observadores (*watcher*) é considerada a primeira rede social que o contribuidor participa. Os autores também afirmam que os contribuidores que entram na rede de observação possuem uma probabilidade maior de contribuir futuramente com o projeto (Sheoran et al., 2014). Assim, decidiu-se analisar o comportamento dos casuais com relação ao uso do recurso "observar".

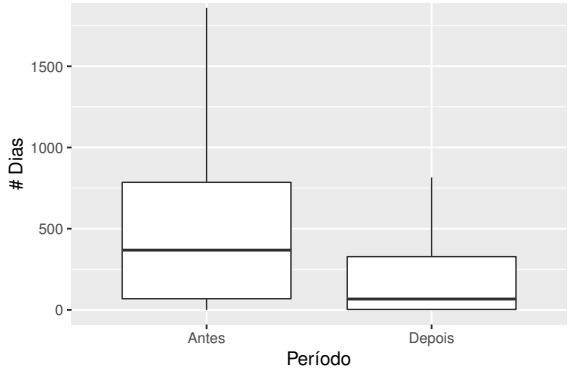
Os resultados da análise mostraram que o comportamento dos contribuidores com relação a "observar o projeto" é semelhante ao observado nas estrelas. Nesse caso, 70% dos contribuidores não observam o projeto (ver Figura - 5.5). Do total de contribuidores que observam o projeto, 95% observaram o projeto antes de realizar a contribuição. Os resultados encontrados são diferentes dos resultados encontrados por (Sheoran et al., 2014). A grande maioria dos contribuidores casuais (70%) não observaram o projeto antes de realizar a contribuição. Por outro lado, uma pequena quantidade de contribuidores observaram o projeto antes de realizar a contribuição. Essa descoberta demonstra que observar o projeto, não é garantia ou indicador de que o contribuidor permaneça no projeto.

Figura 5.5: Comportamento dos contribuidores casuais na rede social *watcher*



Foi analisado ainda o tempo (em dias) que o contribuidor levou para começar a observar o projeto. A Figura - 5.6 exibe os resultados no formato de boxplot. Nota-se que os contribuidores casuais que observam o projeto, começaram a observar muito antes de contribuir, em média 520 dias (mediana = 368, Q1 = 69, Q3 = 786). O interessante desta descoberta é que o contribuidor casual começa a ter o relacionamento com o projeto com uma média superior a 1 ano antes de fazer a contribuição. Observar o projeto antes pode ser vantajoso para o contribuidor casual, observando o projeto ele pode aprender o fluxo de trabalho e as atividades do projeto antes de realizar a sua contribuição (Sheoran et al., 2014). Por outro lado, alguns contribuidores observaram o projeto após a contribuição, em uma média inferior a 1 ano após a contribuição (média=224, mediana=67, Q1=3, Q3=328).

Figura 5.6: Tempo (em dias) para observar o projeto - *outliers* foram removidos para uma fácil visualização



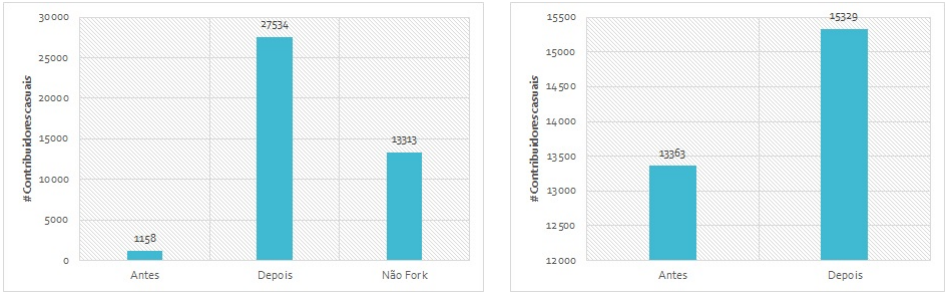
5.1.4 Forks

O *fork* é o processo de criação de uma cópia do projeto para a conta de usuário, podendo realizar modificações sem alterar o projeto original. O contribuidor cria um *fork* do projeto para propor novas mudanças ou como ponto de partida para um novo projeto (GitHub, 2019c). Quando um contribuidor cria um *fork*, pode ser um indicativo da intenção de realizar uma contribuição no projeto de origem. Sabendo disto, foi analisado se o contribuidor casual realiza o *fork* do projeto clonado, e se houve outras mudanças no repositório clonado após a contribuição aceita.

Para verificar se o projeto continuou a ser atualizado, observou-se a data do último *push* no projeto clonado. A data do último *push*, representa o último *commit* enviado para qualquer ramificação (*branch*) do clone. Essa informação é útil para saber se o contribuidor casual realizou algum *commit* no repositório clonado mesmo após a sua contribuição. Tendo em vista que o *push* antes da contribuição aceita podem ser as modificações que foram enviadas para o projeto principal via PR. Neste trabalho foi analisado o período em que o contribuidor casual realizou o *fork* no projeto, assim como criar novos *pushes* no projeto clonado.

A primeira análise realizada diz respeito à presença do *fork* do projeto que o desenvolvedor contribuiu. Analisou-se a criação de novos *pushes* no projeto clonado. A Figura - 5.7 exibe o comportamento do contribuidor em relação a esses dois fatores. Observando a realização do *fork*, (ver Figura - 5.7(a)), nota-se que os contribuidores casuais realizam em sua maioria o *fork* após a contribuição, 65,5% enquanto 2,8% fazem antes de contribuir. Esta taxa pode indicar: (i) o contribuidor casual tem a intenção de contribuir com o projeto após a contribuição aceita, (ii) realizar um novo *fork* para receber modificações feitas por outros contribuidores junto com suas modificações, dessa forma possuir o

Figura 5.7: Comportamento dos contribuidores

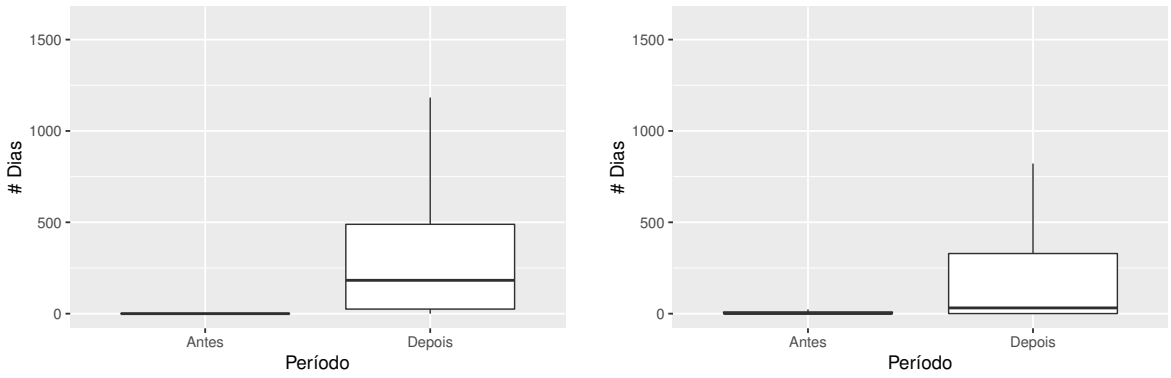


(a) *Fork* do projeto (b) *Push* no projeto clonado

projeto mais atualizado. Por outro lado, alguns contribuidores (31,7%) não apresentaram informação alguma sobre *fork* antes ou depois da contribuição. Isto pode ocorrer pelo fato do contribuidor ter excluído o projeto clonado logo após contribuir. Pode-se afirmar que os contribuidores que excluíram o *fork* não tem intenções de realizar novas contribuições no projeto a curto prazo.

Em relação a criação de novos *pushes* no projeto clonado, (ver Figura - 5.7(b)), observa-se que 53,5% dos contribuidores realizaram novos *pushes* após a contribuição. Considerando essa informação é possível afirmar que os contribuidores casuais em relação ao *push*, sinalizam uma possível contribuição no futuro.

Figura 5.8: Tempo (em dias) para realizar *fork* no projeto e para fazer (*update* ou *push*) - (os *outliers* foram removidos para uma fácil visualização)



(a) *Fork* do projeto (b) *Push* no projeto clonado

A Figura - 5.8 ilustra a distribuição em dias que o contribuidor levou para realizar o *fork* e *push*. Os contribuidores casuais realizam o *fork* praticamente no dia (diferença de poucas horas) em que a contribuição foi enviada (Média = 30, Mediana = 0, Q1 = 0, Q3 = 0). Isso pode ocorrer em alguns casos, como por exemplo, o contribuidor modificar arquivos de texto, como por exemplo, o arquivo "README"[3]. É possível modificar o conteúdo do arquivo na própria plataforma do GitHub. Ao modificar este tipo de arquivo, o GitHub cria automaticamente um PR juntamente com o *fork* do projeto na conta do usuário.

Quando um contribuidor realiza o *fork* do projeto após a sua contribuição, pode ser um indicativo da intenção de realizar novas contribuições. Analisando esse período do *fork* após a contribuição (Média = 327 Mediana = 182, 1Q = 25, 3Q = 490) observa-se que o contribuidor leva em média de 327 dias para realizar o *fork*.

É possível notar que alguns contribuidores casuais também realizaram *push* no projeto após a contribuição (ver Figura - 5.8(b)) Média = 236, Mediana = 31,5, Q1 = 0,45, Q3 = 329). Essa descoberta é interessante e mostra que o contribuidor casual realiza novas correções no projeto após contribuir por um tempo médio de 8 meses. Porém não é possível afirmar que estas correções são enviadas ao projeto original via PR.

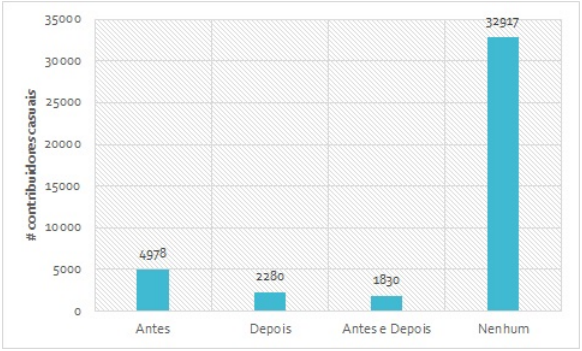
Os dados coletados referentes ao *fork* podem ser sobrescritos caso o contribuidor realize um novo *fork*, ou seja, o contribuidor pode criar um *fork* antes da contribuição, e logo após sobrescrever com um novo *fork* para manter o seu repositório atualizado. É possível que o *fork* não seja um bom indicador de interesse do contribuidor casual em relação ao projeto, pois não é possível ter todo o histórico dos *forks* realizados pelos contribuidores na plataforma do GitHub.

5.1.5 Criação de issues

Uma *issue* pode ser uma sugestão de melhoria, relatar algum defeito ou tirar dúvidas sobre o projeto (GitHub, 2019a). As *issues* podem ser associadas a um PR, podendo ser encerrada automaticamente quando o PR for *merged* - aceito. O contribuidor casual além de contribuir com o projeto por meio de PRs, também pode contribuir criando *issues*. Tal comportamento também pode indicar se os contribuidores tem interesse em permanecer próximo ao projeto. Assim, analisou se o contribuidor casual criou novas *issues* antes ou após seu PR. Para aqueles contribuidores que o fizeram, analisou-se ainda o intervalo de tempo entre o PR e a criação da *issue*. A Figura - 5.9 exibe os dados em relação a criação de *issues* por período.

[3] `https://github.com/lhartikk/ArnoldC/blob/master/README.md`

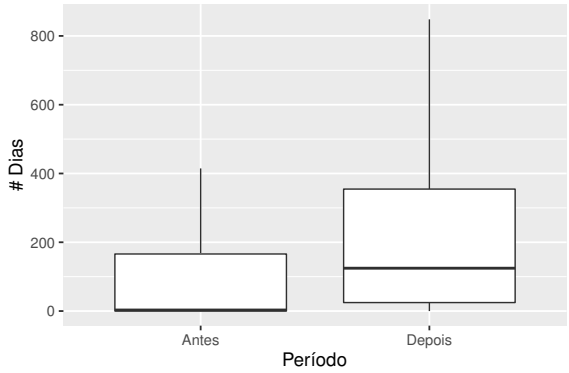
Figura 5.9: Quantidade de *issues* criadas por contribuidores casuais



Como pode-se perceber, a grande maioria dos contribuidores casuais não criam *issues* nos projetos (32.917 – 78,4%). Dentre os contribuidores que criaram, 54,8% fizeram apenas antes do PR; 25% apenas após o PR; e 20,2% criaram issues antes e após a contribuição via PR. Os resultados encontrados indicam que o contribuidor casual, mais uma vez, não se mostra (em geral) interessado em manter-se ativo no projeto. Tal comportamento é semelhante ao observado para as outras características analisadas nas seções 5.1.2, 5.1.3 e 5.1.4.

Em relação à janela de tempo (em dias) entre a data de aceitação do PR e a criação das *issues*, a Figura - 5.10 mostra que o retorno ao projeto não se dá prontamente. Mais da metade das *issues* reportadas pelos contribuidores casuais ocorre após 120 dias ($\tilde{4}$ meses) da aceitação do PR (Mediana = 125, Média = 252, Q1 = 25, Q3 = 355). Tal achado é interessante pois indica que, apesar da grande maioria dos casuais não terem interesse futuro em contribuir, aqueles que o fazem, por muitas vezes retornam após um longo espaço de tempo.

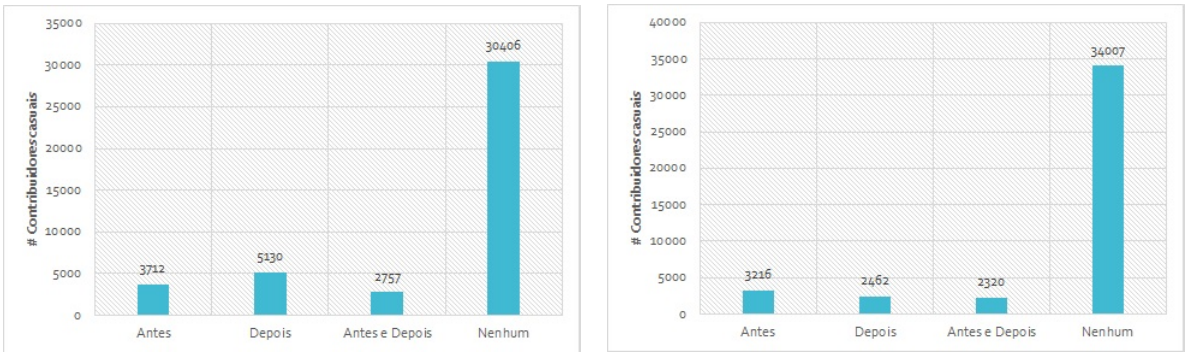
Figura 5.10: Tempo (em dias) para criar outras *issues* no projeto - *outliers* foram removidos para facilitar a visualização



5.1.6 Comentários

Qualquer usuário do GitHub pode comentar em *issues* (e PRs) em qualquer projeto, desde que possua uma conta na plataforma. Os comentários são importantes para o projeto, pois através da discussão é possível aprimorar o projeto, corrigindo problemas encontrados ou discutindo novas soluções para o problema. Neste trabalho, foram analisados os comentários realizados em PRs e em *issues* a fim de entender se contribuidores casuais permanecem interagindo com o projeto.

Figura 5.11: Comportamento dos contribuidores casuais em realizar comentários em *issues* e *pull requests*



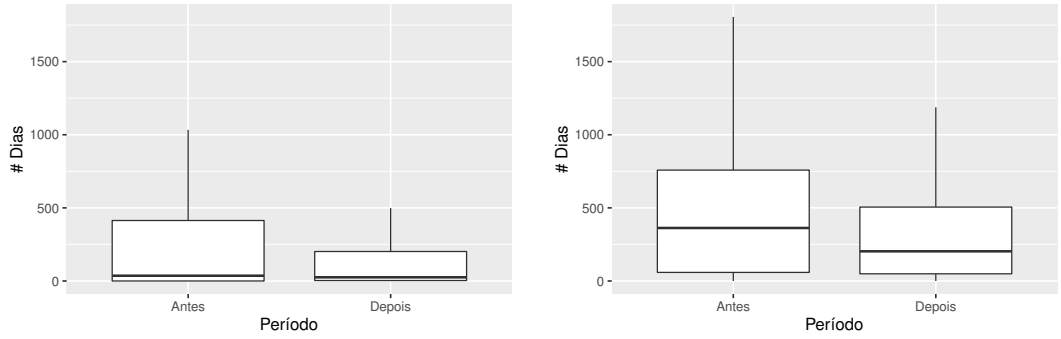
(a) Comentários em *pull requests*

(b) Comentários em *issues*

A Figura - 5.11 ilustra o comportamento do contribuidor casual em relação a realizar comentários em PRs (ver Figura - 5.11(a)) e *issues* (ver Figura - 5.11(b)). Os contribuidores casuais em sua grande maioria não realizam comentários em PRs ou *issues* em momento algum (72,4% e 81%, respectivamente). Analisando tanto issues quanto PRs, pode-se perceber que os comentários se distribuem quase igualmente entre antes e depois do PR ser aceito. Isso indica que não existe padrão que possibilite entender como os casuais se comportam com relação a comentários no projeto.

Mais uma vez analisou-se o intervalo de tempo entre a realização de comentários e a aceitação do PR do contribuidor casual. A Figura - 5.12 ilustra a distribuição dos intervalos tanto para os comentários nos períodos “Antes” do PR ser aceito e “Depois”. Pode-se perceber que os comentários realizados em PRs ocorrem, em sua maioria, em datas próximas à data do PR aceito, indicando a possível interação no próprio PR ou em PRs relacionados às mudanças realizadas. Já os comentários em *issues* aparentam ser mais distantes da data do PR aceito (observando as medianas, 277 dias antes, 203 dias depois). Assim como na criação de *issues*, observa-se que uma pequena parcela dos

Figura 5.12: Comentários realizados em *issues* por período - *outliers* foram removidos para uma fácil visualização



(a) Comentários em *pull requests*

(b) Comentários em *issues*

casuais mantém um interesse de mais longo prazo no projeto, voltando, muitas vezes, mais de um ano depois (ou aparecendo um ano antes) de sua contribuição ter sido incluída no projeto.

5.1.7 Tentativas de novas contribuições - via pull request

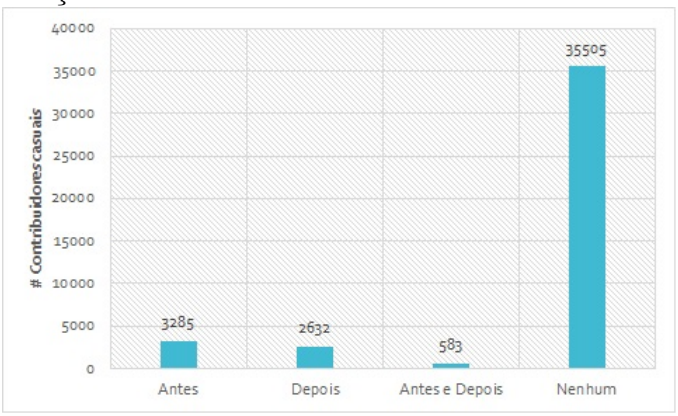
Por fim, analisou se o contribuidor casual tentou novas contribuições no projeto por meio de outro PR, antes ou após da contribuição aceita. Essa informação é relevante, pois é possível saber se o contribuidor persistiu na contribuição até conseguir, ou se tentou realizar uma nova contribuição após o êxito. Em uma visão geral, os contribuidores casuais não tentam novas contribuições, além da contribuição aceita. É possível afirmar que a maioria dos contribuidores tentou uma única vez contribuir com o projeto, e essa única contribuição foi aceita - 84,5% (ver Figura - 5.13).

Em relação aos contribuidores que tentaram novas contribuições, além da contribuição aceita, 50,5% criaram novos PRs antes da contribuição aceita; 40,5% contribuidores tentaram uma nova contribuição após. Encontrou-se ainda alguns casos de desenvolvedores que tentaram contribuir antes e depois (9% daqueles que fizeram alguma outra tentativa).

Os contribuidores que persistiram em novas contribuições (2.632), representam 6,3% do total dos contribuidores casuais - demonstrando o fraco relacionamento do contribuidor casual com o projeto. Um fato interessante é que poucos contribuidores tentaram novas contribuições após a aceitação do PR.

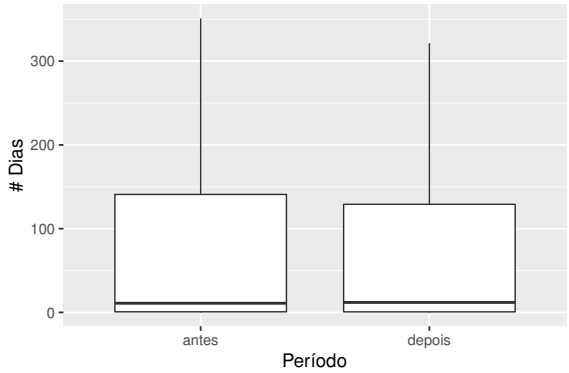
Em relação ao tempo que o contribuidor casual leva para tentar outra contribuição, a Figura - 5.14 ilustra esses dados em formato de boxplot. A grande maioria dos

Figura 5.13: Comportamento do contribuidor casual em relação tentar novas contribuições através do PR - *outliers* foram removidos para uma fácil visualização



contribuidores que tentaram outras contribuições, fizeram próximo à data do PR aceito (medianas de 12 dias para os PRs enviados depois e 11 dias para aqueles submetidos antes). Uma hipótese para esse resultado é que os desenvolvedores estavam se dedicando ao para o projeto naquele intervalo de tempo. Para os contribuidores que tiveram seus PRs rejeitados antes daquele aceito, acredita-se que foram realizadas diferentes tentativas para conseguir que a alteração fosse incluída no projeto. Já aqueles que tentaram depois, hipotetiza-se que possam ser tentativas de adicionar à contribuição recentemente aceita. Assim como nos outros itens analisados, percebe-se alguns desenvolvedores que buscaram interagir com o projeto mais de 100 dias antes e/ou depois da contribuição aceita.

Figura 5.14: Tempo (em dias) para tentar uma nova contribuição (*pull request*)



5.2 Tipos de contribuições

Além de analisar como os casuais interagem com o projeto, este trabalho se preocupou em entender o tipo de contribuição desses contribuidores. Para tanto, foram analisados os arquivos modificados em cada PR aceito que foi realizado por um contribuidor casual. Para analisar os arquivos, primeiro selecionou-se todos os PRs dos casuais. Em seguida, os PRs foram relacionados com os SHAs presentes na tabela *commit* do projeto GHTorrent[4]com o propósito de identificar os arquivos contribuídos em cada *commit*. Através do relacionamento, é possível obter o SHA do *commit* presente no PR. Para cada arquivo modificado pertencente ao *commit* do PR aceito, foram coletados: o nome do arquivo e a quantidade de linhas de código adicionadas e removidas.

Ao todo, foram coletados 4.651 PRs. Alguns *commits* não foram localizados, pois: (i) ao fazer a referência cruzada do PR com o *commit* da tabela do GHTorrent, o *commit* não foi localizado ou não continha a referência do SHA, (ii) o PR não possui nenhum *commit* (por algum problema no GitHub ou nos dados do GHTorrent). Foram analisados 352.524 arquivos.

A análise foi baseada nas heurísticas propostas por Vasilescu et al. (2014). Assim, os arquivos foram classificados conforme ilustra a Tabela - 5.1, seguindo a abordagem a seguir:

1. O primeiro passo, consistiu em classificar os arquivos específicos utilizando a extensão do arquivo, por exemplo, o arquivo "test/framework/src/main/java/org/elasticsearch/test/InternalTestCluster.java" possui a extensão `java`, foi classificado como arquivo de código-fonte da linguagem de programação "Java". A classificação de maneira específica dos arquivos foi realizada primeiramente por extensões conhecidas, e outras baseadas no trabalho de Vasilescu et al. (2014). Algumas extensões classificadas pelos autores, foram classificadas incorretamente, como por exemplo, `mp4`, que foi classificada como multimídia e também como *building*, neste caso específico, os arquivos foram considerados como multimídia. Em algumas categorias foram criadas subcategorias, como por exemplo, arquivos de código-fonte tiveram a subcategoria da linguagem de programação identificada. Um exemplo é o arquivo com extensão "java", foi classificado na subcategoria de linguagem de programação "Java".

2. Os arquivos de extensão "*.html", podem ser classificados de duas formas distintas. De acordo com Vasilescu et al. (2014) esse tipo de arquivo é definido como

[4] `http://ghtorrent.org/relational.html`

Tabela 5.1: Heurísticas utilizadas para classificar os arquivos enviados pelos contribuidores casuais. As linhas hachuradas representam categorias complementares criadas pelo autor dessa dissertação

Categoria	**Descrição**
Código-Fonte	arquivos que contém código-fonte da linguagem do projeto, ou outra linguagem de programação. Exemplos: "*.java", "*.go", "*.exe", "*.c"
Meta	arquivos pertencentes ao tipo de atividade meta não são um artefato direto dos projetos, mas suportam o processo de desenvolvimento de software. Exemplos: "*.svn", "*.git"
Building	são usados para ajudar o desenvolvedores a construir os binários a partir dos recursos disponíveis. Exemplos: "*/build/.*", "*.make"
Documentação	auxiliam o usuário final na utilização da aplicação. Exemplos: "*.pdf", "*.doc", "*.pdf"
Documentação de desenvolvimento	ajudar os contribuidores envolvidos no desenvolvimento do projeto. Exemplos: "*/changes/.*", "*/changelog/.*", "*/todo/.*"
Banco de dados	arquivos de banco de dados, utilizados pelas aplicações, como recurso de gestão de conhecimento. Exemplos: "*.db", "*.mdb", "*.json"
Testes	contém informações necessárias para automatizar o teste lógico. Exemplo: ".*/test(s)/.*"
Biblioteca	contém softwares de terceiros ou bibliotecas da própria aplicação. Exemplos: ".*/library/.*", ".*/libraries/.*"
Imagem	referente a todas as figuras utilizadas no software (ex: ícone dos botões, ilustrações em documentações). Exemplos: "*.png", "*.bmp"
Localização	adaptação do software para outra culturas, incluindo traduções para outros idiomas. Exemplos: ".*/.po(∼?)", ".*/locales/.*"
Interface do usuário (UI)	fornecer uma interface gráfica do usuário para interagir com o aplicativo. Exemplos: " *.ui", "*.xpm"
multimídia	arquivos que contenham som, vídeos e outros tipos de recursos multimídia (excluindo imagens, que possuem uma categoria separada). Exemplos: "*.mp3", "*.mp4"
Configuração	são usados pelos desenvolvedores para descrever algumas propriedades do projeto. Exemplos: ".*/.cfg", ".*/.ini", ".*/.xml"
Plugins	um plug-in é um complemento de software instalado em um programa, aprimorando seus recursos. Exemplos: ".*/plugins/.*"
Diretórios	categoria que possui arquivos cuja a extensão do arquivo possui apenas números.
Próprio projeto	extensões que só existem nos projetos. O projeto `OpenEMU`[5] é um exemplo onde foram encontrados a maioria dos arquivos dessa categoria. Foram encontradas ao menos 14 extensões de arquivos que pertencem unicamente ao projeto.
Página-WEB	arquivos com extensão "*.html", e que estão contidos dentro da pasta "src" - *source* (geralmente a pasta que contém arquivos de código-fonte da aplicação)
Exemplos	arquivos contidos em pastas que servem para tutoriais ou exemplos de como utilizar o sistema para o usuário final. Exemplos: ".*/samples", ".*/examples"

documentação. Porém, arquivos do tipo "html", podem ser arquivos de interface do usuário, por exemplo. Para a correta definição desse arquivo, foi aplicada a seguinte heurística: arquivos com extensão "html", e que estejam dentro do diretório "src", foram definidos como "Página-WEB". Após aplicar esta heurística, os arquivos restante, foram definidos como "Documentação", seguindo a classificação proposta por Vasilescu et al. (2014).

3. Algumas extensões desconhecidas foram encontradas, como por exemplo a extensão "*.zep" que foi encontrada apenas no projeto `cphalcon`[6], a extensão "CrabEmu" que pertence ao projeto `OpenEMU`[7]. Esses arquivos que foram encontrados exclusivamente em alguns projetos, foram classificados com a categoria "Próprio projeto". Além disso foi utilizado a subcategoria para definir a qual projeto a extensão pertence.

4. Após definir os arquivos com extensões conhecidas, e remove-los da lista de arquivos não classificados, a classificação seguiu de acordo com o diretório do arquivo. De acordo Vasilescu et al. (2014), o diretório do arquivo pode ajudar a classificar o tipo de arquivo. Por exemplo, arquivos que estão no diretório "src" (*source*), são arquivos de código-fonte. A categoria biblioteca foi definida apenas utilizando diretórios, pois não existe uma extensão específica para este tipo de arquivo, assim como a categoria "plugins" e "exemplos".

5. Após definir os arquivos conhecidos através das extensões ou diretórios, os arquivos remanescentes e sem categoria foram filtrados, afim de verificar quais arquivos sem classificação possuem o total de 0 modificações em linha de código (*Churn*). Esses arquivos que não tiveram modificações, foram definidos como "desconhecidos" e classificados na subcategoria "CHURN 0".

6. Por último, os arquivos que não foram classificados ou não foram possíveis de identificar corretamente, foram definidos como "desconhecidos".

A tabela 5.2 exibe o resultado da classificação dos arquivos seguindo as heurísticas e abordagem apresentadas.

Os contribuidores casuais, em sua grande maioria, contribuem com arquivos de código-fonte. As contribuições de código-fonte representam 68,7% do total de todas as contribuições casuais realizadas nos projetos. Esse resultado já era esperado, conforme já visto na seção 4.3.2. Com essa descoberta é possível afirmar que mais da metade

[6] `https://github.com/phalcon/cphalcon`
[7] `https://github.com/OpenEmu/OpenEmu`

Tabela 5.2: Resultado da análise das categorias das contribuições realizadas por casuais

Categorias	**# de arquivos**
Código-fonte	241.862 (68,7%)
Documentação	26.356 (7,48%)
Banco de dados	23.194 (6,57%)
Página-Web	16.657 (4,71%)
Imagem	6.974 (1,97%)
Desconhecido	6.527 (1,85%)
Teste	5.206 (1,47%)
Configuração	5.101 (1,44%)
Building	4.728 (1,34%)
Próprio projeto	4.697 (1,33%)
Documentação de desenvolvimento	3.583 (1,00%)
Localização	2.239 (0,63%)
Meta	1.773 (0,50%)
Biblioteca	1.050 (0,29%)
Exemplos	756 (0,21%)
Sem classificação	553 (0,15%)
Diretório	447 (0,12%)
Interface do usuário	364 (0,10%)
Multimídia	323 (0,09%)
Plugins	134 (0,03%)

das contribuições são código-fonte, reiterando estudos anteriores como o de Pinto et al. (2016), no qual afirmam que a contribuição realizada por um contribuidor casual não é trivial e vão além de simples correções de *typos* em documentação. Exemplo, o arquivo: `https://github.com/activeadmin/activeadmin/blob/master/lib/generators/active_admin/resource/templates/admin.rb.erb`, foi modificado por um contribuidor casual, no qual, 3 linhas de código forma alteradas.

A segunda categoria com maior número de arquivos alterados é documentação (7,48%). Ao menos 30 subcategorias foram encontradas para esta categoria (*docs, features, releases, pdf, css*, etc). Um detalhe interessante é que geralmente as alterações ocorrem em arquivos do tipo texto/*markdown*, como por exemplo o arquivo "README.md"[8], presente na maioria dos projetos e que podem ser alterados diretamente na plataforma do GitHub. O contribuidor pode fazer a alteração e propor a mudança do arquivo. Essas modificações criam automaticamente um novo PR dentro do projeto.

Os contribuidores casuais pouco modificaram arquivos de *layout* ou aparência do projeto, interface do usuário (UI - User Interface) - 0,10%, imagens - 1,98% ou multimídia - 0,09%. Dentre as 3 categorias, a que mais se encontrou contribuições foi a de imagens,

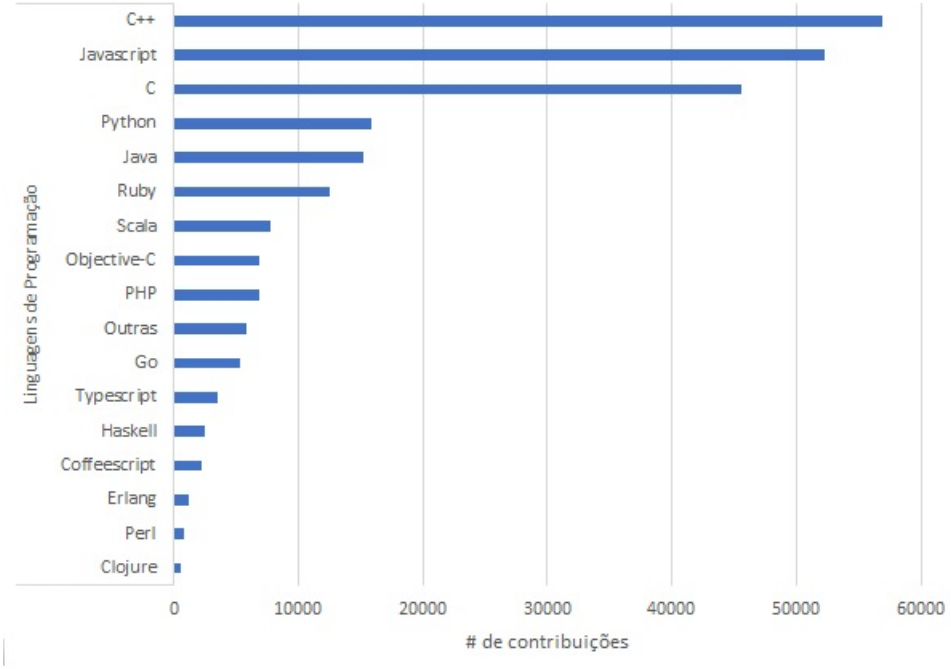
[8] `https://github.com/jquery/jquery/blob/master/README.md`

pois a mesma engloba imagens que podem ser de documentações, e também imagens da aplicação. Uma das hipóteses para este caso é que alguns projetos não possuem interface gráfica, como por exemplo, o projeto `Slim`[9].

Algumas contribuições são modificações no próprio projeto, essas mudanças representam um total de 1,33% das contribuições. Por exemplo, o arquivo `https://github.com/fish-shell/fish-shell/blob/master/share/completions/npm.fish` possui a extensão "fish" que são usadas exclusivamente pelo projeto `fish-shell`[10]. Essas contribuições são interessantes pois demonstram o conhecimento do contribuidor casual com o projeto no qual ele contribui. Contribuir com um arquivo do próprio projeto demonstra um envolvimento de longo prazo.

Em relação às contribuições na categoria de código-fonte, foi analisado qual a linguagem de programação mais utilizada pelos contribuidores casuais, a Figura - 5.15 exibe o resultado dessa análise.

Figura 5.15: Contribuições em código-fonte por linguagem de programação



Dentre os projetos analisados, a linguagem de programação em que mais código foi encontrado foi C++ (23,5%), seguida por Javascript e C, enquanto a menos encontrada foi Clojure (0,22%). Os valores encontrados tem forte correlação com a quantidade de *commits* aceitos pelos projetos escritos em cada linguagem (Tabela - 3.2), sendo o

[9] `https://github.com/slimphp/Slim`
[10] `https://github.com/fish-shell/fish-shell`

coeficiente de correlação de Pearson igual a 0.75–correlação alta (Swinscow et al., 2002). O coeficiente de correlação pode ser interpretado como, 0 a 0.3 (desprezível), 0.3 a 0.5 (fraca), 0.5 a 0.7 (moderada), 0.7 a 0.9 (forte), 0.9 a 1 (muito forte) para valores positivos e negativos.

A diferença entre C++ e Clojure, pode ser pela idade das linguagens. Enquanto C++ tem 40 anos desde a sua criação, Clojure tem 12 anos sendo uma linguagem relativamente nova, dessa forma, os projetos que utilizam Clojure são mais recentes, por exemplo, o projeto `instaparse`[11] que foi criado há 7 anos, utiliza Clojure como a principal linguagem de programação. Outras linguagens de programação (além das 16 utilizadas neste trabalho), também foram encontradas, além de outros arquivos que foram definidos como código-fonte, exemplo: "*.sh", "*.bat", etc.

Outra observação interessante é que alguns projetos utilizam outras linguagens de programação além da linguagem principal. Analisando as contribuições realizadas em projetos cuja a principal linguagem de programação é Typescript, 52% das contribuições são realizadas em outras linguagens, enquanto 48% são feitas em Typescript. Por outro lado, projetos escritos originalmente em C++, possuem 62% do seu código escrito na linguagem principal e 37% escrito em outras linguagens. Um exemplo disso é o projeto `shumway`[12] que originalmente é escrito na linguagem de programação Typescript, e possui contribuições de arquivos de código-fonte escritos em Javascript[13]. Já os projetos escritos na linguagem Haskell, tiveram 96% dos códigos escritos na linguagem principal do projeto, e 4% foram escritos em outras linguagens. Por exemplo o projeto `purescript`[14] escrito em Haskell, possui arquivos de outras linguagens como o arquivo `index.js`[15].

Por fim, foi analisado quais tipos de arquivos o contribuidor modificou em sua contribuição, como por exemplo, o *commit* do PR pode ter arquivos do tipo código-fonte e arquivos de teste. A Tabela - 5.3 apresenta a quantidade de tipos de arquivos (categorias) que foram modificados por contribuição.

Observando a Tabela - 5.3 nota-se que a maioria dos PRs (69%) continham apenas 1 tipo de arquivo, enquanto 0,5% das contribuições continham todos os tipos de arquivos das categorias analisadas. Das contribuições que continham apenas 1 tipo de arquivo, 70,7% eram arquivos de código-fonte, 0,01% UI, 0,03% localização, 16,5% documentação, 2% banco de dados, 0,1% bibliotecas, 1% *building*, 0,9% configuração, 0,09% diretórios, 1,87% documentação de desenvolvimento, 0,09% exemplos, 0,09% imagens, 0,31% meta, 1,7%

[11] `https://github.com/Engelberg/instaparse`
[12] `https://github.com/mozilla/shumway`
[13] `https://github.com/mozilla/shumway/blob/master/src/htmlparser.ts`
[14] `https://github.com/purescript/purescript`
[15] `https://github.com/purescript/purescript/blob/master/app/static/index.js`

Tabela 5.3: Quantidade de categorias modificadas por contribuição

Tipos diferentes de arquivos na mesma contribuição	# de PRs
1	3.208
2	598
3	198
4	119
5	85
6	92
7	92
8	84
9	73
10	44
11	33
12	25
TOTAL	**4.651**

plugins, 0,9% próprio projeto, 0,1% testes e 2,68% desconhecido. Um fato importante a ser ressaltado é que as contribuições que continham somente código-fonte em sua maioria foram feitas com ausência de arquivos de teste. Sabendo disso, foi analisado as contribuições que alteraram 2 tipos de arquivos. Foram analisadas 2 combinações: código-fonte + teste e código-fonte + documentação. As duas combinações foram analisadas por serem as principais em projetos de software, pois envolvem documentações e testes. Das 598 contribuições que continham duas categorias de arquivos, 261 (43,7%) tiveram código-fonte e documentação, e 26 (4,35%) com código-fonte e teste. Essas descobertas são interessantes pois demonstram que muitos contribuidores não enviam arquivo de teste junto com suas contribuições, podendo assim, gerar novos *bugs* através de suas contribuições.

5.3 Agrupamento dos contribuidores casuais de acordo com o relacionamento com o projeto

Ainda buscando entender o perfil do contribuidor casual, foram realizadas algumas tentativas de classifica-los automaticamente. O objetivo era saber se haviam diferentes "tipos" de contribuidores, que poderiam ser identificados com base na forma como os casuais interagem com o projeto. Para isso, foram utilizados diferentes métodos de agrupamento (*clustering*). Foram utilizados como entrada os dados discutidos anteriormente,

a saber: criação novas *issues*, comentários em outros PRs ou *issues* e tentativa de novas contribuições (PRs). As informações relativas às redes sociais: seguir membros/dono dos projetos, marcar o projeto com "estrela", observar o projeto, criar *fork* ou manter o repositório clonado atualizado não foram utilizadas por serem valores dicotômicos (sim/não). A Tabela - 5.4 exibe um exemplo considerando parte dos dados utilizados para essa análise. Para o exemplo da tabela, os valores de cada característica são definidos pela quantidade de eventos. Por exemplo, o "contribuidor 1" realizou 1 tentativa de contribuição antes da contribuição aceita e nenhuma após, 1 comentário antes e 3 após e não criou nenhuma *issue* no projeto `YouCompleteMe`[16].

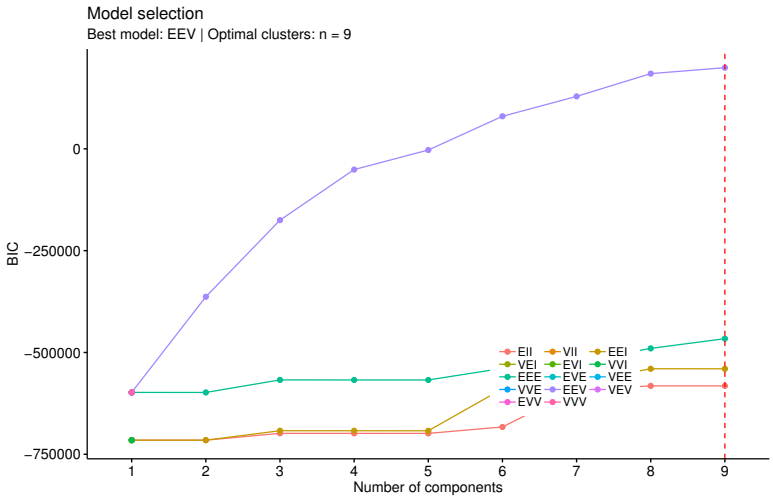
Tabela 5.4: Exemplo dos dados utilizados para gerar o agrupamento

Contribuidor	Projeto	# tentativas		# comentários		# issues	
		antes	depois	antes	depois	antes	depois
1	YouCompleteMe	1	0	1	3	0	0
2	youtube-dl	1	0	0	0	3	0
3	redis	0	0	3	3	0	0
4	GPUimage	0	0	0	0	0	0
5	nimbus	0	0	0	0	1	0
6	youtube-dl	0	0	4	0	0	0
7	awesome-python	0	0	0	0	1	0

A primeira tentativa de classificação foi utilizando o algoritmo de agrupamento conhecido como *Model-based Clustering* (Fraley e Raftery, 2002). O algoritmo foi escolhido por não necessitar de uma parametrização, por exemplo, informar a quantidade de grupos que devem ser gerados. A única informação necessária é o conjunto de dados de cada observação para gerar os agrupamentos. O algoritmo recebe como entrada, os dados de uma matriz, do mesmo modo em que os dados deste trabalho foram separados para esta análise. Após executar o algoritmo na ferramenta RStudio[17], utilizando o pacote *MClust*[18], observou-se que a cada execução do algoritmos os resultados eram diferentes. Os resultados variaram entre 2 e 9 grupos gerados. A Figura - 5.16 exibe o gráfico gerado pela ferramenta exibindo o melhor resultado e também o melhor modelo (**EEV** - Volume = **E**qual, Forma = **E**qual e Orientação = **V**ariable) para a quantidade de *clusters*(grupos) "K = 9".

[16] `https://github.com/ycm-core/YouCompleteMe`
[17] `https://rstudio.com/`
[18] `https://cran.r-project.org/web/packages/mclust/index.html`

Figura 5.16: Gráfico dos do Melhor modelo do MCLUST



A Figura - 5.17 exibe os *clusters* após a execução do algoritmo. Nota-se que os *clusters* são próximos, sendo difícil separa-los visualmente. Alguns componentes de alguns grupos se encontram mais afastados de outros do próprio grupo. Em uma breve análise, notou-se que os grupos são próximos quando checou-se as características analisadas, sendo difícil caracterizar cada grupo com precisão. A Tabela - 5.5 exibe os centroides de cada grupo pelas características utilizadas pelo algoritmo. Os valores dos centroides indicam a diferença para a média da característica, quando o valor é positivo, esta acima da média.

A Tabela - 5.6 exibe a quantidade de grupos criados pelo algoritmo e quantos componentes (contribuidores casuais) estão presentes em cada grupo:

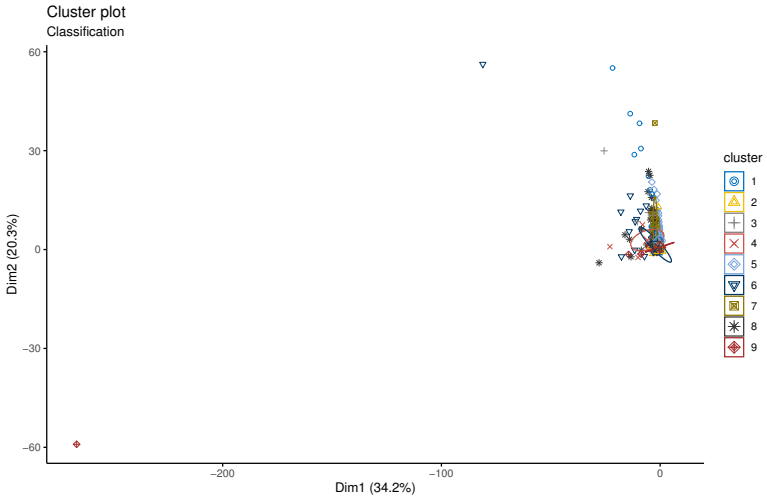
Tabela 5.5: Centroides dos grupos gerados pelo algoritmo Model-based clustering

	#1	#2	#3	#4	#5	#6	#7	#8	#9
Tentativas antes	-0,25	0,13	2.20	2,88	1,87	-0,27	2,46	-0,27	-0,27
Tentativas depois	2,42	-0,23	-0,23	2,37	3,81	-0,21	8,53	-0,23	2,11
Comentário antes	-0,00	-0,01	0,08	3,08	0,00	0,17	0,57	-0,10	0,12
Comentário depois	0,13	-0,02	0,09	4,83	0,02	0,00	0,21	0,00	0,14
Issue antes	0,34	-0,03	1,18	0,18	0,04	0,67	3,63	-0,03	-0,04
Issue depois	1,85	-0,16	1,55	3,03	0,26	-0,13	3,00	0,09	-0,16

Tabela 5.6: Quantidade de *clusters* (grupos) gerados pelo algoritmo do Mclust

# Grupo (*cluster*)	# Contribuidores casuais
1	537
2	34.616
3	365
4	9
5	1.022
6	143
7	19
8	3.620
9	1.674

Figura 5.17: Gráfico dos *clusters* (grupos gerados) pelo algoritmo *MClust*

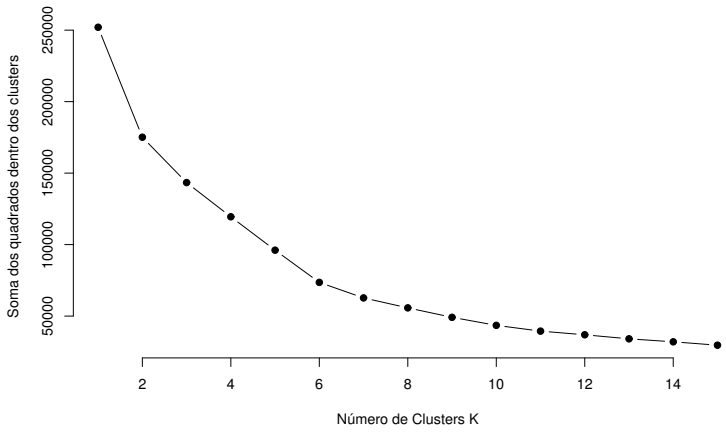


Após verificar que o algoritmo *Model-based clustering* não obteve resultados satisfatórios, uma nova abordagem foi utilizada para classificar os contribuidores casuais em grupos: o algoritmo *K-means*. O *K-means* é um algoritmo utilizado para mineração de dados, funcionando de forma similar ao algoritmo de agrupamento *Model-based clustering*, ou seja, baseado em análise e comparações entre os valores numéricos dos dados. Porém para ser utilizado é necessário informar a quantidade de *clusters* (grupos) pretendida. Para saber a "melhor" quantidade de *clusters*, foi necessário realizar o método conhecido como "cotovelo" - (*Elbow-method*).

O método do cotovelo examina a porcentagem de variação explicada como uma função do número de *clusters*: para tal deve-se escolher um número de *clusters* para que a adição de outro *cluster* não forneça uma modelagem muito melhor dos dados. Mais precisamente, se alguém traçar a porcentagem de variância explicada pelos *clusters* em relação ao número

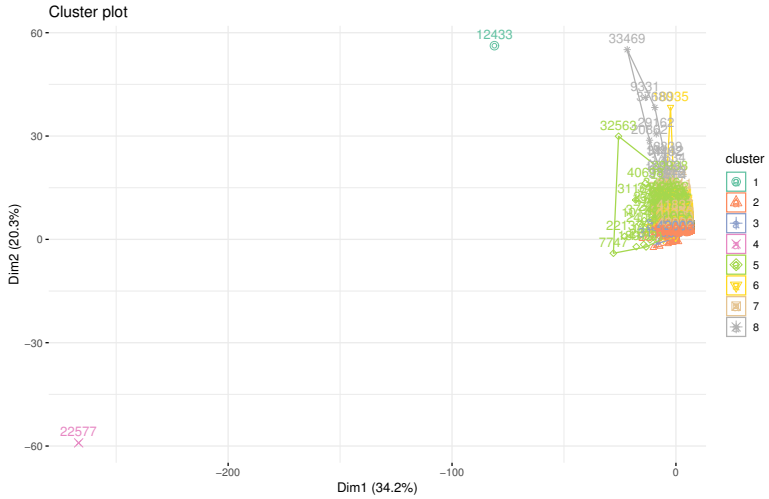
de *clusters*, os primeiros *clusters* adicionarão muita informação (explicam muita variação), mas em algum ponto o ganho marginal cairá, dando um ângulo no gráfico. O número de *clusters* é escolhido neste ponto, daí o "critério do cotovelo" (Ketchen e Shook, 1996). A Figura - 5.18 exibe o gráfico gerado pelo método. Observando o gráfico nota-se que a partir de 8 *clusters* a diferença começa a cair.

Figura 5.18: Método *Elbow* para determinar a melhor quantidade de *clusters* utilizando o *K-means*



Sendo assim, foi determinado que a quantidade "ótima" de *clusters* para o algoritmo *K-means* através do método cotovelo é 8. O algoritmo K-means foi executado, então, com a quantidade de *clusters* = 8. Após a execução, ficou constatado que também não é possível determinar grupos com características claramente distintas (ver Figura - 5.19).

Figura 5.19: Agrupamento dos *clusters* utilizando o algoritmo *K-means*



A Tabela - 5.7 exibe os centroides de cada grupo pelas características utilizadas pelo algoritmo. Os valores dos centroides indicam a diferença para a média da característica, quando o valor é positivo, esta acima da média.

Tabela 5.7: Centroides dos grupos gerados pelo algoritmo K-means

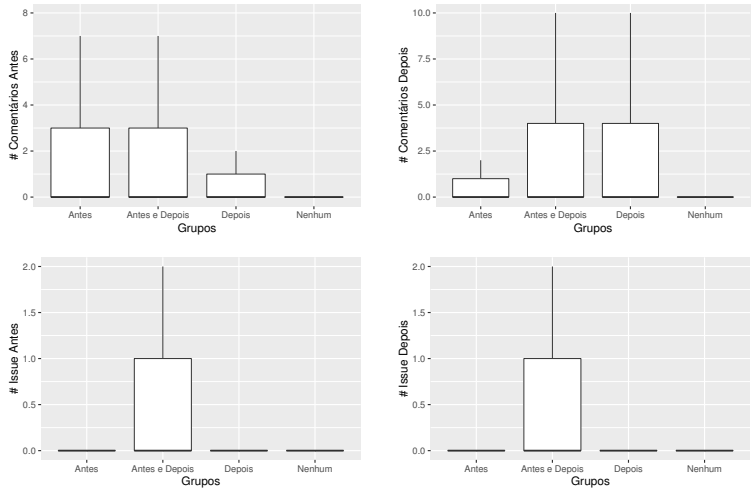
	#1	#2	#3	#4	#5	#6	#7	#8
Tentativas antes	-0,27	-0,27	2.09	-0,27	0.17	-0,17	5,56	1,07
Tentativas depois	-0,23	-0,23	0,02	-0,23	0,40	2,85	0,45	1,27
Comentário antes	47,66	-0,01	0,01	193,73	0,44	-0,00	0,02	0,06
Comentário depois	1,70	-0,01	-0,00	197,40	0,72	0,03	0,01	1,05
Issue antes	153,61	-0,03	0,44	20,24	2,21	-0,00	0,19	6,40
Issue depois	2,17	-0,07	-0,02	1,39	5,44	0,11	0,09	37,12

Nenhum dos algoritmos utilizados alcançaram resultados satisfatórios, pois os grupos se apresentavam muito próximos. Portanto uma terceira tentativa foi realizada para classificar os contribuidores casuais. A nova abordagem foi separar os grupos manualmente. Os contribuidores casuais foram separados em 4 grupos: “Antes”, “Depois”, “Nenhum” e “Antes/Depois”. Os grupos foram baseados nas tentativas de submissão de outros PRs antes e após o PR aceito. Para o grupo “Antes”, foram selecionados os contribuidores que tentaram realizar PRs apenas antes daquele aceito; o grupo “Depois”, era composto de contribuidores que tentaram submeter outro PR apenas após aquele aceito. O grupo “Nenhum” foi criado com os contribuidores que não tentaram nenhuma outra vez além do PR aceito. Por fim, o grupo “Antes/Depois”, continha com os contribuidores que tentaram submeter PRs antes e depois daquele aceito.

A Figura - 5.20 exibe a distribuição das características, comentários e *issues* nos grupos criados. Nota-se que o Grupo “nenhum” pouco participa do projeto, quanto a criar novas *issues* ou novos comentários. Enquanto os outros 3 grupos, se apresentaram próximos sendo difícil separa-los de maneira correta utilizando essas características.

Observando a Figura - 5.20 nota-se que apenas o grupo “Antes/Depois” se difere dos demais em relação à quantidade de *issues* criadas pelos membros dos grupos. Um fato interessante em relação aos comentários, é que os membros do grupo “Depois” realizam comentários antes da contribuição, enquanto os membros do grupo “Antes” também criam comentários após a contribuição, mas em menos quantidade do que aos membros do grupo “Depois”. Analisando os resultados obtidos foi necessário realizar um novo teste para saber a acurácia do método proposto. Para isso, foi aplicado uma técnica estatística

Figura 5.20: Características por grupo - *outliers* foram removidos para uma fácil visualização



conhecida como Regressão Logística Multinomial. A técnica tem o objetivo de produzir um modelo que realize predições de valores baseados em uma variável.

A regressão logística multinomial é usada para prever a probabilidade de uma variável dependente, ser utilizada como base para múltiplas variáveis independentes (Starkweather e Moske, 2011). A variável dependente escolhida foi a quantidade de tentativas de novas contribuições. O modelo proposto foi usado para predizer se a tentativa de contribuição pode definir um perfil do contribuidor casual com base em outras características. Para utilizar a técnica é necessário informar a variável dependente (tentativas de PR), e também informar quais são os fatores que podem predizer essa variável. Foram utilizados os seguintes fatores: quantidade de comentários antes e depois, e quantidade de *issues* antes e depois. As quatro variáveis foram escolhidas pois já foram estudadas neste trabalho (ver seções 5.1.5 e 5.1.6) e assim é possível ter um prévio conhecimento de seus valores.

A técnica foi aplicada utilizando a ferramenta RStudio, com o propósito de analisar os resultados. O modelo proposto foi baseado no grupo "Antes / Depois" como grupo de referência para executar o método. Esse grupo foi escolhido por ser o que possui menos contribuidores, porém qualquer outro grupo pode ser utilizado como referência. A Tabela - 5.8 ilustra os resultados obtidos após a execução do método. Os resultados em relação ao fator **comentários antes da contribuição**, indicam que a cada novo comentário realizado, por um contribuidor do grupo de referência ("Antes / Depois"), a possibilidade de pertencer aos grupos: "Depois" diminui em 0,36%; do grupo "Antes" aumenta em 0,20%; e do grupo "Nenhum" aumenta em 0,44%. Porém esse, fator (comentários antes)

é considerado não significante para determinar a probabilidade de pertencer aos grupos Depois e Antes (*p-values* = 0,09 e 0,28, respectivamente). Por outro lado, esse fator é significante para determinar a probabilidade de pertencer ao grupo Nenhum (*p-value* (0,001) < 0,05).

Os resultados obtidos para os comentários realizados depois, demonstram que caso o contribuidor realize mais 1 comentário após a contribuição aceita, a probabilidade dele pertencer ao grupo "Depois" sobe em 0,35% enquanto a probabilidade de pertencer ao grupo "Antes" diminui em 0,49% e a probabilidade de pertencer ao grupo "Nenhum" diminui em 0,79%. Comparando os resultados entre o fator de comentário nos 2 períodos (antes e depois), para o Grupo "Nenhum" o fator é significante em ambos os períodos, porém para os outros grupos esse fator é não significante, ou seja, esse fator não é relevante para o modelo. O modelo apresentou também um valor que diverge dos resultados esperados, como por exemplo, criar um comentário antes, aumenta a probabilidade de pertencer ao Grupo "Nenhum", enquanto ao comentar depois a probabilidade é reduzida.

Tabela 5.8: Regressão Logística Multinomial - Em relação ao grupo Antes/Depois - Fator: Comentários

Grupos	Antes		Depois	
	Coeficiente	*p-value*	Coeficiente	*p-value*
Depois	-0,36%	0,09	0,35%	0,13
Nenhum	0,44%	**0**	-0,79%	**0**
Antes	0,20%	0,28	-0,49%	0,11

A Tabela - 5.9 exibe os resultados da regressão logística multinomial para o fator de criar novas *issues*, separado pelos períodos antes e depois. Todos os resultados diminuem as probabilidades em todos os grupos, por exemplo, a cada *issue* criada antes da contribuição aceita por algum membro do grupo "Antes/Depois", diminui a possibilidade de pertencer ao grupo "Antes" em 0,39%, porém esse fator se demonstrou não significante (*p-value* = 0,71).

Existem várias formas de saber o ajuste do modelo de regressão logística. Neste trabalho o modelo de ajuste escolhido foi o modelo de *McFadden's Pseudo R2* que utiliza o método da máxima verossimilhança (McFadden, 1974). Quando o valor do teste de verossimilhança esta entre 0,2 e 0,4 o modelo é considerado excelente (McFadden, 1977). Para o modelo proposto neste trabalho, quantidade de tentativas, dependendo das variáveis de (quantidade de comentários+quantidade de *issues*), o resultado do ajuste do modelo foi de 0,007. Assim o resultado obtido é considerado ruim para o modelo. Pode-se

Tabela 5.9: Regressão Logística Multinomial - Em relação ao grupo Antes/Depois - Fator: *Issues*

Grupos	Antes		Depois	
	Coeficiente	*p-value*	Coeficiente	*p-value*
Depois	-1,97%	0,16	-1,38%	0,29
Nenhum	-5,67%	**0**	-15,55%	**0**
Antes	-0,39%	0,71	-5,67%	**0**

afirmar que o modelo proposto não é recomendado para predizer a participação de um membro nos grupos propostos.

5.4 Discussão

Os resultados em relação ao relacionamento do contribuidor casual com o projeto no qual ele contribuiu, de acordo com as características analisadas (seguir os donos/membros do projeto, marcar o projeto com uma estrela, observar o projeto, comentar em *issues* do projeto, criar novas *issues* e tentar outras contribuições via PR) apontam a falta de interesse do contribuidor com o projeto. Analisando os resultados obtidos nas 3 redes sociais: seguidores (*followers*), estrelas (*stars*), observadores (*watchers*), notou-se que os contribuidores casuais, em sua maioria, não utilizam as redes sociais do GitHub com o projeto no qual contribuíram, demonstrando a falta de interesse no projeto. Quando utilizam tais redes, o relacionamento se dá, na sua maioria, antes dos contribuidores terem seus PRs aceitos no projeto. Uma descoberta, que diverge dos estudos de Sheoran et al. (2014), é que a primeira rede social que o contribuidor começa a participar é a rede de observar o projeto, porém, conforme os resultados obtidos neste trabalho, poucos contribuidores observam o projeto antes de contribuir. Poucos contribuidores mantém a cópia do projeto *fork*, sendo que, aqueles que mantém a cópia fazem por aproximadamente 1 ano. Pode-se afirmar que o contribuidor casual realiza poucas tarefas de outra natureza no projeto (e.g., comentar em outras *issues* ou PRs, assim como criar novas *issues* e PRs), o que pode demonstrar falta de interesse de longo prazo no projeto. Ainda assim, em alguns casos alguns contribuidores continuam participando do projeto, porém são a minoria. Muitos contribuidores acabam abandonando o projeto por uma série de fatores, como tempo, não ter mais nada a contribuir, falta de conhecimento, barreiras encontradas (Balali et al., 2018; Lee et al., 2017; Pinto et al., 2016; Steinmacher, 2015).

Em relação ao tipo de contribuição realizada pelo contribuidor casual, em sua maioria são modificações que alteram o código-fonte (68,7%). Esse resultado pode estar relacionado ao que se encontrou no estudo de Pinto et al. (2016), em que grande parte dos casuais reportou a motivação de *"scratching an itch"*. Isso significa, em geral, corrigir uma falha no sistema que está impedindo o desenvolvedor ou construir uma nova funcionalidade que o desenvolvedor necessita. Em geral essas correções são em nível de código-fonte. Além disso, descobriu-se que os contribuidores casuais alteram a documentação do projeto com certa frequência, podendo essa documentação abranger vários tipos de arquivos (PDF, DOC, README, etc.).

Não foi possível identificar diferentes tipos de contribuidores casuais baseado nas características de interação com o projeto. Os critérios utilizados e analisados neste trabalho não foram suficientes para definir grupos distintos um do outro. As tentativas de criar um perfil do contribuidor casual através de algoritmos de agrupamento *Model based clustering e K-means* falharam ao apresentar os grupos de contribuidores muito próximos, ficando impossível determinar a diferença entre eles. Além da tentativa dos grupos pelos algoritmos, a tentativa de agrupá-los de forma manual, também falhou. Foi aplicado a técnica estatística Regressão logística multinomial para saber a efetividade do modelo proposto. Os resultados do teste demonstram que o modelo proposto não foi o melhor modelo.

Uma possibilidade é que, não existam contribuidores casuais suficientes que executem outros tipos de tarefa (revisar, comentar, etc.) para caracterizar um grupo. Visualiza-se como trabalho futuro, entretanto, analisar outros aspectos do perfil dos casuais, como por exemplo, o comportamento em outros projetos.

5.5 Ameaças à validade

Algumas ameaças que foram destacadas na seção 4.4.5, também se aplicam ao estudo reportado neste capítulo. Os dados analisados neste capítulo foram obtidos por meio de relacionamento de tabelas do GHTorrent e a tabela dos PRs coletados. Esse relacionamento, principalmente com os dados dos usuários do GHTorrent, podem ser afetados caso o autor do PR altere (ou remova) o nome de usuário, impedindo o relacionamento do autor do PR com o usuário do GitHub. Outra ameaça está relacionada aos dados coletados que não foram contemplados pelo GHTorrent. Foi necessário coletar as informações de comentários dos PRs. Alguns comentários não foram coletados, por falha na API do GitHub ou inexistência de comentário. Essa ameaça ocorreu principalmente pela exclusão do PR por membros do projeto. Os comentários foram coletados até meados de Agosto

de 2018. Após observar manualmente os dados, nota-se que o impacto dessa ameaça é pequena e não interfere nos resultados obtidos. Quanto à classificação manual dos tipos de arquivos modificados pelos contribuidores (seção 5.2) por ser uma classificação manual, algumas falhas humanas podem ocorrer. Além da falha humana é possível que alguma heurística proposta por Vasilescu et al. (2014) esteja incorreta. Para reduzir essa ameaça, todos os resultados foram checados por, pelo menos, mais um pesquisador.

6

Conclusão

O objetivo deste trabalho é explorar o interesse e as interações do contribuidor casual com o projeto no qual ele realizou sua contribuição. Para tal, foram analisados as características da contribuição e o comportamento nas redes sociais na plataforma do GitHub.

Antes de analisar os contribuidores, foi replicado o estudo de (Pinto et al., 2016) a fim de definir como se daria a identificação dos casuais. O trabalho realizado por Pinto, seleciona os contribuidores casuais pela quantidade de *commits*. Foi avaliado o método utilizado por (Pinto et al., 2016), e outro método que utiliza o PR como unidade para selecionar os contribuidores, em conjunto com as heurísticas propostas por Gousios et al. (2014) para identificar PRs com status não condizente com o real. O método escolhido para o trabalho foi a utilização de PRs, que possibilitou encontrar 30% mais contribuidores casuais em relação ao método do *commit*. As características (# de arquivos alterados, # linhas adicionadas, # de linhas removidas e # de commits - ver Figura - 4.6) das contribuições utilizando os dois métodos são semelhantes. Alguns cuidados devem ser tomados para utilizar o método do PR, sabendo que alguns projetos não utilizam o PR como forma de contribuir. As heurísticas propostas foram analisadas para saber a eficiência de cada uma delas. Embora as heurísticas apresentem falhas, são fortemente encorajadas a serem utilizadas. Recomenda-se a utilização das heurísticas em 2 casos: (i) encontrar mais contribuidores, porém com uma efetividade inferior a 90%, para isso, utilizar a combinação das heurísticas SHA e palavra-chaves, (ii) encontrar novas contribuições, porém menores mas com efetividade de 100% combinando as três heurísticas.

Foram analisadas as características dos contribuidores casuais, focando nos diferentes meios de interação com o projeto. As características analisadas foram em relação a participação do contribuidor casual em redes sociais do GitHub, como por exemplo, *followers* e características envolvendo outras atividades como realizar comentários em PRs do projeto. Com os resultados encontrados é possível afirmar que o contribuidor casual não se relaciona com o projeto por muito tempo. Em uma das descobertas nota-se que o contribuidor casual permanece no projeto no período de 1 ano em média. Além disso, o contribuidor casual não utiliza as redes sociais da plataforma para interagir com os mantenedores do projeto, e quando utiliza, tendem a interagir antes de contribuir com o projeto. As contribuições em sua grande maioria (68,7%) modificam código-fonte do projeto. O segundo maior tipo de contribuição é de documentação (7,48%).

Foram realizadas três tentativas de agrupar os contribuidores casuais através de suas características, mas nenhum dos métodos propostos obtiveram êxito. Os algoritmos utilizados (*K-Means* e *Model-based clustering*) apresentaram resultados similares e pouco efetivo quanto a separação dos grupos, impossibilitando a caracterização de cada grupo obtido. O método manual utilizado—agrupando os contribuidores de acordo com as tentativas de contribuições (antes e depois do PR aceito— também se demonstrou inadequado para dividir os contribuidores casuais de acordo com a interação com os projetos.

6.1 Trabalhos futuros

Um potencial próximo passo para esse trabalho seria desenvolver um sistema de sugestão para novas contribuições. Para isso, seria necessário analisar outros projetos no qual o contribuidor participou e analisar o envolvimento do mesmo. Além disso é necessário criar uma rede entre os contribuidores para descobrir os projetos que fazem parte do ecossistema do projeto contribuído, por exemplo, os contribuidores casuais que participaram do projeto `node` de forma casual, participam de quais outros projetos? O sistema pode se basear na contribuição de outros contribuidores do projeto, e assim, sugerir contribuições em projetos que façam parte do ecossistema, ou da mesma linguagem de programação.

Alguns contribuidores casuais obtidos neste trabalho podem ao decorrer do tempo realizar novas contribuições, e assim deixar de ser considerado um contribuidor casual. Para isso seria necessário realizar uma nova coleta nos projetos utilizados, além de verificar

se o projeto ainda está ativo. Uma alternativa, seria utilizar um banco de dados orientado a documento como o MongoDB[1].

[1] http://ghtorrent.org/mongo.html

REFERÊNCIAS

Alexander Hars, S. O. Working for free? motivations for participating in open-source projects. *International Journal of Electronic Commerce*, v. 6, n. 3, p. 25–39, 2002.

Balali, S.; Steinmacher, I.; Annamalai, U.; Sarma, A.; Gerosa, M. A. Newcomers' barriers... is that all? an analysis of mentors' and newcomers' barriers in oss projects. *Computer Supported Cooperative Work (CSCW)*, p. 1–36, 2018.

Balestra, M.; Zalmanson, L.; Cheshire, C.; Arazy, O.; Nov, O. It was fun, but did it last?: The dynamic interplay between fun motives and contributors' activity in peer production. *Proceedings of the ACM: Human-Computer Interaction*, v. 1, 2017.

Barcomb, A. Episodic volunteering in open source communities. *EASE 2016*, p. 1–3, 2016.

Bezerra, R. M. M.; da Silva, F. Q. B.; Santana, A. M.; Magalhaes, C. V. C.; Santos, R. E. S. Replication of empirical studies in software engineering: An update of a systematic mapping study. In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2015a, p. 1–4.

Bezerra, R. M. M.; da Silva, F. Q. B.; Santana, A. M.; Magalhaes, C. V. C.; Santos, R. E. S. Replication of empirical studies in software engineering: An update of a systematic mapping study. In: *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2015b, p. 1–4.

Boehmke, B. C.; Hazen, B. T. The Future of Supply Chain Information Systems: The Open Source Ecosystem. *Global Journal of Flexible Systems Management*, v. 18, n. 2, p. 163–168, 2017.

Borges, H.; Hora, A.; Valente, M. T. Understanding the factors that impact the popularity of github repositories. In: *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*, IEEE, 2016, p. 334–344.

Bryen, L. M.; Madden, K. M. *Bounce-back of episodic volunteers: What makes episodic volunteers return? working paper no. cpns 32.* Working paper, QUT, 2006.

Campbell, D. T. Experimental and quasi-experimental designs for research on teaching. *Handbook of research on teaching*, v. 5, p. 171–246, 1963.

Carver, J. C. Towards reporting guidelines for experimental replications: A proposal. In: *1st international workshop on replication in empirical software engineering*, Citeseer, 2010, p. 2–5.

Chen, D.; Stolee, K.; Menzies, T. Replication can improve prior results: A github study of pull request acceptance. *arXiv preprint arXiv:1902.04060*, 2019.

Coelho, J.; Valente, M. T. Why modern open source projects fail. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2017, New York, NY, USA: ACM, 2017, p. 186–196 (*ESEC/FSE 2017*,).

Cosentino, V.; Luis, J.; Cabot, J. Findings from github: methods, datasets and limitations. In: *Proceedings of the 13th International Conference on Mining Software Repositories*, ACM, 2016, p. 137–141.

Crowston, K.; Wei, K.; Howison, J.; Wiggins, A. Free/libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, v. 44, n. 2, p. 7, 2012.

Câmara, G.; Fonseca, F. Information policies and open source software in developing countries. *Journal of the American Society for Information Science and Technology*, v. 58, n. 1, p. 121–132, 2007.

Dabbish, L.; Stuart, C.; Tsay, J.; Herbsleb, J. Social coding in github: transparency and collaboration in an open software repository. In: *Proceedings of the ACM 2012 conference on computer supported cooperative work*, ACM, 2012, p. 1277–1286.

Dinh-Trong, T. T.; Bieman, J. M. The freebsd project: A replication case study of open source development. *IEEE Transactions on Software Engineering*, v. 31, n. 6, p. 481–494, 2005.

Fogel, K. *Producing open source software: How to run a successful free software project.* "O'Reilly Media, Inc.", 2005.

Fraley, C.; Raftery, A. E. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*, v. 97, n. 458, p. 611–631, 2002.

GHTorrent Downloads. http://ghtorrent.org/downloads.html, acessado em: 20/01/2019, 2019a.

GHTorrent The ghtorrent project. http://ghtorrent.org/, acessado em: 20/01/2019, 2019b.

Git Git Essencial - Gravando Alterações no Repositório. https://git-scm.com/book/pt-br/v1/Git-Essencial-Gravando-Alteraμes-no-Repositrio, acessado em: 30/04/2019, 2019a.

Git, D. Distributed Git - Maintaining a Project . https://git-scm.com/book/en/v2/Distributed-Git-Maintaining-a-Project, acessado em: 12/03/2019, 2019b.

GitHub Fork A Repo. https://help.github.com/articles/fork-a-repo/, acessado em: 02/03/2018, 2018a.

GitHub GitHubs is how people build software. https://github.com/about, acessado em: 01/06/2018, 2018b.

GitHub Notifications & Stars — The GitHub Blog. https://blog.github.com/2012-08-06-notifications-stars/, acessado em: 04/06/2018, 2018c.

GitHub About issues. https://help.github.com/en/articles/about-issues, acessado em: 23/06/2019, 2019a.

GitHub About organizations. https://help.github.com/en/articles/about-organizations, acessado em: 10/07/2019, 2019b.

GitHub Fork a repo. https://help.github.com/en/articles/fork-a-repo, acessado em: 06/12/2019, 2019c.

GitHub GAutolinked references and URLs. https://help.github.com/articles/autolinked-references-and-urls/, acessado em: 11/12/2018, 2019d.

GitHub Github: Forks, Collaborators, Watchers. https://www.metrics-toolkit.org/github-forks-collaborators-watchers/, acessado em: 06/12/2019, 2019e.

GitHub GitHub Glossary. https://help.github.com/articles/github-glossary/, acessado em: 11/12/2018, 2019f.

GitHub O que é Software Livre. https://help.github.com/articles/about-stars/, acessado em: 20/01/2019, 2019g.

GitHub Understanding the GitHub flow. https://guides.github.com/introduction/flow/, acessado em: 03/12/2019, 2019h.

Gnu O que é Software Livre. https://www.gnu.org/philosophy/free-sw.html, acessado em: 29/01/2018, 2018.

Gousios, G. The GHTorrent dataset and tool suite. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR, 2013, p. 233–236 (*MSR*,).

Gousios, G.; Pinzger, M.; Deursen, A. v. An exploratory study of the pull-based software development model. In: *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, 2014, p. 345–355 (*ICSE 2014*,).

Gousios, G.; Storey, M.-A.; Bacchelli, A. Work practices and challenges in pull-based development: the contributor's perspective. In: *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*, IEEE, 2016, p. 285–296.

Grissom, R.; Kim, J. *Effect sizes for research: Univariate and multivariate applications*. Taylor & Francis, 2005.

Haff, G. Node.js: Community for casual contributors — Opensource.com. https://opensource.com/article/17/3/nodejs-community-casual-contributors, acessado em: 02/03/2018, 2018.

Hannebauer, C.; ..., V. G. o. t. t. I. S. o. O.; 2017, U. On the Relationship between Newcomer Motivations and Contribution Barriers in Open Source Projects. *dl.acm.org*, 2017.

Hannebauer, C.; Wolff-Marting, V.; Gruhn, V. Contributor-interaction patterns in floss development. *EuroPLoP 2011*, 2011.

Herzig, K.; Just, S.; Zeller, A. The impact of tangled code changes on defect prediction models. *Empirical Software Engineering*, v. 21, n. 2, p. 303–336, 2016.

Kalliamvakou, E.; Gousios, G.; Blincoe, K.; Singer, L.; German, D. M.; Damian, D. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, v. 21, n. 5, p. 2035–2071, 2016.

Ketchen, D. J.; Shook, C. L. The application of cluster analysis in strategic management research: an analysis and critique. *Strategic management journal*, v. 17, n. 6, p. 441–458, 1996.

Lee, A.; Carver, J. C.; Bosu, A. Understanding the Impressions, Motivations, and Barriers of One Time Code Contributors to FLOSS Projects: A Survey. *ICSE'17*, 2017.

Lu, Y.; Mao, X.; Li, Z.; Zhang, Y.; Wang, T.; Yin, G. Does the role matter? An investigation of the code quality of casual contributors in GitHub. *APSEC 2017*, 2017.

McFadden, D. Analysis of qualitative choice behavior. zarembka, p.(ed.): Frontiers in econometrics. 1974.

McFadden, D. Quantitative methods for analyzing travel behaviour of individuals: Some recent developments (cowles foundation discussion papers no. 474). *Cowles Foundation for Research in Economics, Yale University*, 1977.

Metz, C. How GitHub conquered Google, Microsoft, and everyone else. `https://www.wired.com/2015/03/github-conquered-google-microsoft-everyone-else/`, acessado em: 29/01/2018, 2015.

Mockus, A.; Fielding, R. T.; Herbsleb, J. A case study of open source software development: The apache server. In: *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, New York, NY, USA: ACM, 2000, p. 263–272 (*ICSE '00*,).

Pham, R.; Singer, L.; Schneider, K. Building test suites in social coding sites by leveraging drive-by commits. In: *ICSE 2013*, ICSE '13, 2013, p. 1209–1212 (*ICSE '13*,).

Pinto, G.; Steinmacher, I.; Dias, L. F.; Gerosa, M. On the challenges of open-sourcing proprietary software projects. *Empirical Software Engineering*, p. 1–27, 2018.

Pinto, G.; Steinmacher, I.; Gerosa, M. A. More common than you think: An in-depth study of casual contributors. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016, p. 112–123.

Ramos, F. V.; Gerosa, M. A.; Chaves, A. P.; Steinmacher, I. Um estudo exploratório sobre contribuições casuais em projetos de software livre: caso do projeto libreoffice. *WDES 2015*, p. 9, 2015.

Ray, B.; Posnett, D.; Devanbu, P.; Filkov, V. A large-scale study of programming languages and code quality in GitHub. *Commun of the ACM*, v. 60, n. 10, p. 91–100, 2017.

Rebouças, M.; Santos, R. O.; Pinto, G.; Castor, F. How does contributors' involvement influence the build status of an open-source software project? In: *Proceedings of the 14th International Conference on Mining Software Repositories*, IEEE Press, 2017, p. 475–478.

Romano, J.; Kromrey, J.; Coraggio, J.; Skowronek, J. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys? In: *annual meeting of the Florida Association of Institutional Research*, 2006, p. 1–3.

Shah, S. K. Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science*, v. 52, n. 7, p. 1000–1014, 2006.

Sheoran, J.; Blincoe, K.; Kalliamvakou, E.; Damian, D.; Ell, J. Understanding "watchers"on GitHub. *MSR 2014*, p. 336–339, 2014.

Shull, F. J.; Carver, J. C.; Vegas, S.; Juristo, N. The role of replications in empirical software engineering. *Empirical Softw. Engg.*, v. 13, n. 2, p. 211–218, 2008.

Starkweather, J.; Moske, A. K. Multinomial logistic regression. *Consulted page at September 10th: http://www. unt. edu/rss/class/Jon/Benchmarks/MLR_JDS_Aug2011. pdf*, v. 29, p. 2825–2830, 2011.

Statcounter Desktop vs Mobile vs Tablet Market Share Worldwide — StatCounter Global Stats. `http://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/{\#}monthly-200901-201712`, acessado em: 20/03/2018, 2018.

Steinmacher, I.; Chaves, A. P.; Conte, T. U.; Gerosa, M. A. Preliminary empirical identification of barriers faced by newcomers to open source software projects. In: *2014 Brazilian Symposium on Software Engineering*, IEEE, 2014, p. 51–60.

Steinmacher, I.; Pinto, G.; Wiese, I.; Gerosa, M. A. Almost there: A study on quasi-contributors in open-source software projects. In: *ICSE'18*, 2018, p. 1–12.

Steinmacher, I. F. *Supporting newcomers to overcome the barriers to contribute to open source software projects.* Tese de Doutoramento, Universidade de São Paulo, 2015.

Swinscow, T. D. V.; Campbell, M. J.; et al. *Statistics at square one.* Bmj London, 2002.

Torvalds, L. G1 - Brasil ganha em independência ao adotar software livre, diz pai do Linux - notícias em Tecnologia e Games. `http://g1.globo.com/tecnologia/noticia/2010/08/brasil-ganha-em-independencia-ao-adotar-software-livre-diz-pai-do-linux.html`, acessado em: 29/01/2018, 2018.

Vasilescu, B. Human aspects, gamification, and social media in collaborative software engineering. In: *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, 2014, p. 646–649.

Vasilescu, B.; Serebrenik, A.; Goeminne, M.; Mens, T. On the variation and specialisation of workload-a case study of the gnome ecosystem community. *Empirical Software Engineering*, v. 19, 2014.

Wang, J.; Sarma, A. Which bug should i fix: helping new developers onboard a new project. In: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, ACM, 2011, p. 76–79.

Wilcoxon, F. Individual comparisons by ranking methods. *Biometrics bulletin*, v. 1, n. 6, p. 80–83, 1945.

Yu, Y.; Yin, G.; Wang, H.; Wang, T. Exploring the patterns of social behavior in github. In: *Proceedings of the 1st international workshop on crowd-based software development methods and technologies*, ACM, 2014, p. 31–36.

Zhou, M.; Mockus, A. Who will stay in the floss community? modeling participant's initial behavior. *IEEE Transactions on Software Engineering*, v. 41, n. 1, p. 82–99, 2015.

A

Apêndice A

Tabela 1.1: Projetos utilizados neste trabalho

Projetos	Linguagem	Idade	# Estrelas	# PRs
ccv	C	9	5859	41
cphalcon	C	8	8024	2454
emscripten	C	9	12702	2449
godot	C	7	8900	4045
gumbo-parser	C	7	4040	115
jq	C	8	9475	301
masscan	C	7	6664	98
memcached	C	10	6550	192
mjolnir	C	6	4913	80
openframeworks	C	10	5375	2549
php-src	C	9	12493	2704
redis	C	10	24610	1612
scikit-learn	C	10	20492	5341
the_silver_searcher	C	9	12936	563
twemproxy	C	9	7059	186
appjs	C++	9	3696	52
bitcoin	C++	10	15989	7805
cocos2d-x	C++	10	10484	14131
fish-shell	C++	8	7939	1112

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
folly	C++	8	8969	336
hhvm	C++	10	14439	2656
mongo	C++	10	11886	1152
mosh	C++	10	6532	288
nw.js	C++	8	31694	656
openage	C++	6	5870	490
osquery	C++	6	9644	2228
pdf2htmlEX	C++	8	6235	88
phantomjs	C++	10	22767	982
rethinkdb	C++	8	19395	498
rocksdb	C++	7	8375	1758
textmate	C++	8	11221	382
trinitycore	C++	10	3854	3881
xbmc	C++	10	6115	12378
aleph	Clojure	10	1844	113
cascalog	Clojure	9	1297	151
compojure	Clojure	10	3190	42
enlive	Clojure	10	1371	59
incanter	Clojure	10	1885	181
instaparse	Clojure	7	1804	51
Korma	Clojure	9	1275	161
leiningen	Clojure	10	5466	724
LightTable	Clojure	7	9533	322
Midje	Clojure	10	1292	78
modern-cljs	Clojure	8	2383	321
mori	Clojure	8	2696	82
om	Clojure	7	6153	223
overtone	Clojure	10	3749	152
pedestal	Clojure	8	1778	250
plumbing	Clojure	8	1261	74
quil	Clojure	9	1909	75
riemann	Clojure	9	3315	512
at.js	Coffeescript	9	4537	104

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
bacon.js	Coffeescript	9	5664	285
bootbox	Coffeescript	9	4106	176
bootstrap-tour	Coffeescript	8	3850	160
brunch	Coffeescript	10	6062	497
chaplin	Coffeescript	9	2927	417
chosen	Coffeescript	9	20753	705
codecombat	Coffeescript	7	5588	2569
coffeescript	Coffeescript	10	13886	902
docpad	Coffeescript	9	2893	146
Framer	Coffeescript	8	4790	218
hubot-scripts	Coffeescript	9	3377	1496
jquery.payment	Coffeescript	8	3648	126
karma	Coffeescript	9	8808	1269
morris.js	Coffeescript	9	6541	179
pow	Coffeescript	10	3350	77
quill	Coffeescript	7	14482	290
turbolinks	Coffeescript	8	4023	348
ChicagoBoss	Erlang	10	1645	367
cowboy	Erlang	10	4161	386
disco	Erlang	10	1453	238
ejabberd	Erlang	10	3082	454
lfe	Erlang	10	1488	155
mochiweb	Erlang	10	1511	97
n2o	Erlang	7	1038	195
nitrogen	Erlang	10	820	45
rabbitmq-server	Erlang	10	3060	584
rebar	Erlang	8	894	401
riak_core	Erlang	10	821	716
tsung	Erlang	10	1304	151
yaws	Erlang	10	988	124
beego	Go	8	11892	952
cayley	Go	6	10260	320
drone	Go	7	10835	966

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
etcd	Go	7	14530	5179
flynn	Go	7	6192	2154
gogs	Go	7	20260	1166
groupcache	Go	7	5627	43
hub	Go	10	11071	497
influxdb	Go	7	11236	3583
lime	Go	8	13232	186
ngrok	Go	7	11026	58
nsq	Go	8	10057	585
packer	Go	7	6747	2144
revel	Go	8	8566	478
syncthing	Go	7	17016	1438
websocketd	Go	8	8375	83
cgrep	Haskell	6	636	16
elm-compiler	Haskell	8	3941	537
fay	Haskell	8	1109	125
ghcjs	Haskell	10	1711	100
gitit	Haskell	10	1402	123
hakyll	Haskell	10	1566	252
haste-compiler	Haskell	8	1210	66
Haxl	Haskell	6	2788	37
Idris-dev	Haskell	9	1968	2132
IHaskell	Haskell	7	1506	242
lens	Haskell	8	1100	318
pandoc	Haskell	10	9224	741
purescript	Haskell	7	3850	1393
scotty	Haskell	9	1073	105
shellcheck	Haskell	7	8583	80
yesod	Haskell	10	1749	550
yi	Haskell	9	1087	300
ActionBarSherlock	Java	9	7180	253
android-async-http	Java	9	9761	262
Android-Bootstrap	Java	8	4215	99

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
Android-PullToRefresh	Java	9	8182	79
androidannotations	Java	9	9476	569
elasticsearch	Java	10	24517	12262
iosched	Java	7	14454	101
jenkins	Java	10	8796	2962
libgdx	Java	8	12033	2355
netty	Java	10	10937	3425
phonegap-plugins	Java	10	3340	694
picasso	Java	7	13849	452
retrofit	Java	9	23266	777
RxJava	Java	8	26605	2750
SlidingMenu	Java	8	10433	129
spring-framework	Java	10	15617	1493
zxing	Java	8	14967	184
angular.js	Javascript	10	40000	7541
backbone	Javascript	10	26077	1810
brackets	Javascript	9	27324	5136
Chart.js	Javascript	8	31576	1243
d3	Javascript	10	40000	1052
html5-boilerplate	Javascript	10	37495	835
impress.js	Javascript	9	31734	254
jquery	Javascript	10	40000	2273
jQuery-File-Upload	Javascript	10	26961	305
meteor	Javascript	9	37354	2053
Modernizr	Javascript	10	20632	983
moment	Javascript	9	32601	1426
node	Javascript	6	38082	9077
react	Javascript	7	40000	5684
select2	Javascript	9	20728	1170
Semantic-UI	Javascript	7	36209	683
socket.io	Javascript	10	34946	673
three.js	Javascript	10	34509	4857
underscore	Javascript	10	20968	1400

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
AFNetworking	Objective-C	9	29424	1338
Alcatraz	Objective-C	7	9859	200
asi-http-request	Objective-C	10	5705	128
AsyncDisplayKit	Objective-C	6	11693	1920
CocoaLumberjack	Objective-C	10	9289	415
FlatUIKit	Objective-C	7	7521	97
GPUImage	Objective-C	9	16099	414
MagicalRecord	Objective-C	10	10307	390
Mantle	Objective-C	8	10542	306
MBProgressHUD	Objective-C	10	13481	191
nimbus	Objective-C	9	6244	310
OpenEmu	Objective-C	10	7810	234
ReactiveCocoa	Objective-C	8	17512	1597
RestKit	Objective-C	10	9982	667
Shimmer	Objective-C	7	7983	33
SVProgressHUD	Objective-C	10	9933	284
three20	Objective-C	10	7449	242
ViewDeck	Objective-C	9	5111	132
xctool	Objective-C	7	6368	300
ack2	Perl	9	1227	125
contrib	Perl	9	939	766
Dancer	Perl	10	698	604
duckduckgo	Perl	9	1237	121
FlameGraph	Perl	9	4506	90
git-cal	Perl	7	966	39
lua-nginx-module	Perl	10	4867	394
mojo	Perl	10	1828	499
MySQLTuner-perl	Perl	10	3676	134
prey-bash-client	Perl	10	884	25
rainbarf	Perl	8	916	11
rcm	Perl	7	1592	101
sshuttle	Perl	6	8284	77
_s	PHP	9	7599	733

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
cakephp	PHP	10	7065	6634
CodeIgniter	PHP	9	14619	2542
composer	PHP	9	9687	2206
DesignPatternsPHP	PHP	7	12935	226
facebook-php-sdk	PHP	9	3378	155
Faker	PHP	9	11394	918
guzzle	PHP	9	9674	706
laravel	PHP	9	33840	2719
linux-dash	PHP	7	7702	198
Mobile-Detect	PHP	8	7084	177
openbay	PHP	6	3574	59
piwik	PHP	9	7770	2420
slim	PHP	10	8141	1053
webmachine	PHP	10	1325	208
yii2	PHP	7	10412	5620
zendframework	PHP	10	5425	5544
zephir	PHP	7	1953	715
awesome-python	Python	6	37420	823
django	Python	8	27288	8977
flask	Python	10	28290	1157
httpie	Python	9	30898	173
ipython	Python	10	11616	5247
Mailpile	Python	9	6935	476
reddit	Python	10	13554	867
requests	Python	10	26850	1667
scrapy	Python	10	22060	1544
sentry	Python	10	13562	3254
tornado	Python	10	13908	974
YouCompleteMe	Python	8	14187	554
youtube-dl	Python	10	28406	2335
activeadmin	Ruby	10	7427	1912
cancan	Ruby	10	6129	200
capistrano	Ruby	10	9646	791

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
capybara	Ruby	10	7883	735
carrierwave	Ruby	10	7553	683
devise	Ruby	10	17313	1047
diaspora	Ruby	10	11203	3206
discourse	Ruby	8	22225	4682
homebrew-cask	Ruby	9	11339	33851
huginn	Ruby	8	16706	931
jekyll	Ruby	10	30323	2837
octopress	Ruby	10	9381	793
paperclip	Ruby	10	8551	751
rails	Ruby	10	36073	19837
resque	Ruby	10	7849	821
sinatra	Ruby	10	9366	702
spree	Ruby	10	8541	4871
akka	Scala	10	7438	6243
ArnoldC	Scala	7	4964	31
finagle	Scala	10	5835	379
flockdb	Scala	10	3067	50
gatling	Scala	9	3135	584
gitbucket	Scala	7	6437	531
gizzard	Scala	10	2119	72
incubator-predictionio	Scala	7	10225	324
playframework	Scala	9	9479	4996
sbt	Scala	10	3160	1374
scala-js	Scala	7	3039	1519
scalatra	Scala	10	2163	336
scalaz	Scala	10	3162	1107
scalding	Scala	9	2828	1169
snowplow	Scala	8	3512	283
spray	Scala	10	2482	452
summingbird	Scala	7	1859	468
swagger-core	Scala	9	4330	765
textteaser	Scala	7	1865	12

continuação da Tabela - 1.1

Projetos	Linguagem	Idade	# Estrelas	# PRs
doppio	Typescript	8	1658	105
egret-core	Typescript	7	1778	87
litecoin	Typescript	8	1699	175
primecoin	Typescript	7	243	22
reddcoin	Typescript	6	159	25
shellshape	Typescript	9	362	23
shumway	Typescript	9	3368	1602
trNgGrid	Typescript	7	258	13
tsd	Typescript	8	1096	72
turbulenz_engine	Typescript	7	2692	13
TypeScript	Typescript	6	25060	4931
typescript-node-definitions	Typescript	8	196	48
winjs	Typescript	7	3895	117