UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

UM LEVANTAMENTO ESTRUTURADO DO USO DA FERRAMENTA DE REGISTRO DE PROBLEMAS NA PLATAFORMA GITHUB

Casimiro Conde Marco Neto

Orientador
Márcio de Oliveira Barros

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2016

UM LEVANTAMENTO ESTRUTURADO DO USO DA FERRAMENTA DE REGISTRO DE PROBLEMAS NA PLATAFORMA GITHUB

Casimiro Conde Marco Neto

DISSERTAÇÃO APRESENTADA COMO REQUISITO PARCIAL PARA OBTENÇÃO DO TÍTULO DE MESTRE PELO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA DA UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO (UNIRIO). APROVADA PELA COMISSÃO EXAMINADORA ABAIXO ASSINADA

Aprovada por:

_____

Márcio de Oliveira Barros, D.Sc. – UNIRIO

_____

Gleison dos Santos Souza, D.Sc. – UNIRIO

_____

Leonardo Gresta Paulino Murta, D.Sc. - UFF

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2016

Marco Neto, Casimiro Conde.
M321 Um levantamento estruturado do uso da ferramenta de registro de problemas na plataforma GitHub / Casimiro Conde Marco Neto, 2016.
96 f. ; 30 cm

Orientador: Márcio de Oliveira Barros.
Dissertação (Mestrado em Informática) - Universidade Federal do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

1. Software - Desenvolvimento. 2. GitHub. 3. Gestão de problemas. 4. Issue Tracking System. I. Barros, Márcio de Oliveira. II. Universidade Federal do Estado do Rio de Janeiro. Centro de Ciências Exatas e Tecnológicas. Curso de Mestrado em Informática. III. Título.

CDD - 005.1

Para Marco e Soraya, cuja força de vontade e dedicação me inspiram a acreditar sempre no melhor de mim e do próximo.

Agradecimentos

Agradeço aos meus pais, Marco Aurélio e Soraya Pacca, por estarem ao meu lado a cada desafio enfrentado, a cada conquista realizada e a cada perda sentida. Sem sua base, nada seria possível.

Agradeço aos meus familiares e amigos por todo o apoio, carinho, atenção e conselhos dados ao longo dessa jornada.

Agradeço ao meu orientador Márcio Barros, que ao longo desses anos esteve sempre presente, disponível e aberto para compartilhar sua experiência e conhecimento, tornando esse trabalho possível e acima de tudo, melhor. Sua dedicação e profissionalismo me inspiram.

Deixo também meu agradecimento a todos os professores do PPGI e do BSI, que me acompanham desde 2009 nessa jornada, primeiro me ensinando os caminhos do Bacharelado e agora do Mestrado.

Por fim, agradeço a Deus por ter iluminado meu caminho, ter me dado forças e me guiado, nos momentos de dificuldades.

Obrigado a todos.

NETO, Casimiro Conde Marco. **Um Levantamento Estruturado do Uso da Ferramenta de Registro de Problemas na Plataforma GitHub.** UNIRIO, 2016. 83 páginas. Dissertação de Mestrado. Departamento de Informática Aplicada, UNIRIO

RESUMO

A utilização de ferramentas de *issue tracking* oferece suporte aos colaboradores de um projeto de desenvolvimento de software na identificação e resolução de problemas e questões a serem tratadas durante o ciclo de vida do projeto. Visando compreender as características dos projetos que utilizam essa ferramenta, essa Dissertação replicou o estudo realizado por Bissyande *et al.* em 2013 e divulgado no artigo "*Got Issues? Who Cares About It? A Large Scale Investigation of Issue Trackers from GitHub*". Visando estudar novos pontos de vista sobre a utilização deste tipo de ferramenta, esse estudo acrescenta novas questões de pesquisa ao estudo anterior, buscando compreender quais as características dos colaboradores e *issues* envolvidos com a funcionalidade do GitHub que permite o encerramento de *isssues* através de *commits.* Essa análise tem como objetivo verificar se essa funcionalidade pode ser utilizada como uma evidência do relacionamento entre um *issue* e a parte do código-fonte do projeto a qual este *issue* se refere, caso ele esteja relacionado ao código.

Palavra-chave: GitHub, *Issues*, Gestão de Problemas, *Issue Tracking System*, *Commit*

ABSTRACT

Issue tracking systems support project contributors to identify and resolve problems that must be addressed during the life-cycle of a software product. To better understand the characteristics of projects that uses this kind of tool, this work replicates the study presented by BISSYANDE *et al*. in the paper entitled "*Got Issues? Who Cares About It? A Large Scale Investigation of Issue Trackers from GitHub*" and published in 2013. To study new perspectives regarding issue tracking systems, the present study adds new research questions to the former one. These questions aim to identify the characteristics of collaborators or issues involved with the functionality provided by GitHub that allows closing issues through commits to the version control system. This analysis aims to determine if this functionality can be used as evidence of a relationship between the issue and parts of the source code to which it is related, if this issue is related to the code.

Keywords: GitHub, Issues, Problem, Issue Tracking System, Commit

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

O desenvolvimento de software é uma atividade complexa e que envolve muito esforço intelectual para construir um produto que atenda às necessidades e expectativas do cliente. É comum, durante o desenvolvimento, que sejam identificadas questões a serem analisadas e respondidas sobre o software em construção. Essas questões podem ser chamadas de *issues*.

Issues indicam a existência de defeitos que devem ser corrigidos, problemas ou riscos que podem atrapalhar o andamento do projeto, oportunidades de melhoria identificadas pelos usuários e desenvolvedores, pendências e dúvidas a serem respondidas ou atividades a serem executadas pela equipe envolvida no projeto. De acordo com BISSYANDE *et al.* (2013), desenvolvedores e usuários identificam *issues* com frequência em sistemas de software e são encorajados pela equipe de desenvolvimento a registrar esses *issues* em um *issue tracking system* para facilitar a comunicação entre os grupos envolvidos no desenvolvimento e uso do sistema em questão.

A utilização de *issue tracking systems* é um assunto que vem sendo estudado por diversos pesquisadores, conforme será apresentado no próximo capítulo, incluindo BISSYANDE *et al.* (2013). Essa área de estudo é importante porque analisa como *issue tracking systems* podem apoiar a equipe de um projeto durante o seu desenvolvimento, quais as características dos usuários que registram *issues* e qual é o ciclo de vida de um *issue* em um projeto.

Um tópico pouco abordado nos artigos analisados é a relação entre os *issues* registrados em um projeto e o código-fonte envolvido em sua resolução, se essa relação existe e como ela pode ser identificada. O estudo apresentado nesta Dissertação tem como objetivo identificar se essa relação existe, quais as características dos *issues* que possuem essa relação e dos desenvolvedores que a constroem.

Para entender a relação entre um issue e o código-fonte do projeto, utilizaremos dados de softwares *open source,* onde as informações sobre o desenvolvimento, os desenvolvedores e os *issues* estão disponíveis para consulta pública. Serão coletados os dados de aproximadamente 220.000 projetos registrados na plataforma de suporte ao desenvolvimento de software GitHub, que atualmente é a maior plataforma de hospedagem de projetos *open source* na web[1].

1.1 Objetivos

O objetivo principal desse estudo é compreender as características da relação entre *issues* e o código-fonte de um projeto de software. Essa análise será feita sobre uma amostra de aproximadamente 77.000 projetos, resultantes da filtragem feita nos 220.000 projetos coletados da plataforma GitHub. Além deste objetivo principal, o estudo reportado nesta Dissertação se propõe a analisar como é a utilização de *issue tracking systems* em projetos *open source*.

Para isso, esse estudo será dividido em duas partes. Na primeira parte será realizada a replicação do maior estudo feito sobre *issue tracking systems* até o momento, publicado em 2013, por BISSYANDE *et al.* (2013). Nesse estudo, foram elaboradas diversas hipóteses e questões de pesquisa que buscaram entender as características de projetos e colaboradores que participam destes projetos através da plataforma GitHub e que utilizam a ferramenta de *issue tracking system*. A amostra utilizada neste estudo foi de 100.000 projetos coletados do GitHub.

Os autores executaram uma análise empírica dos dados coletados, apresentando as suas conclusões para cada questão de pesquisa através de testes estatísticos e gráficos, assim como a relação das conclusões com as hipóteses inicialmente elaboradas. Este trabalho propõe a replicação das análises feitas pelo estudo de 2013, utilizando uma amostra mais recente de projetos da plataforma GitHub e um projeto experimental tão próximo quanto possível do utilizado no estudo anterior. Com isso, busca-se avaliar se as conclusões do artigo de 2013 se mantêm após três anos desde a sua divulgação.

A replicação de estudos experimentais é uma tarefa bastante importante para garantir que os resultados encontrados anteriormente são observáveis e verificar se eles se

[1] https://github.com/open-source

mantém idênticos ou se o seu comportamento apresentou alguma mudança. Porém, de acordo com YONG (2012), a replicação de estudos é pouco disseminada devido à dificuldade de realizar essas replicações. Os pesquisadores que as realizam muitas vezes acabam não conseguindo chegar a resultados semelhantes aos encontrados nos estudos originais e têm dificuldade em publicar estudos apresentando resultados negativos. Também de acordo com YONG (2012), as replicações exatas de estudos são muito difíceis de serem realizadas, pois em grande parte das vezes não é possível replicar o método utilizado pelo estudo original. Sendo assim, grande parte das replicações realizadas são "conceituais", onde o conceito e os objetivos da replicação são idênticos aos do estudo original, porém o método utilizado para sua realização é diferente. Esse estudo representa uma replicação conceitual do estudo realizado por BISSYANDE *et al.* (2013), visto que os métodos utilizados originalmente não foram disponibilizados. De acordo com BIOLCHINI *et al.* (2005), sem uma cultura de pesquisa que pregue a forte realização de replicações e revisões sistemáticas, é mais simples para pesquisadores empreenderem em pesquisas que mais se adéquam às suas próprias áreas de interesse do que contribuírem para o interesse geral da comunidade científica.

Na segunda parte desse estudo serão discutidas novas questões de pesquisa que buscam analisar a relação entre *issues* e o código-fonte de um projeto. Essa relação não pode ser identificada de forma explicita no GitHub, visto que não existe uma funcionalidade nativa que permita o relacionamento de um *issue* com os trechos do código-fonte que ele afeta, quando há esta relação. Para analisar essa relação, assumiremos que o relacionamento entre código e *issue* pode ser construído através da funcionalidade de fechamento de *issues* por *commit*, já existente na plataforma GitHub. Ou seja, consideraremos que um *issue* está relacionado ao código-fonte quando esse for fechado através de um *commit*, sendo então relacionados aos módulos de código-fonte afetados por este *commit*. Esse fechamento de *issues* por *commits* é realizado através de *tags* utilizadas no texto do *commit* que identificam o *issue* que deve ser encerrado. Essas análises serão realizadas utilizando a mesma estrutura do artigo de BISSYANDE *et al.* (2013) e a mesma amostra de projetos utilizada para a sua replicação.

Em uma análise preliminar, foi identificado que a utilização da funcionalidade de fechamento de *issues* através de *commits* não é difundida em larga escala. Sendo assim, como um dos objetivos da segunda parte desse estudo, buscaremos identificar se esse

mecanismo pode ser utilizado como uma evidência do relacionamento entre *issues* e o código do projeto. Também tentaremos caracterizar os *issues* fechados por *commit* com outras características de *issues*, como os desenvolvedores envolvidos, o uso de rótulos e seu tempo de correção. Enquanto a primeira parte do estudo pretende adicionar evidências atualizadas aos resultados encontrados por BISSYANDE *et al.* (2013), a segunda parte pretende agregar conhecimento e novas informações aos resultados identificados pelos autores.

1.2 Organização da Dissertação

Esta dissertação está organizada em cinco capítulos. O Capítulo 1 apresenta a introdução do trabalho e os objetivos aos quais ele pretende atender. O Capítulo 2 apresenta uma revisão bibliográfica dos trabalhos que analisaram projetos da plataforma GitHub. O Capítulo 3 apresenta a metodologia utilizada nesse estudo, as hipóteses e questões de pesquisa estudadas, a curadoria realizada nos dados coletados e as dificuldades encontradas durante a realização do estudo. O Capítulo 4 apresenta a análise de dados e as conclusões alcançadas após a análise das questões de pesquisa propostas nesse estudo. Finalmente, o Capítulo 5 apresenta as conclusões extraídas após a análise das questões de pesquisa, as limitações e ameaças à validade desse estudo e propostas para trabalhos futuros.

2. Revisão Bibliográfica

Este capítulo apresenta estudos sobre o GitHub, suas funcionalidades e características que propiciam a interação e a colaboração entre desenvolvedores, além de trabalhos que analisam a utilização, a aplicabilidade e as características de *issue tracking systems* em projetos de software. A primeira seção do capítulo apresenta a plataforma GitHub e os motivos que embasaram a sua escolha como base para esse estudo. A segunda seção apresenta uma revisão literária de estudos que analisaram os padrões de utilização do GitHub e o impacto do ambiente colaborativo no desenvolvimento de software. A terceira seção apresenta análises já executadas sobre *issue tracking systems* em projetos armazenados principalmente no GitHub, observando diversas características referentes à colaboração e sucesso dos projetos. Por fim, a quarta seção apresenta as considerações finais sobre a revisão literária realizada.

2.1 A Plataforma GitHub

Conforme descrito por KALLIAMVAKOU *et al.* (2014), o GitHub é um ambiente baseado na web de suporte ao desenvolvimento de código colaborativo, construído sobre o sistema de controle de versão *git*. Dentre as principais ferramentas oferecidas pela plataforma estão: armazenamento e versionamento de código, *issue tracking*, gestão de equipes de desenvolvimento, revisão colaborativa e mecanismos de interação social. De acordo com o próprio GitHub, em outrubro de 2016 a plataforma possui mais de 12 milhões de usuários e um acervo de mais de 38 milhões de projetos, entre *open source* e privativos, se caracterizando como o maior repositório de código online do mundo.

O GitHub é utilizado pelos desenvolvedores para armazenar e desenvolver diversos tipos de software, entre eles: ferramentas de desenvolvimento de software, motores para jogos, ferramentas de gestão, editores de texto e frameworks. É utilizado por

desenvolvedores independentes e empresas para armazenar tanto software livre, quanto privado.

O GitHub foi escolhido como fonte para esse estudo pelos seguintes motivos: (i) ele é a maior plataforma de armazenamento de código existente hoje e possui um acervo variado, com diversos tipos de projetos para análise; (ii) possui uma ferramenta de *issue tracking* bastante flexível, que possibilita que o acompanhamento de *issues* seja adequado às necessidades de cada projeto, utilizando marcação através de rótulos (*labels)* que podem ser definidos pelos colaboradores do projeto e o fechamento de *issues* através de um *commit.* Além disso, (iii) é uma plataforma que integrou funcionalidades sociais com sucesso às suas funcionalidades de suporte ao desenvolvimento, conforme apontado por LEE *et al*. (2013). Essas funcionalidades permitem que os desenvolvedores "sigam" (funcionalidade *follow*) outro desenvolvedor ou "assistam" (funcionalidade *watch*) outros repositórios, permitindo que as ações de desenvolvedores seguidos ou em repositórios assistidos sejam informadas para os usuários interessados.

2.2 GitHub e o Ambiente de Desenvolvimento Colaborativo

A plataforma GitHub tem sido alvo de diversos estudos durante os últimos anos. O objetivo principal desses estudos é entender qual o comportamento dos usuários que a utilizam, como esse comportamento pode influenciar no sucesso de um determinado projeto e o comportamento de outros usuários da plataforma. Nessa seção, serão apresentados estudos presentes na literatura de Engenharia de Software que analisaram a colaboração na plataforma GitHub.

DABBISH *et al.*(2012) investigaram o valor da transparência na colaboração entre os usuários e qual a sua influência sobre os envolvidos ao projeto. Os autores definiram transparência como a visibilidade das ações de outros usuários dentro de um ambiente colaborativo. Com base nesse conceito, buscaram responder as questões de pesquisa a seguir: quais conclusões podem ser feitas sobre o projeto ou pessoas quando a transparência está integrada a um ambiente de trabalho na web? E qual o valor da transparência para a colaboração em trabalhos baseados em conhecimento?

Durante o estudo foram identificadas diversas características referentes à colaboração transparente que geraram boas conclusões em relação ao interesse e o nível de comprometimento de um usuário, a qualidade do trabalho, experiência e competência de um desenvolvedor, a importância de um usuário, projeto ou ação para a comunidade, identificada através de ações que representam interesse de outros usuários (*following, watching e* comentários) e a relevância pessoal de um usuário para um projeto ou um grupo de usuários específico. Essas características são: o volume de atividade de um usuário ao longo do tempo, a sequência de atividades executadas por esse usuário, a atenção dada por outros usuários a um desenvolvedor ou repositório e o detalhamento de cada ação desse usuário. Segundo DABBISH *et al.* (2012), essas conclusões e a transparência no trabalho em grupo oferecem suporte à colaboração entre os desenvolvedores e participantes de um projeto, o aprendizado ao observar as ações de usuários mais experientes, o feedback rápido das ações executadas e a gestão da reputação na comunidade, possibilitando a autopromoção de um usuário ou o reconhecimento de um projeto ou grupo de desenvolvedores. Com isso, os autores concluíram que a transparência no ambiente colaborativo permite que os usuários do GitHub conheçam melhor outros desenvolvedores e tirem suas conclusões sobre o seu trabalho, possibilitando o reconhecimento, aproximação e *feedback* recorrente entre os usuários.

HELLER *et al.* (2011) apresentaram um estudo com uma análise geográfica dos relacionamentos entre os usuários do GitHub. Para isso, aplicaram técnicas de visualização para encontrar padrões dentro da comunidade de desenvolvimento, como: (i) efeito da distância geográfica no relacionamento entre os desenvolvedores; (ii) conectividade social e influência entre as cidades, que busca entender quais cidades se relacionam através do GitHub, através do número de usuários existentes nessa cidade e o número de relações entre essa cidade e as outras; e (iii) as variações nos estilos de contribuição (centralizada vs. distribuída) entre os projetos. Algumas observações relevantes desse artigo são: a maior concentração de desenvolvedores na Europa e nos EUA, fraca concentração de desenvolvedores no continente africano, o maior relacionamento de usuários da América do Sul com colaboradores da Europa (do que com os americanos) e a existência de núcleos de desenvolvedores em Nova York, São Francisco, Londres e Chicago.

Com foco de análise na localização geográfica, podemos citar também o estudo de RUSK *et al.* (2014), que construíram uma ferramenta web para identificar as tecnologias mais utilizadas pelos desenvolvedores de certas regiões do globo. Nesse estudo é utilizado como exemplo a cidade de Victoria, BC no Canadá. Os autores coletaram diversos dados sobre os usuários desta cidade e as tecnologias empregadas nos projetos em que estão envolvidos e elencaram as principais tecnologias utilizadas nesta cidade. Após a construção dessa ferramenta, os autores analisaram a sua utilidade e identificaram quais benefícios ela pode apresentar, elencando a possibilidade de encontrar desenvolvedores que trabalham com a mesma linguagem do usuário, a possibilidade de encontrar recursos para um projeto em determinada linguagem e a ajuda na definição da linguagem a se investir em uma região como principais ganhos com a utilização dessa ferramenta. Como exemplo para esse artigo, os autores citaram que na região de Victoria, BC no Canadá as principais linguagens utilizadas são JavaScript, Shell, Python, Ruby e CSS.

LEE *et al.* (2013) buscaram identificar como as ações dos desenvolvedores muito conectados e seguidos, caracterizados como *rockstars*, podem influenciar nas atividades daqueles que os seguem. Para isso, os autores coletaram informações sobre os usuários da plataforma GitHub e todos os tipos de ações que esses executaram durante um período de três meses. Ao final da coleta, foram identificados quatro desenvolvedores que foram classificados como *rockstars* por possuírem um número de contribuições bastante superior àqueles considerados como desenvolvedores padrões. De posse desses dados, os autores chegaram às seguintes conclusões: (i) *rockstars* realmente possuem maior influência em seus seguidores do que os desenvolvedores padrões, visto que os seguidores de *rockstars* costumam executar atividades subsequentes no mesmo projeto que eles com maior frequência do que os seguidores de desenvolvedores regulares; e (ii) algumas ações desses desenvolvedores afetam seus seguidores de forma diferente. Por exemplo, quando um *rockstar* faz um *fork* ou *watch* em um projeto, seus seguidores tendem a executar menos ações nesse mesmo projeto do que quando o *rockstar* comenta, abre um *issue* ou *pull request* em um projeto; (iii) a idade do projeto afeta a influência dos *rockstars* em seus seguidores, visto que a atividade de um *rockstar* em um projeto mais antigo é seguida por menos ações de seus seguidores do que as ações feitas por ele em projetos mais novos; (iv) o número de ações de um *rockstar* em um projeto afeta seus seguidores, visto que quanto mais ações um *rockstar* executa em um

projeto, mais seus seguidores são atraídos para esse projeto; e (v) os seguidores utilizam os *rockstars* como guias para projetos que sejam do seu interesse, visto que diversos seguidores são atraídos a projetos que o *rockstar* não é o dono, após uma ação sua. LEE *et al*. (2013) concluíram seu estudo identificando uma grande influência daqueles usuários classificados como *rockstars* no comportamento dos demais, apontando que as funcionalidades sociais oferecidas pelo GitHub criam diversas maneiras da comunidade desenvolvedora colaborar e influenciar nas atividades dos seus membros.

WEICHENG *et al.* (2013) exploraram padrões nos *commits* feitos pelos desenvolvedores no GitHub tentando buscar uma relação entre esses padrões e a evolução do software. Entre as questões investigadas podemos citar: (i) por que os *commits* acontecem com uma frequência específica? e (ii) podemos prever a evolução do software no GitHub a partir das atividades dos desenvolvedores? WEICHENG *et al.* (2013) identificaram dois padrões na evolução do software, sendo eles: (i) arquivo alterado em uma versão tende a afetar um arquivo dependente dele em uma próxima versão; e (ii) o tempo médio para a ocorrência de um "grande *commit*", definido como um *commit* em que mais de cinco arquivos tiveram mais de cinco linhas alteradas é três vezes maior que o tempo entre *commits* normais. Usando os padrões apresentados, os autores afirmam poder prever o próximo *commit* do projeto e quais arquivos podem ser alterados nesse *commit*, gerando assim informações para apoiar o planejamento da evolução dos softwares hospedados na plataforma GitHub.

WEST *et al.* (2012) utilizaram um algoritmo de reconhecimento de reputação utilizado em páginas Wiki, chamado de WikiTrust[2], para reconhecer a reputação dos desenvolvedores em repositórios de código. O algoritmo se baseia em identificar no texto as alterações feitas por determinado autor (*commiter*) que se mantiveram por um maior número de revisões posteriores, aumentando assim a reputação e confiabilidade desse desenvolvedor. Ao final, os autores concluíram que o algoritmo precisa passar por ajustes para se adequar ao cenário de análise de código e não de texto livre, devido às diferenças entre repositórios de código e de conteúdo. Entre esses ajustes podemos citar: o tratamento de comentários no código e se isso deve contar ou não para a reputação, se a existência de linhas em branco deve influenciar na análise, se a movimentação de um mesmo código entre os arquivos ou o isolamento de uma função devem influenciar a

[2] http://wikitrust.soe.ucsc.edu/

análise de reputação. Porém, ainda assim o algoritmo foi considerado promissor, pois conseguiu identificar com sucesso grande parte dos eventos que caracterizam ganho ou perda de reputação.

GOUSIOS (2013, 2014) e IZQUIERDO *et al.* (2015) apresentaram análises sobre duas ferramentas, o GHTorrent[3] e o GiLA[4], respectivamente. A primeira é um repositório de dados que visa facilitar a obtenção de informações sobre os projetos públicos do GitHub para pesquisadores interessados em minerar e estudar os dados de projetos presentes na plataforma sem utilizar a API disponibilizada pelo GitHub. GiLA, conforme descrito por IZQUIERDO *et al.* (2015), é uma ferramenta que gera diversas visualizações para facilitar a análise de *issues* em um projeto através da uma classificação baseada em rótulos, apoiando na identificação de padrões e evolução dos *issues* de cada categoria. Ambas as ferramentas servem como um mecanismo para gerar conhecimento e possibilitam que pesquisadores acessem dados sobre a plataforma GitHub, que podem ser estudados e analisados, visando compreender diversos aspectos da utilização do GitHub e do desenvolvimento de softwares livres.

FARAH *et al.* (2013) apresentaram uma ferramenta, chamada Archnalyser, que busca reduzir o tempo e esforço necessários para encontrar componentes ou aplicações de alta qualidade e que atendam às necessidades de determinado produto. Essa ferramenta se aplica em um cenário onde a estratégia adotada para reduzir custo, esforço e tempo para construção de um software é o reuso de componentes configuráveis e utilizáveis em diversos contextos, estratégia conhecida como "*componente-based software enginnering*" (CBSE), conforme citado por ANDREOU *et al.* (2015). A ferramenta utiliza a base do GitHub para buscar, através de palavras chave, os repositórios que possuam algum software que atenda a um conjunto de critérios e analisa esses repositórios para disponibilizar informações que apoiem a escolha do usuário.

KALLIAMVAKOU *et al.* (2014) apresentaram um estudo sobre as promessas e os perigos que podem ser encontrados ao analisar os dados disponíveis na plataforma GitHub. Para isso, os autores executaram um *survey* com 1.000 usuários do GitHub e uma análise quantitativa e qualitativa dos dados coletados. Entre as principais

[3] http://ghtorrent.org/
[4] https://github.com/SOM-Research/gila

promessas que justificam a utilização do GitHub como base para estudo, podemos citar: (i) os relacionamentos entre desenvolvedores, *pull requests, issues* e *commits* oferecem uma visão compreensiva das atividades de desenvolvimento, e (ii) o GitHub provê uma fonte de dados para estudo de revisão de código bastante grande e diversificada. Por outro lado, vários são os perigos identificados, que devem ser observados por pesquisadores que buscam estudar os dados contidos na plataforma. Entre os perigos, podemos citar: (i) repositórios não são necessariamente projetos, pois existem alguns repositórios que são usados para armazenamento de arquivos ou como biblioteca de artigos; (ii) a maioria dos repositórios apresenta poucos *commits* para análise; (iii) a maioria dos projetos estão inativos por bastante tempo; e (iv) muitos projetos ativos não executam todas as atividades de desenvolvimento no GitHub (testes e *issue tracking* são exemplos).

THUNG *et al.* (2013) analisaram a estrutura de rede de codificação social formada na plataforma GitHub. Os autores analisaram os relacionamentos entre projetos e entre desenvolvedores. Utilizando a API disponibilizada pelo GitHub, os autores analisaram aproximadamente 100.000 projetos e 30.000 desenvolvedores, com base no algoritmo *PageRank*, utilizado pelo Google para identificar as páginas web mais importantes para a sua ferramenta de busca. Após a análise, os autores identificaram que as redes de projetos são mais interconectadas que as redes de desenvolvedores. Além disso, os autores concluíram que as funcionalidades sociais disponibilizadas pelo GitHub realmente melhoram a colaboração entre desenvolvedores, visto que os desenvolvedores podem acompanhar as ações de seus companheiros apoiando na solução de problemas identificados e na evolução dos projetos em que eles estão empenhados. Também utilizando o *PageRank*, que identifica os nós mais interconectados em uma rede, os autores identificaram os projetos mais importantes armazenados no GitHub. São eles: mxcl/homebrew, rails/rails, lifo/docralis, joyent/node e rubinius/rubinius, onde os nomes dos projetos são formados por usuário/repositório. Além disso, também identificaram os usuários mais importantes na rede, que são Joshua Peek, Aman Gupta, Steve Richert, Michael Klishim e Josh Kalderimis.

VENKATARAMANI *et al.* (2013) implementaram um estudo visando criar grupos de projetos com base na rede formada pelas conexões entre seus desenvolvedores. Esses grupos foram mapeados, através de uma taxonomia criada pelos autores, em domínios

de acordo com uma análise semântica de cada repositório. Essa análise é baseada na taxonomia criada, que identifica os termos relacionados ao mesmo domínio de software. Alguns domínios identificados entre os projetos analisados são redes sociais, representação de conhecimento, jogos e computação em nuvem. A técnica apresentada pelos autores pode ser utilizada para identificar projetos semelhantes, apoiar no recrutamento de profissionais capacitados para um projeto (através da identificação dos profissionais que atuam no mesmo grupo) e na medição da popularidade de um projeto na comunidade de desenvolvimento.

YU *et al.* (2014) apresentaram um estudo da evolução dos sistemas de controle de versão até a criação do GitHub. Além disso, buscaram identificar os motivos do crescimento da plataforma nos últimos anos e as influências que possui na comunidade de desenvolvimento de software. O artigo apresenta um histórico da evolução dos sistemas de controle de versão, iniciada pelo CVS (*Concurrent Versioning System*), sistema cliente-servidor que permitia ao usuário armazenar versões do seu software, criando um *check out* dessa versão indicando que ela está sendo alterada por algum usuário e, posteriormente, criando *check in* com uma nova versão. Similar ao CVS, o Subversion[5] também possibilitava a gestão da evolução do código, porém apresentava vantagens, como suporte a arquivos de formato binário e *commits* atômicos. Seguindo com a evolução dos sistemas de controle de versão, os autores citam o Git, um sistema que possui como principal característica o suporte ao desenvolvimento distribuído, importante característica para os softwares de código aberto. Após apresentar a linha do tempo dos sistemas de controle de versão, YU *et al.* (2014) apresentaram uma análise empírica sobre a plataforma GitHub. Durante essa análise, os autores apresentaram diversos números sobre o GitHub, como quantidade de acessos, eventos por dia, projetos em maior atividade, entre outros. Com essa análise, os autores buscaram justificar o crescimento da plataforma e os seus impactos no desenvolvimento distribuído de software.

2.3 *Issue Tracking* e as suas Implicações

Issue tracking é um assunto que tem levantado interesse de diversos autores. Esses estudos abordam o assunto de diversos pontos de vista, principalmente buscando

[5] https://subversion.apache.org/

identificar qual a importância do registro e correção de *issues* para o ciclo de vida e sucesso de um projeto ou software. Os principais tópicos abordados em estudos sobre *issue tracking* são as características que influenciam os usuários a fazer uso das ferramentas de *issue tracking*, modelos de predição do tempo de correção ou de abertura de um *issue*, características dos usuários de registram e corrigem *issues*, entre outros. Nessa seção, serão apresentados estudos presentes na literatura que analisaram mais detalhadamente esse assunto.

XAVIER *et al.* (2014) apresentaram uma análise sobre as diferenças entre os usuários que tipicamente registram defeitos (*reporters)* e aqueles que os corrigem (*assignee*). Para essa análise, os autores utilizaram como base 90 projetos armazenados no GitHub, que em conjunto possuíam mais de 150.000 *issues* cadastrados. De todos os *issues* encontrados, o artigo foca naqueles que são classificados como defeitos (*bugs*), reduzindo a amostra para aproximadamente 57.000 *issues*.

Após diversas análises sobre os dois perfis de usuários que participam do ciclo de vida de um *issue*, os autores puderam identificar que o número de colaboradores de cada tipo e o perfil desses colaboradores é bastante diferente. Segundo XAVIER *et al.* (2014), enquanto os *reporters* não tendem a participar efetivamente na equipe de nenhum projeto, grande parte dos *assignees* participam de pelo menos um projeto no papel de membro da equipe de desenvolvimento. Além disso, os *assignees* tendem a contribuir muito mais com a evolução dos softwares através de *commits*, possuem um número geralmente maior de seguidores e um tempo de registro no GitHub maior do que os usuários categorizados como *reporters.*

Apesar dos *assignees* possuírem uma participação mais substancial na evolução de projetos que na identificação de *issues*, o número de usuários nessa categoria é bem menor do que aqueles identificados como *reporters*. Essa diferença pode ser justificada devido às atividades mais especializadas desempenhadas pelos *assignees*, enquanto para um *reporter* é mais fácil apenas identificar possíveis defeitos e registrá-los para futura análise e correção. Ainda segundo XAVIER *et al.* (2014), é possível observar que a popularidade de um usuário é influenciada pela participação deste na evolução de um software através de *commits*. Porém, essa popularidade também é influenciada pela popularidade identificada em plataformas externas ao GitHub. Consequentemente, a

popularidade e o sucesso dos projetos em que o mesmo está envolvido também sofrem influência da popularidade desse desenvolver em outras plataformas sociais.

CABOT *et al.* (2015) avaliaram a utilização do mecanismo de categorização de *issues*, os rótulos, dentro da plataforma GitHub. As questões de pesquisa investigadas pelo artigo são: quantos rótulos são utilizados no GitHub? Quantos rótulos são utilizados por projeto? Quais são os mais populares? A utilização de rótulos influencia a evolução do software?

Respondendo às questões de pesquisa, CABOT *et al.* (2015) concluíram que apenas 3,25% dos 3.757.038 projetos avaliados possuíam algum *issue* categorizado através de um rótulo. Desse grupo de projetos, aproximadamente 70% utilizam no máximo dois rótulos distintos, enquanto apenas 2,6% utilizam mais de 10 rótulos. Além disso, 70% dos *issues* que são categorizados com rótulos estão presentes em projetos com até 100 *issues* cadastrados. Ao investigar os principais rótulos utilizados, os autores identificaram as seguintes: *enhancement* (melhoria), *bug* (defeito), *question* (questão), *feature* (funcionalidade) e *documentation* (documentação), que mostra que os rótulos são utilizados de maneira flexível pelos projetos, desde o apoio na identificação de possíveis problemas até a identificação de necessidade de documentação ou novas funcionalidades. Ao investigar o impacto da utilização de rótulos na gestão dos *issues* de um projeto, os autores identificaram que os projetos que possuíam rótulos apresentavam um maior envolvimento de pessoas na solução e discussão dos problemas, além de um aumento dos *issues* fechados, assim mostrando que a utilização de rótulos tende a influenciar positivamente na evolução do projeto.

FEJZER *et al.* (2014) dividiram o seu estudo em três tópicos principais. O primeiro busca identificar quais as características da equipe de um projeto que contribuem para a evolução do software. Nesse tópico, os autores identificaram a existência de um *Core Team* dentro dos projetos avaliados, onde apenas um número reduzido de colaboradores é responsável pela maioria dos desenvolvimentos substanciais existentes no software, enquanto um grande número de outros colaboradores participa de forma pontual, contribuindo com pequena parte da codificação. O segundo tópico abordado é referente à correção de *bugs*, onde os autores identificaram, através da busca por palavras chave dentro dos *commits*, que grande parte dos *commits* é feita para correção de defeitos encontrados no software e que isso se deve ao grande número de pessoas que podem

contribuir com o repositório registrando *issues*. Porém, foi observado que o uso do mecanismo de vínculo entre *issues* e *commits* presente no GitHub é pouco aproveitado. Por fim, como terceiro tópico, os autores identificaram que os projetos presentes na amostra não demonstraram etapas de desenvolvimento bem definidas, indicando que a evolução desses softwares é feita de forma contínua e incremental. O ponto fraco das análises feitas pelos autores é que a base utilizada é muito pequena para gerar conclusões fortes, visto que o estudo foi realizado sobre apenas 43 projetos.

Assim como FEJZER *et al.* (2014), o estudo feito por KO *et al.* (2010) também identificou que existe um grupo de usuários dentro dos projetos que contribui com o registro e correção de *issues* de forma mais frequente que a maioria dos usuários comuns. Além disso, KO *et al.* (2010) reforçam que uma grande vantagem obtida pela liberação do registro de *issues* para qualquer usuário em repositórios públicos é a possibilidade de identificação de talentos entre os desenvolvedores e *reporters*. Um *reporter* identificado como talento pode, posteriormente, ser recrutado para participar da equipe *core* do projeto.

GIGER *et al.* (2010) abordaram o tema *issue tracking* buscando criar um modelo de predição do tempo de correção de um *issue*. Para isso, os autores criaram uma árvore de decisão capaz de classificar os *issues* entre correção "rápida" ou "demorada", de acordo com os atributos de cada *issue.* A árvore criada foi aplicada em um experimento envolvendo seis projetos, divididos em três grupos: Eclipse, Mozilla e Gnome. Nesses experimentos, os autores comprovaram a capacidade do modelo de classificar corretamente entre 60% e 70% dos *issues* e, além disso, foram capazes de apresentar os atributos de maior relevância para previsão do tempo de correção de um *issue* de cada projeto usado no experimento. Os autores também identificaram que a inclusão de dados após o cadastro do *issue* influencia positivamente no desempenho do modelo de predição, podendo aumentar a sua precisão entre 5% e 10%.

Seguindo o mesmo objetivo, MARKS *et al.* (2011) criaram um modelo de predição para o tempo de correção de um *issue*. Ao contrário de GIGER *et al.* (2010), esse estudo utilizou três dimensões no seu modelo de predição, que são compostas por atributos e métricas especificas. São elas: (i) localização do defeito, que engloba atributos como o componente, a versão e o sistema operacional onde foi identificado o defeito; (ii) o *reporter* do defeito, que engloba atributos como popularidade, tipo e tempo levado para

correção do *reporter* do defeito; e a (iii) descrição do *issue*, que engloba atributos como severidade, prioridade, quantidade de código e clareza da descrição.

Assim como GIGER *et al.* (2010), esse artigo propôs a utilização de uma árvore de decisão para a construção do modelo de predição. Essa árvore é chamada de *Random Forest.* No estudo de caso com os projetos Mozilla e Eclipse, os autores executaram a árvore de decisão isoladamente para cada dimensão, identificando qual dimensão apresentava o melhor desempenho na previsão do tempo de correção de um *issue* e quais atributos da dimensão sob análise possuíam maior impacto na capacidade de previsão. Para o projeto Mozilla, as dimensões que apresentaram melhor desempenho são localização, descrição e *reporter*, nessa ordem. Para o projeto Eclipse, a ordem de melhor desempenho é localização, *reporter* e descrição. Diferente do proposto inicialmente pelos autores, que acreditavam que a dimensão descrição seria a que mais influenciaria para a previsão, ela não apresentou um bom desempenho, sendo a segunda colocada no projeto Mozilla e terceira no projeto Eclipse. Principalmente no caso do projeto Eclipse, o desempenho dessa dimensão foi bem abaixo do esperado, o que indica que é necessária uma melhoria dos dados cadastrados na abertura de um defeito, visando aumentar a assertividade do modelo de predição.

Usando o modelo de predição proposto, MARKS *et al.* (2011) alcançaram uma taxa de sucesso na previsão do tempo de correção de um defeito de aproximadamente 65% e comprovaram que o modelo proposto possui um desempenho melhor do que modelos randômicos. Além disso, foi observado pelos autores que atributos como "prioridade" e "severidade" de um defeito possuem pouca influência no seu tempo de correção em grandes softwares de código aberto, o que contradiz o senso comum de que defeitos com maior prioridade ou severidade tendem a ser atacados de maneira mais rápida.

Outro estudo que apresenta um modelo de predição do tempo de correção de um *issue* foi apresentado por WEISS *et al.* (2007). Nesse estudo, diferente dos anteriores, os autores não utilizaram uma árvore de decisão para executar a previsão do tempo de correção de um *issue.* Para isso, foi utilizado um framework chamado Lucene, desenvolvido pela Fundação Apache, que contém um mecanismo de medição da similaridade de textos. Com base nesse *framework*, os autores utilizaram a abordagem do vizinho mais próximo para identificar outros *issues* que mais se assemelhavam com o *issue* em questão, através dos textos de suas descrições. Ao identificar *issues*

candidatos entre os vizinhos, o modelo de predição utiliza o tempo de correção desses *issues* como parâmetro para prever o tempo de correção de um novo *issue* aberto.

Após a execução de um experimento feito sobre uma base de 567 *issues* do projeto JBoss, esse modelo de predição, segundo as conclusões de WEISS *et al.* (2007), foi capaz de se aproximar do esforço real necessário para correção dos *issues*. Para *issues* classificados como defeito, essa previsão possui uma margem de erro de apenas uma hora. Esse resultado impressionou os autores, levando em consideração que o modelo de predição proposto utiliza apenas dois campos para análise: título e descrição do *issue.*

KENMEI *et al.* (2008) também apresentaram um modelo de predição, porém não focado em prever o tempo de correção de um *issue*, mas o crescimento da demanda de solicitações de mudança (cadastro de *issues*) ao longo do ciclo de vida do projeto. A importância em analisar o crescimento do cadastro de *issues* é justificada pela possibilidade de identificar, através dessa análise, aspectos como o aumento ou redução do interesse da comunidade, a solicitação de novas funcionalidades, a maturidade do software ou a redução de sua qualidade.

Para esse estudo, os autores utilizaram um modelo matemático chamado ARIMA (*AutoRegressive Integrated Moving Avarege*) e buscaram provar que esse modelo tem um desempenho melhor na previsão de tendências do que modelos simples, como regressão linear e busca randômica. Para isso, o estudo apresenta três questões de pesquisa. São elas: Para quais cenários o modelo ARIMA é uma abordagem válida para prever a evolução de solicitações de mudança? Qual a precisão do modelo ARIMA na previsão da evolução das solicitações de mudança? A densidade de solicitações de mudança para um sistema especifico possui uma tendência particular?

KENMEI *et al.* (2008) identificaram que o modelo ARIMA apresenta desempenho melhor do que os modelos simplificados de previsão. Além disso, identificaram que os erros de previsão cometidos pelo modelo são menores que 25% para a maioria dos casos estudados, comprovando que o modelo pode ser utilizado em casos reais, visto que um erro menor que 25% pode aceitável em diversos cenários. Por fim, os autores não foram capazes de identificar tendências em alguns casos estudados, concluindo que a densidade de solicitações de mudança abertas não está relacionada ao crescimento ou redução do tamanho do sistema.

BISSYANDE *et al.* (2013) estudaram o assunto *issue tracking systems* de um modo mais abrangente, buscando identificar características que diferenciam os projetos que adotaram a utilização de *issue tracking* daqueles que não utilizam essa ferramenta no GitHub. Para o estudo, os autores utilizaram uma amostra de 100.000 projetos, dos quais foram coletadas diversas informações como: idade do projeto, tamanho da equipe de desenvolvimento, linhas de código, popularidade do dono do repositório, entre outras.

De posse dessa amostra, BISSYANDE *et al.* (2013) realizaram um estudo empírico, visando responder diversas questões de pesquisa são elas: Qual a proporção de projetos que recebem *issues* e como eles podem ser diferenciados sob esse aspecto? Quantos issues são registrados em projetos onde *issue tracking* é adotado? Qual a popularidade da utilização de rótulos? Quem registra *issues* em um projeto? Quantos deles fazem parte da equipe de desenvolvimento? Qual a relação entre a utilização do *issue tracking* e o sucesso do projeto? O tamanho da equipe do projeto tem relação com o tempo de resolução de um *issue*?

O artigo de BISSYANDE *et al.* (2013) foi replicado durante a execução do presente estudo e os seus resultados e conclusões serão apresentadas detalhadamente nos próximos capítulos. Esses resultados serão comparados com os encontrados durante a sua replicação, com o objetivo de verificar a manutenção ou não das conclusões apresentadas pelos autores. O presente estudo utilizou o artigo de BISSYANDE *et al.* (2013) como base e irá apresentar novas hipóteses e questões de pesquisa, visando acrescentar mais informações à base de conhecimento sobre o assunto issue tracking.

O estudo de KIKAS *et al.* (2015) foca em compreender como se comporta o ciclo de vida de um *issue* durante o tempo de um projeto. Para isso, os autores estudaram uma base de 4.000 projetos armazenados no GitHub e utilizaram seus dados para responder três questões de pesquisa. São elas: Qual a média de criação de *issues* e como ela muda durante o tempo? Como o número de *issues* abertos e pendentes evolui com o tempo? Qual a média do tempo de vida de um *issue* e como ela muda durante o tempo?

Como conclusões do estudo, os autores observaram que a média de cadastro de *issues* é maior durante os primeiros meses após a criação do repositório e tende a decrescer com o passar do tempo. Por outro lado, a quantidade de *issues* pendentes

tende a aumentar constantemente com o passar do tempo, principalmente devido aos *issues* que são abertos e não mais fechados durante o período de observação do estudo. Por fim, os autores concluíram que o tempo de vida de um *issue* se mantém estável durante o ciclo de vida do projeto.

2.4 Considerações Finais

O GitHub tem despertado o interesse de diverso estudo nos últimos anos devido as suas características altamente colaborativas e sociais, sendo um dos poucos repositórios de código que, além de possuir funcionalidades voltadas para o desenvolvimento, como: *pull requests*, *issue tracking* e *forks*, também possui diversas outras funcionalidades voltadas para a interação social entre os usuários, que modificam a forma como o colaborador interage como o projeto e com outros colaboradores.

Diversos dos estudos apresentados nesse capítulo apresentam análises interessantes sobre a colaboração e o ambiente de desenvolvimento colaborativo. Nesses estudos, pudemos identificar as características dos usuários existentes no GitHub, e como as suas ações dentro da plataforma impactam outros usuários e outros repositórios. Além disso, alguns estudos analisaram os relacionamentos entre os projetos, usuários e equipes, focando em análises geográficas ou de interação social.

O foco da segunda seção desse capítulo foi apresentar estudos voltados para o entendimento da utilização do *issue tracking* e seu impacto na evolução do software. Pudemos observar que a ferramenta de *issue tracking* oferecida pelo GitHub também possui características sociais que influenciam a forma como os *reporters* e os *assignees* atuam na resolução de *issues* do software.

Além disso, também observamos estudos focados na criação de mecanismos que facilitem a gestão de *issues* em projetos, apoiando na previsão de criação de issues, na previsão do tempo necessário para sua correção e na previsão do impacto de um *commit* no próximo ciclo de evolução do software.

No próximo capítulo, iremos apresentar a proposta de estudo que foi elaborada durante a execução do estudo em questão, quais as hipóteses que iremos utilizar para análise e quais as questões de pesquisa iremos responder, contextualizando assim o

objetivo desse estudo. Além de apresentarmos os resultados da replicação feita do artigo de BISSYANDE *et al.* (2013).

3. Metodologia e Coleta de Dados

Este capítulo descreve o processo de coleta e análise dos dados utilizados neste estudo. Apresentaremos as ferramentas utilizadas, os dados coletados e suas relações, o passo a passo do processo de coleta e análise dos dados e as principais dificuldades enfrentadas durante o andamento do trabalho.

3.1 GitHub e Ferramentas de Coleta e Análise de Dados

O GitHub é uma plataforma de suporte ao desenvolvimento de software que disponibiliza diversas funcionalidades para seus usuários, entre elas o armazenamento e versionamento de código, mecanismos de interação social, criação de equipes de desenvolvimento e uma ferramenta de *issue tracking*, foco principal deste estudo.

Este trabalho propõe uma análise sobre a utilização da ferramenta de *issue tracking* disponibilizada pelo GitHub. Para tanto, o primeiro passo foi coletar os dados relevantes ao estudo. Para isso, utilizamos uma API[6] disponibilizada pelo GitHub que possibilita a interação com a plataforma através de linha de comando. Entre as funcionalidades oferecidas por essa API, podemos citar a criação de repositórios, a cópia de um repositório para um diretório local, os comandos de *commit*, *merge*, *pull request* e *fork*, além da consulta dos diversos dados existentes na plataforma.

Porém, levando em consideração que um dos objetivos desse estudo é realizar sua análise utilizando uma grande amostra de projetos, foi necessário identificar um método para automatizar as consultas realizadas através dessa API. Para isso, utilizamos um framework chamado Egit[7], disponível para o ambiente de desenvolvimento Eclipse[8], que possibilita a utilização da do sistema de controle de versionamento Git através da

[6] https://developer.github.com/
[7] https://github.com/eclipse/egit-github/tree/master/org.eclipse.egit.github.core
[8] https://eclipse.org/

linguagem de programação Java[9]. Visto que esse sistema é utilizado pelo GitHub, o Egit pôde ser utilizado para acessar as informações dos repositórios armazenados no GitHub.

Utilizando estas ferramentas, foram desenvolvidas rotinas para a coleta de dados diretamente da base do GitHub. Estas ferramentas, além de realizar a coleta inicial de dados, também executavam os primeiros tratamentos nos dados coletados. Entre os dados necessários, que serão detalhados na próxima seção, a informação da quantidade de linhas de código-fonte (LOC) presente em um repositório não é disponibilizada nativamente pela API. Para obter essa informação foi necessário copiar cada repositório da amostra para um diretório local e utilizar uma ferramenta de contagem de linhas de código. A ferramenta utilizada nesse estudo foi a cLOC[10]. Não foi utilizada a ferramenta SLOCCount, conforme o estudo original, pois essa ferramenta está disponível apenas para ambiente Linux e o estudo atual foi realizado em um ambiente Windows.

O GitHub possui uma política que restringe a quantidade de requisições feitas por um mesmo usuário através de sua API para 5.000 por hora. Essa característica foi a principal dificuldade enfrentada durante a coleta de dados, pois tornou o processo mais lento devido ao tempo de espera sempre que o limite de requisições era atingido. Devido a essa característica, optamos por não realizar a análise de *commits* durante a coleta de dados, visto que cada *commit* analisado exigiria uma nova requisição. Assim, além do número de linhas de código, coletamos o *log* de *commits* copiando cada repositório para um diretório local. Diferente da consulta pelos *commits* através da API, uma única requisição por repositório é necessária para a geração do *log* e não uma por *commit*, reduzindo o tempo de coleta. Como a análise de *commits* não foi realizada durante a coleta de dados usando a API, foi necessário a construção de uma segunda rotina, que foi executada após a coleta inicial de dados com o objetivo de analisar os *commits* presentes nos arquivos de *log* e complementar os dados de cada repositório com as informações referentes a esses *commits.*

De posse dos dados necessários e após o tratamento dos arquivos, foi utilizada a ferramenta R[11] para a realização das análises estatísticas que geraram as conclusões apresentadas nesse estudo.

[9] https://www.java.com/pt_BR/
[10] http://cloc.sourceforge.net/
[11] https://www.r-project.org/

3.2 As Entidades e Dados Coletados

As análises realizadas nesse estudo foram executadas com uma amostra de aproximadamente 220.000 projetos do GitHub, coletados aleatoriamente da plataforma GitHub. Visando compreender as características que envolvem a utilização da ferramenta de *issue tracking* disponibilizada pelo GitHub, analisamos as características presentes em três entidades existentes nessa plataforma. São eles: o projeto, o *issue* e o colaborador. Abaixo, estão descritos os dados coletados para análise, divididos pelas entidades as quais eles pertencem.

a. Projeto: foram coletadas informações sobre a idade do projeto, o dono do repositório, a principal linguagem de desenvolvimento, se possui a funcionalidade de *issue tracking* ativa, tamanho do projeto em KB, seu número de linhas de código, número de *issues* abertos, fechados, fechados por um *commit*, número de colaboradores (sendo eles desenvolvedores, *reporters* de *issues* ou ambos), se o projeto é um *fork* ou não e o tempo médio de correção de *issues*.

Para indicar o sucesso e popularidade de um projeto utilizamos o número de "*Watchers*" e "*Forks*". "*Watcher*" é o termo utilizado para descrever usuários que "seguem" determinados repositórios, recebendo atualizações e relatórios sobre a evolução do software. A quantidade de "*watchers"*" serve com indicador da quantidade de atenção dada a um determinado projeto pela comunidade de desenvolvedores (BISSYANDE *et al.*, 2013).

"*Fork*" é o termo utilizado para aqueles projetos que são criados com base em um projeto já existente, ou seja, quando um projeto é copiado para um novo ramo de desenvolvimento, independente do original. A quantidade de "*forks*" serve como um indicador da atividade e envolvimento da comunidade de desenvolvimento no crescimento da base de código do projeto e melhoria da sua qualidade (BISSYANDE *et al.* 2013).

b. Issues: foram coletadas informações sobre todos os *issues* cadastrados em cada projeto, abertos ou fechados, o usuário responsável pelo registro do *issue*, os rótulos (*labels*) relacionados a eles, seu tempo de correção e se ele foi fechado por um *commit*;

c. **Colaboradores:** para todo usuário que interagiu pelo menos uma vez com algum projeto da amostra foi coletado o nome, o login, o e-mail, se esse colaborador fez *commit* ou registrou *issue* e se ele fechou algum *issue* através de um *commit*. Também classificamos os colaboradores de acordo com a sua interação com um projeto. Classificamos como *reporters* aqueles que nunca fizeram um *commit* e apenas contribuíram com o registro de *issues,* já colaboradores que fizeram pelo menos um *commit* foram classificados como desenvolvedores.

3.3 Curadoria dos Dados

A curadoria da amostra de dados se baseou em duas premissas. A primeira foi utilizada pelo artigo original e sugere retirar da amostra todos os projetos em que a ferramenta de *issue tracking* estiver desativada. Essa premissa foi assumida devido aos casos em que a gestão de *issues* era feita em ferramentas externas à plataforma. Assim como no artigo original, o estudo atual retirou da amostra todos os projetos onde essa ferramenta está desativada. Este conjunto representa um total de 97.286 projetos dos 220.010 projetos coletados, restando assim 122.724 projetos na amostra. Caso algum desses projetos utilize uma ferramenta externa para registro de *issues* e mantenha a funcionalidade do GitHub ativa, este projeto não foi retirado da amostra pela curadoria.

Uma segunda premissa assumida neste estudo, mas não utilizada no artigo original, foi aplicada com o objetivo de garantir que os repositórios analisados armazenavam o código-fonte de um software. Para isso, foram retirados da amostra todos os projetos em que não foi identificada a presença de nenhuma linguagem de programação. Dessa forma, excluímos da amostra repositórios que estavam vazios ou serviam apenas para armazenamento de documentação. Neste sentido, foram retirados mais 45.815 projetos da amostra, restando 76.909 projetos para a execução das análises, o que representa 35% do total de projetos inicialmente coletados.

3.4 Hipóteses e Questões de Pesquisa

Conforme proposta desse trabalho, as hipóteses e questões de estudo utilizadas como parte da análise serão as mesmas utilizadas pelo artigo original (dentro do que se pode

inferir pelo conteúdo do artigo), visando garantir a replicação mais fiel possível das análises realizadas no ano de 2013. As hipóteses criadas no estudo original são:

H1. Projetos open-source recebem grande um número de *issues*.

H2. Durante o ciclo de vida de um projeto, desenvolvedores constroem código, enquanto uma comunidade distinta de usuários reporta *issues*.

H3. Ambientes colaborativos de desenvolvimento de software geram um impacto positivo no registro de *issues*.

H4. Quanto mais pessoas estiverem interessadas em um projeto e identificarem *issues*, mais rápido eles serão solucionados.

Com base nestas hipóteses e na amostra de dados coletada, foram investigadas as seguintes questões de pesquisa, buscando explorar diferentes aspectos existentes em um projeto de software e suas relações com as atividades de *issue tracking*. As seis primeiras questões de pesquisa foram retiradas do estudo replicado. Sendo assim os resultados identificados para essas questões serão comparados àqueles encontrados no estudo original.

RQ1. *Qual a proporção de projetos que recebem issues e como projetos que recebem issues podem ser diferenciados daqueles que não recebem com base em suas características?*

Com essa questão, buscamos estudar a adoção de *issue tracking* na amostra em estudo e as características dos projetos que recebem *issues*. Procuramos evidenciar as características que diferenciam projetos que possuem *issues* registrados daqueles que não possuem, entre elas estão a idade do projeto, a quantidade de desenvolvedores, o número de linha de códigos de um projeto e a quantidade de seguidores do dono do projeto. Assim, entenderemos se existe diferença significativa para projetos com e sem *issue* em relação a essas características e qual é essa diferença.

RQ2. *Quantos issues são mapeados em projetos onde issue tracking é utilizado?*

Com essa questão, buscamos investigar a utilização de *issue tracking* nos projetos da amostra. Quantificaremos número de projetos que possuem *issues*

registrados, quais as quantidades mais recorrentes de *issues* registrados em projetos, a distribuição da quantidade para cada 1.000 linhas de código e as principais linguagens usadas em projetos onde são registrados *issues*.

RQ3. *Qual a popularidade da utilização de rótulos? Quais são os principais rótulos utilizados em projetos de desenvolvimento de software?*

Nessa questão será discutida a utilização de rótulos para classificação de *issues.* Observaremos a quantidade de rótulos, a quantidade de rótulos distintos, os agruparemos pela sua semelhança e identificaremos os grupos mais utilizados. Com base nos rótulos, identificaremos os principais motivos pelos quais os usuários registram *issues* em um projeto.

RQ4. *Quem cadastra issues? Quantas destas pessoas fazem parte da equipe de desenvolvimento?*

Esta questão de pesquisa investiga a comunidade de usuários que registra *issues,* buscando identificar se é mais comum o registro de *issues* pelos desenvolvedores de um projeto ou por usuários que não participam diretamente do desenvolvimento do software e contribuem apenas com o registro de *issues.*

RQ5. *Qual a relação entre a utilização de issue tracking e o sucesso do projeto?*

Esta questão de pesquisa visa estabelecer uma correlação entre o sucesso de um projeto, medido através no número de *watchers* e *forks* existentes no mesmo, e a utilização de *issue tracking* por parte de seus colaboradores, medido através do número de *issues* registrados e de *reporters* de *issues*.

RQ6. *O tamanho da comunidade de usuários afeta o tempo de correção de um issue?*

Esta questão de pesquisa investiga a correlação entre o tempo de correção de *issues* e o número de *reporters* em um projeto. Buscamos identificar se esse tempo tende a ser menor que em projetos com maior número de colaboradores que registram *issues*.

Incluiremos nessa questão de pesquisa algumas análises que não estão presentes no artigo original, visando identificar se existem características em um *issue* que

influenciam na velocidade do seu encerramento. As características avaliadas são a existência de rótulos, se esse *issue* possui um dos rótulos mais utilizados e a quantidade de comentários recebidos.

Encerrada a replicação do trabalho de BISSYANDE *et al.*, (2013), iniciamos a partir da quinta hipótese as análises referentes ao segundo objetivo desse trabalho, buscando compreender quais são as características dos *issues* e colaboradores envolvidos com a funcionalidade de encerramento de *issues* através de *commits.* As novas hipóteses foram formuladas visando direcionar esse estudo e estão relacionadas abaixo.

H5. Usuários mais seguidos, mais antigos ou que possuem mais repositórios públicos são os que mais registram *issues*.

H6. *Issues* encerrados por *commit* são geralmente relacionados à correção de erros, melhorias ou novas funcionalidades adicionadas ao código-fonte.

H7. Desenvolvedores que encerram *issues* através de *commits* têm maior participação na evolução do código-fonte.

A partir da sétima questão de pesquisa analisaremos os tópicos referentes ao segundo objetivo desse estudo, analisando a relação entre os *commits* realizados pelos desenvolvedores e os *issues* registrados em um projeto. Essas questões de pesquisa serviram como base para a avaliação das hipóteses acima descritas.

RQ7. *Quais as características em comum dos usuários que mais utilizam o issue tracking system?*

Esta questão de pesquisa visa compreender as características dos usuários que utilizam a ferramenta de *issue tracking* com maior frequência. Observaremos características como o tempo de utilização do GitHub, o número de seguidores, o número de seguidos e o número de projetos dos colaboradores de projetos registrados no GitHub.

RQ8. *Qual a relação entre as características de um issue o seu encerramento por commit?*

Nesta questão de pesquisa, busca-se entender quais as características comuns aos *issues* que são encerrados por *commits*. As características avaliadas são a utilização de rótulos, se os *issues* encerrados por *commits* possuem um dos

rótulos mais utilizados por *issues* em geral e o tempo de vida dos *issues* encerrados por *commits*.

RQ9. *Qual a característica dos desenvolvedores que encerram issues via commit?*

Com essa questão de pesquisa avaliaremos quais são as características dos desenvolvedores que utilizam a funcionalidade de encerramento de *issues* por *commit*. A única característica dos colaboradores que é avaliada é o número de *commits* realizados no projeto. Não são avaliadas outras características, como o tempo de utilização do GitHub, o número de seguidores, seguidos e repositórios públicos, pois após as análises realizadas na sétima questão de pesquisa, identificamos que essas características são irrelevantes

3.5 Dificuldades Durante a Execução do Estudo

Durante a execução do estudo foram encontradas dificuldades que tiveram que ser superadas. Dentre elas, a principal foi a limitação de 5.000 requisições por hora imposta pela API do GitHub para evitar o congestionamento de seus servidores. Essa limitação impediu que a coleta de dados fosse feita de forma contínua, pois sempre que o limite de requisições era alcançado o servidor do GitHub bloqueava o acesso à sua base. Visando superar essa dificuldade, a rotina de coleta de dados foi construída para executar o menor número possível de requisições. Além disso, foram construídos mecanismos que, sempre que o limite era atingido, forçavam a parada da coleta e a sua retomada após uma hora de espera. A constante necessidade de aguardar a liberação de requisições tornou o processo de coleta de dados muito demorado e, como alternativa para aceleração, foram utilizadas duas máquinas em paralelo para a coleta. O tempo total de coleta foi de aproximadamente dois meses e meio.

Outro obstáculo a ser superado foi a escolha dos projetos que seriam analisados. O artigo original optou por coletar os primeiros 100.000 projetos retornados pela API do GitHub, executando assim uma análise sobre os projetos mais antigos existentes na plataforma naquele momento. A primeira coleta realizada nessa Dissertação manteve a mesma estratégia, coletando os primeiros 200.000 projetos retornados pela API, porém identificamos que essa amostra não representava bem a população de projetos, visto que se concentravam projetos muito antigos. Por esse motivo, buscamos coletar uma

amostra de dados que fosse melhor distribuída, contendo projetos com idades variadas. Para isso, foi executada uma segunda coleta, gerando a amostra utilizada nesse estudo. Antes de executar a coleta de dados, foi necessário coletar o nome de todos os repositórios existentes na plataforma até abril de 2016, o que resultou em uma lista de aproximadamente 35 milhões de projetos. Essa coleta demorou 3 semanas para ser realizada e, após ela foram sorteados aleatoriamente os 220.000 projetos que foram analisados nesse estudo. Nesse processo, foram sorteados projetos cadastrados na plataforma entre 2008 e 2016.

Outra dificuldade enfrentada foi a inexistência de padrões de escrita nos campos de textos coletados, principalmente durante a análise dos *commits* e *issues*. Nos textos foram identificados diversos caracteres de formatação, *tags* de HTML e pontuações que atrapalhavam o processo de análise, pois alteravam a formatação dos arquivos gerados, inserindo células e linhas indevidas nos dados coletados. Visando superar essa dificuldade, foram construídos mecanismos na rotina de coleta de dados para tratamento prévio desses campos de texto, retirando a "poluição" do texto. Além disso, antes da execução das análises foram executados tratamentos manuais dos arquivos gerados, onde era identificada e corrigida qualquer inconsistência nos dados coletados.

Nesse estudo, não pode ser realizada a replicação exata do método utilizado por BISSYANDE *et al.* (2013), pois os detalhes técnicos do estudo realizado por esses autores não foi disponibilizado. Sendo assim, a replicação realizada nesse estudo é conceitual, de acordo com a nomenclatura utilizada por YONG (2012), pois replicamos todos os conceitos, curadoria, questões de pesquisa e hipóteses realizadas no estudo original, porém utilizando métodos construídos especificamente para esse estudo. Por esse motivo, não há garantias que o método utilizado nesse estudo não teve alguma influencia sobre os resultados identificados.

Além dessas dificuldades, durante a coleta dos dados também foram enfrentados problemas técnicos, como a indisponibilidade de máquina para a continuação da coleta e problemas de configuração das ferramentas utilizadas. Esses problemas foram rapidamente abordados e solucionados, possibilitando a continuidade do estudo.

3.6 Considerações Finais

Este capítulo apresentou as ferramentas utilizadas durante esse estudo, detalhando a forma e o motivo de sua utilização. Apresentou também a descrição das entidades e dados coletados para a execução das análises. Foram apresentadas as regras utilizadas para a curadoria dos dados coletados, as hipóteses e questões de pesquisa nas quais esse estudo se baseou para a execução das análises. Por fim, foram descritas as dificuldades enfrentadas durante a execução do estudo e os meios como elas foram superadas.

No próximo capítulo serão apresentados os resultados das análises realizadas durante esse estudo. Nele, cada uma das questões de pesquisa expostas será detalhada, desenvolvida e as suas conclusões serão discutidas e comparadas com os resultados alcançados no estudo original. Nesse capítulo também será observada a manutenção ou não as hipóteses propostas pelos autores do estudo replicador.

4. Análise de Dados

Este capítulo apresenta o desenvolvimento das análises realizadas para responder às questões de pesquisa propostas. Avaliaremos as características dos colaboradores, projetos e *issues* sob diversos aspectos, buscando identificar relações entre essas características e a utilização da ferramenta de *issue tracking* do GitHub. Serão apresentados os resultados encontrados para cada questão de pesquisa, as conclusões observadas e as respostas para as hipóteses apresentadas no capítulo anterior.

Conforme proposta dessa Dissertação, entre a primeira e a sexta questões de pesquisa serão apresentados os resultados da replicação do estudo anterior. A partir da sétima questão de pesquisa será analisado o relacionamento entre os commits e os issues de um projeto, avaliando assim se a funcionalidade de encerramento de issues através de commits disponibilizada pelo GitHub é utilizada com frequência (e em que contextos) e pode ser usada para identificar relações entre o código-fonte e os issues de um projeto.

4.1 RQ1 – Qual a proporção de projetos que recebem *issues* e como projetos que recebem *issues* podem ser diferenciados daqueles que não recebem com base em suas características?

Essa questão de pesquisa investiga a distribuição do uso de *issue tracking* nos projetos da amostra. Em relação à amostra reduzida (76.909 projetos), que são aqueles projetos que possuem a ferramenta de *issue tracking* ativa e possuem alguma linguagem de programação, apenas 11.662 projetos possuem pelo menos um *issue* registrado na plataforma GitHub, o que representa 15% da amostra reduzida. Esse número representa 5% do total de projetos coletados na amostra original. Alguns dos projetos que não possuem *issues* registrados no GitHub podem utilizar uma ferramenta externa para fazer o cadastro de *issues*. Na curadoria dos dados coletados, filtramos todos os projetos que desabilitaram a ferramenta de *issue tracking* disponibilizada pelo GitHub. Porém, não foi possível identificar se algum projeto restante utiliza ferramentas externas para

registro de *issues*. Dessa forma, esse estudo se baseia apenas nos dados existentes no GitHub e assumem o uso da ferramenta de *issue tracking* se ela não estiver desligada.

Com base na amostra reduzida, será investigada a relação entre a presença ou ausência de *issues* com algumas características do projeto, como sua idade, seu tamanho, o tamanho da equipe de desenvolvimento e a popularidade do dono de projeto. Utilizaremos essa amostra reduzida visando apresentar as reais diferenças entre projetos de software que utilizam ou não o *issue tracking*, retirando da amostra aqueles projetos que não utilizam a ferramenta de *issue tracking* disponibilizada pelo GitHub e que não possuem nenhuma linguagem de programação.

4.1.1 Idade do Projeto

A idade do projeto indica o número de semanas decorridas desde a data de criação do projeto no GitHub até o início do mês de maio de 2016. A Figura 1 demonstra a distribuição da idade dos projetos em nossa amostra em formato de *boxplot*. Foi identificado que a média de idade para projetos que possuem *issues* é de 94,6 semanas, com mediana de 71 semanas. Para os projetos que não possuem *issues* a média é de 82,2 semanas, com mediana de 62 semanas. Utilizando o teste de Wilcoxon-Mann-Whitney (WMW), um teste não paramétrico que verifica se existe uma diferença estatisticamente significativa entre as medianas de dois conjuntos de dados, foi identificado que a diferença de idade entre os conjuntos de projetos com e sem *issues* é estatisticamente significativa com 99% de certeza (p-value < 0,01).

Utilizaremos a métrica de tamanho de efeito de Vargha & Delaney para identificar o tamanho de efeito para a comparação entre os projetos com e sem *issues*. Quando esta métrica assume o valor de 50%, ela indica que os dois grupos tem chances iguais de gerar o melhor resultado (neste caso, a maior idade). Valores acima de 50% indicam que o primeiro grupo tem chances maiores de apresentar o melhor resultado (por exemplo, se o valor de 80% for observado, concluímos que o primeiro grupo tem chances de apresentar maior resultado 80% das vezes em que for comparado com o segundo grupo). Utilizando Vargha & Delaney para comparar as idades dos projetos, identificamos um valor de 55% de tamanho de efeito, o que demonstra um pequeno tamanho de efeito, de acordo com COHEN (1992), para a diferença entre a idade dos projetos que contêm *issues* e aqueles que não os contêm.

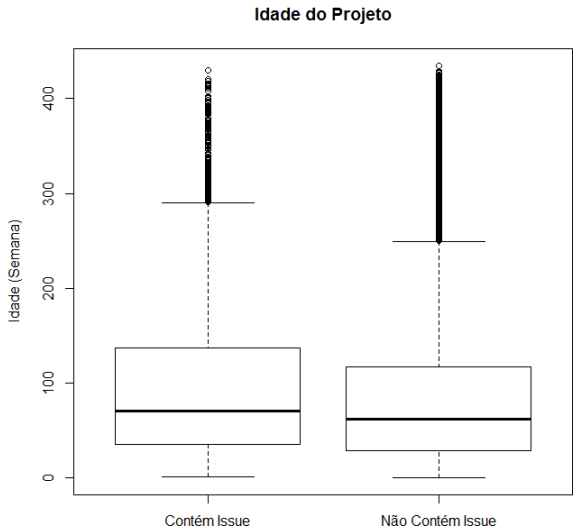

Figura 1 - *Issues* e a idade dos projetos. Observa-se a utilização mais frequente de *issue tracking* em projetos mais antigos.

Tabela 1- *Issue* e a idade dos projetos

Análise	*Média com issues*	*Mediana com issues*	*Média sem issues*	*Mediana sem issues*
BISSYANDE et al. (2013)	81,57	70,04	70,05	59,57
Atual	94,6	71	82,2	62

Esse resultado confirma o encontrado por BISSYANDE *et al.* (2013), que identificou um maior uso de *issue tracking* em projetos mais antigos. Identificamos também que a diferença entre a idade média dos projetos é próxima de 12 semanas, valor próximo ao do artigo original, que apresentava uma diferença de 11 semanas.

4.1.2 Tamanho do Projeto

A Figura 2 apresenta a um *boxplot* com o tamanho em número de linhas de código (LOC) dos projetos da amostra, divididos entre aqueles que possuem *issues* e aqueles que não possuem. Os projetos com *issues* registrados possuem uma média de 24.272,7 linhas de código, com uma mediana de 957. Enquanto isso, os projetos sem *issues* registrados possuem uma média de 22.080,4 linhas de código e uma mediana de 515. O teste WMW confirmou que as duas distribuições são significativamente diferentes (p-value < 0,001). O tamanho de efeito para esse conjunto de dados é de 55%, utilizando Vargha & Delaney. Esse tamanho de efeito é pequeno de acordo com COHEN (1992), indicando que em 55% das vezes os projetos com maior linha de código utilizaram a ferramenta de *issue tracking*.

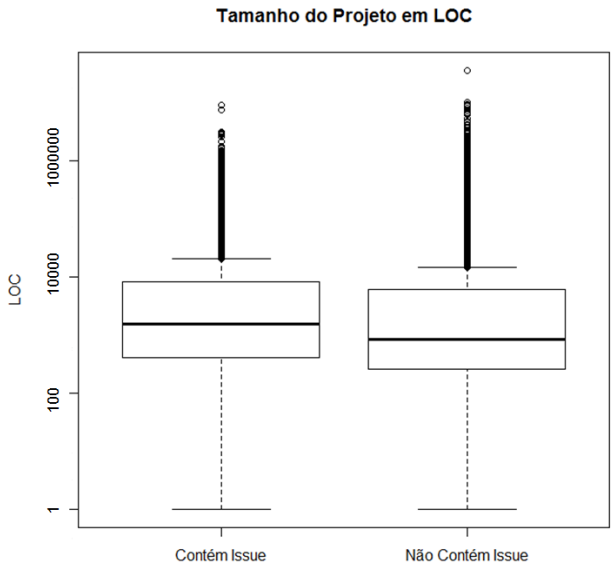

Figura 2 - *Issues* e tamanho em LOC. Observa-se a maior utilização de *issue tracking* em projetos com maior número de LOC.

Tabela 2 - *Issues* e tamanho dos projetos em LOC

Análise	*Média com issues*	*Mediana com issues*	*Média sem issues*	*Mediana sem issues*
BISSYANDE et al. (2013)	NA	1.820	NA	1.027
Atual	24.272,7	957	22080,4	515

O resultado encontrado no estudo atual confirma a conclusão apresentada pelos autores sobre esse tópico no artigo original, identificando que projetos com menor base de código tendem a ter menos *issues* registrados.

4.1.3 Tamanho da Equipe de Desenvolvimento

A Figura 3 apresenta a distribuição dos projetos com relação ao número de desenvolvedores que compõem a equipe de desenvolvimento. Foi encontrada uma média de 12 desenvolvedores e uma mediana de três desenvolvedores para projetos com *issues* registrados, enquanto que para projetos sem *issues* a média é de 4,2 desenvolvedores e a mediana de apenas um desenvolvedor. Utilizando o teste de WMW encontramos diferença significativa entre os dois conjuntos de dados (p-value < 0,001). Utilizando Vargha & Delaney, o tamanho de efeito para esse conjunto de dados é de 81%, grande segundo COHEN (1992), indicando que em 81% das vezes os projetos que contêm mais desenvolvedores na equipe utilizam a ferramenta de *issue tracking*.

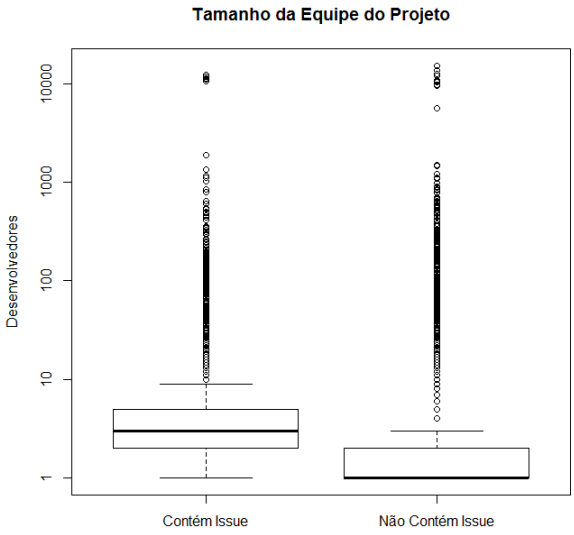

Figura 3 - *Issues* e tamanho da equipe de desenvolvimento. Observa-se a maior utilização de *issue tracking* em projetos que possuem mais desenvolvedores envolvidos.

Tabela 3 - *Issues* e tamanho da equipe de desenvolvimento

Análise	*Média com issues*	*Mediana com issues*	*Média sem issues*	*Mediana sem issues*
BISSYANDE et al. (2013)	49	5	43	2
Atual	12	3	4,2	1

Observamos que a mediana para projetos sem *issues* é de apenas um desenvolvedor, pois 67% dos 65.247 projetos desse conjunto possuem apenas um desenvolvedor na equipe. O percentual de projetos sem *issue* registrado e com dois desenvolvedores é de 17%; com cinco ou mais desenvolvedores, o percentual é de apenas 2,6%. Assim como no artigo original, foi confirmada a relação entre o tamanho da equipe e a adoção de *issue tracking* em um projeto, indicando que projetos com uma equipe maior têm maior probabilidade de utilizar a ferramenta de *issue tracking*.

4.1.4 Popularidade do Líder do Projeto

A Figura 4 mostra a distribuição dos projetos com relação à popularidade do seu dono. Como medida de popularidade, utilizamos o número de usuários que segue o dono desse projeto na plataforma do GitHub e investigamos se essa popularidade tem alguma relação com o uso de *issue tracking*.

A média de seguidores de donos de projetos com *issues* é de 44,7 usuários, com uma mediana de apenas dois usuários. Para projetos sem *issues* registrados, a média de seguidores é 15 usuários, com mediana de um usuário. Novamente, utilizando o teste

WMW identificamos uma distribuição significativamente diferente (p-value < 0,001) entre o número de seguidores dos donos de projetos que contêm e que não contêm *issues*. Utilizando Vargha & Delaney, identificamos o tamanho de efeito de 54%, pequeno segundo COHEN (1992), mostrando que em 54% das vezes os projetos com donos mais populares utilizaram a ferramenta de *issue tracking*.

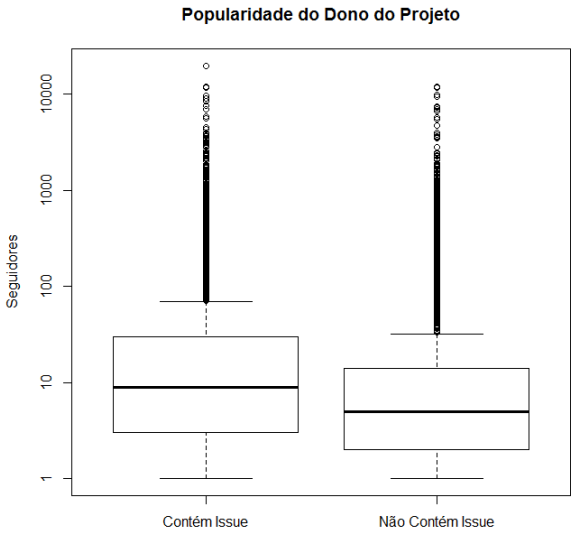

Figura 4 - *Issues* e a popularidade do dono do projeto. Observa-se a maior utilização de *issue tracking* em projetos cujos donos possuem mais seguidores.

Tabela 4 - *Issues* e a popularidade do dono dos projetos

Análise	*Média com issues*	*Mediana com issues*	*Média sem issues*	*Mediana sem issues*
BISSYANDE et al. (2013)	130	15	21	2
Atual	44,7	2	15	1

Como no tópico anterior, a mediana de seguidores para projetos sem *issues* é de apenas um usuário. Isso se deve ao fato de que 54% dos donos de projetos desse conjunto possuem apenas um ou nenhum seguidor (42% com nenhum seguidor e 12% com apenas um seguidor). Donos de projetos com cinco ou mais seguidores representam 29% do conjunto, enquanto donos de projetos com dez ou mais seguidores representam 19% e donos com 20 ou mais seguidores representam 11%.

Assim como o artigo original, foi confirmado que projetos com donos menos populares possuem uma chance menor de receber *issues* registrados pelos colaboradores do que projetos com donos mais populares. Conforme apontado por BISSYANDE *et al.* (2013), "o dono do projeto deve ser conhecido por outros usuários do GitHub, o que pode influenciar no registro de issues através da ferramenta de issue tracking".

Todos os tópicos abordados nessa replicação apresentaram comportamento semelhante ao observado no artigo original. Sendo assim, podemos assumir que "projetos com *issues* reportados tendem a ser mais velhos, ter mais linhas de código, maior número de desenvolvedores envolvidos e donos mais populares que projetos sem *issues*" de acordo com BISSYANDE *et al.* (2013).

Semelhante aos resultados encontrados no estudo original, as conclusões encontradas durante a replicação invalidam a H1, visto que apenas 11% dos projetos *open-source* estudados possuíam algum *issue* registrado. Assim, podemos concluir que a utilização de *issue tracking* em projetos *open-source* é bastante baixa.

4.2 RQ2 – Quantos *issues* são mapeados em projetos onde *issue tracking* é utilizado?

O estudo atual foi realizado sobre uma base de 226.340 *issues* reportados em 11.662 projetos (15% dos projetos presentes na amostra reduzida de 76.909). No artigo original, os autores coletaram 803.840 *issues* em 20.041 projetos (próximo de 30% de uma base de 100.000 projetos).

Nessa questão de pesquisa será investigada a distribuição de *issues* por projetos que possuem *issues* registrados. Na Tabela 5 observamos que 93,1% dos projetos possuem menos de 50 *issues* reportados, contra 86% apresentados no artigo original. Também observamos que não foi encontrado nenhum registro de projeto com mais de 5.000 *issues* reportados, contra apenas 4 registros no artigo original.

Tabela 5 - Número de *issues* por projeto

# Issue	*# Projetos*	*% do total*	*# Projetos BISSYANDE et al.* (2013)	*% do total BISSYANDE et al.* (2013).
1-9	8.803	75,48%	11.602	57,89%
10-49	2.054	17,61%	5.526	27,57%
50-99	355	3,04%	1.411	7,04%
100-249	272	2,33%	976	4,87%
250-499	108	0,93%	290	1,44%
500-999	58	0,50%	163	0,81%
1000-4999	12	0,10%	69	0,34%
5000-9999	0	0,00%	2	0,01%
>= 10000	0	0,00%	2	0,01%

Investigamos também o número de *issues* a cada 1.000 linhas de código (kLOC) para os projetos que possuem *issues* registrados. Foi encontrada uma média de 13,5 *issues* por kLOC, com uma mediana de 1,54 *issues* por kLOC. No artigo original, foi encontrada uma mediana de 6,32 *issues* por kLOC. Essas medianas apresentam valores quebrados, visto que é a relação de *issues* por kLOC, o que na maior parte dos casos é um valor com decimal. A Figura 5 apresenta a distribuição de *issues* por kLOC, na forma de um *boxplot* em escala logarítmica.

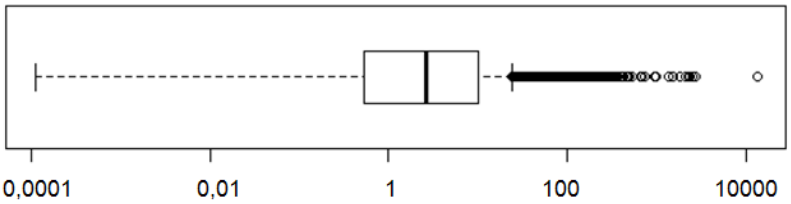

Figura 5 - *Issues* por KLOC

Na Figura 6 é apresentado um scatterplot em escala logarítmica do número de *issues* contra o número de LOCs do projeto. Calculando a correlação de Spearman, que mede a dependência entre duas variáveis e deve ser aplicada quando os dados não possuem distribuição próxima à curva normal, encontramos o valor de 0,199, indicando que o número de LOC e o de *issues* possuem uma relação fraca, segundo COHEN (1992). Esse resultado é ligeiramente inferior ao apresentado no artigo original, que encontrou uma correlação de 0,341.

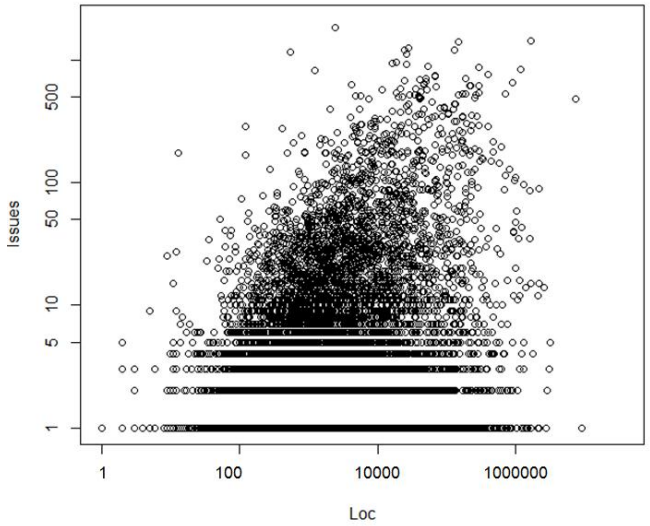

Figura 6 - *Issues* por LOCs. Observa-se uma relação fraca entre o número de *issues* e o de linhas de código em um projeto.

A Figura 7 apresenta a distribuição do número de *issues* entre os projetos que utilizam alguma das dez linguagens de programação com maior número de *issues* registrados em nossa amostra. Dos 226.340 *issues* presentes na amostra, 194.081 estão distribuídos em projetos que utilizam uma das linguagens citadas na Figura 7 (60.902 projetos, considerando os projetos com e sem *issues*). Assim como no artigo de BISSYANDE *et al.* (2013), observamos que várias das linguagens listadas são focadas para desenvolvimento web, como Ruby, Python e PHP. Também observamos que a maior média apresentada é para a linguagem C, com 5,27 *issues* por projeto. A Tabela 6 apresenta a média de *issues* por projeto para cada uma dessas linguagens.

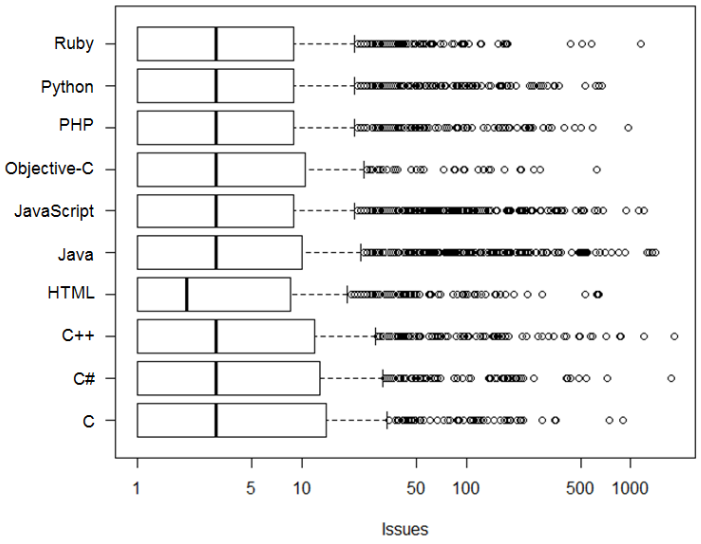

Figura 7 - Número de *issues* em projetos com as linguagens de programação mais utilizadas. Observa-se uma quantidade de *issues* por projeto muito próxima para as dez linguagens de programação selecionadas, com pequena superioridade para projetos na linguagem C.

Tabela 6 - Média de *issues* por projeto

Linguagem	*Projetos*	*#issues*	*issues / projeto*
C	2.998	15.807	5,27
C++	3.565	18.536	5,20
C#	3.004	13.673	4,55
Java	12.111	43.211	3,57
Python	6.668	21.440	3,22
Java Script	14.023	39.358	2,81
Objective-C	2.053	5.647	2,75
PHP	5.147	14.112	2,74
HTML	4.960	11.214	2,26
Ruby	6.373	11.083	1,74

Como BISSYANDE *et al.* (2013), concluímos que existe uma forte utilização da ferramenta de *issue tracking* em projetos desenvolvidos para web. Também se nota a presença de uma linguagem de desenvolvimento para *mobile* entre as 10 linguagens que mais receberam registro de *issues*, o Objective-C.

Como conclusão dessa questão de pesquisa podemos citar que a maioria dos projetos têm poucos *issues* identificados ou nenhum *issue*. Apenas 6,9% dos projetos possuem mais de 50 *issues* dentro da amostra utilizada. Observamos uma relação fraca, segundo COHEN (1992), entre o número de LOC e o número de *issues*, ligeiramente menor que a identificada no artigo de BISSYANDE *et al.* (2013), que observou uma relação moderada. Em relação à quantidade de *issues* reportados, as conclusões do artigo atual são semelhantes às conclusões encontradas originalmente.

4.3 RQ3 – Qual a popularidade da utilização de rótulos? Quais são os principais rótulos utilizados em projetos de desenvolvimento de software?

Nessa questão de pesquisa avaliaremos a utilização de rótulos para categorizar issues no GitHub. Os rótulos utilizados podem ser os já disponibilizados pela plataforma ou novos rótulos criados pelos próprios usuários de um projeto. Dessa forma, os desenvolvedores podem criar rótulos específicos para seus projetos para filtrar e acompanhar os issues de acordo com suas próprias categorias. Entretanto, a criação de rótulos pelos usuários tem seu preço, visto que reporters podem categorizar issues com erros de digitação ou usando termos específicos para a equipe do projeto, segundo BISSYANDE et al. (2013). Devido à flexibilidade proporcionada pelo GitHub para criação de rótulos, uma grande variedade de rótulos pode possuir o mesmo significado.

Nos 226.340 *issues* identificados na amostra reduzida, foram encontrados 304.033 rótulos (380.994 rótulos considerando todos os *issues* identificados na amostra inicial), sendo 2.596 rótulos distintos, distribuídos em 118.596 *issues*. Dentre esses rótulos, vários possuíam significados semelhantes e foram agrupados em 301 grupos de rótulos. A Tabela 7 apresenta a relação dos dez grupos de rótulos mais frequentes na amostra estudada e no artigo original. Nesse estudo, os três rótulos mais frequentemente utilizados no artigo original (*bug*, *feature* e *enhacement*) também foram identificados entre os dez grupos de rótulos mais utilizados, porém em outras posições. Esses rótulos representam 12% dos rótulos coletados, contra 34% no artigo original. Os três grupos de

rótulos mais utilizados encontrados nesse estudo representam 72% do total de rótulos utilizados para marcação de *issues*. Os rótulos presentes nestes três grupos foram encontrados em 82.782 *issues* distintos, sendo que 60.874 *issues* possuem os 3 rótulos ao mesmo tempo.

Tabela 7 - Os dois rótulos mais frequentemente utilizados no GitHub

Rótulos Trabalho Atual	*Quantidade*	*%*	*Rótulos BISSYANDE et al.* (2013)	*Quantidade*	*%*
AUTO-MIGRATED	81.703	27%	BUG	40.112	18%
PRIORITY	75.447	25%	FEATURE	22.477	10%
DEFECT	61.631	20%	ENHANCEMENT	11.584	5%
ENHANCEMENT	23.368	8%	WIN 7	9.736	4%
BUG	9.405	3%	IE8	7.626	3%
MILESTONE	6.643	2%	CHROME	6.817	3%
STATUS	3.645	1%	OTHER BROWSER	6.667	3%
COMPONENT	3.170	1%	FIRE FOX	5.669	3%
TYPE	2.943	1%	FEATURE REQUEST	5.594	2%
FEATURE	2.655	1%	WRONG-UNCLEAR	5.464	2%

A média de rótulos identificados por *issue* é de 2,6, com uma mediana de três rótulos. O número máximo de rótulos em um *issue* é 10. A Figura 8 apresenta um histograma do número de rótulos por *issues*. A Tabela 8 apresenta os principais rótulos que foram agrupados como *bug*, *feature*, *priority*.

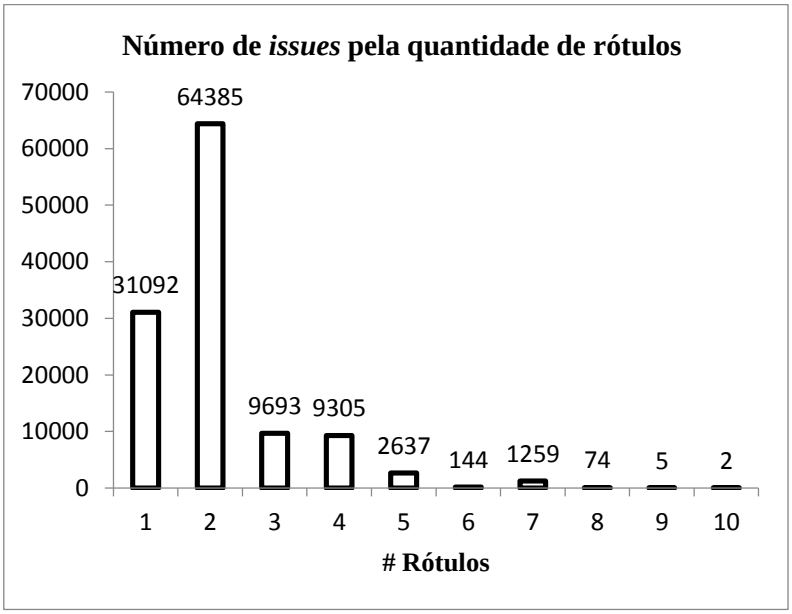

Figura 8 - Quantidade de *issues* com um determinado número de rótulos. Observa-se que 80% dos *issues* com rótulos possuem apenas um ou dois rótulos.

Tabela 8 - Principais grupos de rótulos

GRUPO	*# Rótulos*	*Principais Rótulos*
BUG	52	*bug; bugs; Bug; bug report; bug!!; bud-dev; bug-fix; bugfix; bug-report; confirmed_bug*
FEATURE	38	*features; feature; Feature; feature Enhancement; feature request; Feature Request; Feature!; feature_request; feature-request*
AUTO-MIGRATED	1	*auto-migrated*
PRIORITY	66	*Priority-Medium; Priority-Low; Priority-High; High prioirty; Priority-Critical; Low-pri; Priority-Triage; priority: high*
DEFECT	5	*Type-Defect; 2:Defect; defect; Possible defect; T: defect;*

O artigo original criou um mecanismo baseado na distância de Levenshtein (Levenshtein, 1966) para acrescentar outros rótulos dentro dos grupos de *bug* e *feature*, totalizando ao final do processo 18 e 91 rótulos para *bug e features*, respectivamente. Esse mecanismo não foi replicado para o estudo atual, onde o agrupamento foi feito de forma manual. Foram identificados manualmente todos os rótulos que possuem semelhança textual. Para isso, foi utilizada a ferramenta Excel e a funcionalidade de filtros. Após filtrar rótulos com semelhança textual, foi analisado se eles eram utilizados com o mesmo objetivo: aqueles que não se igualavam ao contexto dos demais rótulos selecionados para formar um grupo eram eliminados dos filtros. Depois de identificar os rótulos semelhantes, eles foram agrupados de acordo com seu contexto e o processo de filtragem, análise e agrupamento foi repetido. Buscamos não criar grupos de rótulos que possuíssem rótulos muito diferentes, mesmo que contextualmente fossem parecidos. Por esse motivo, foram criados grupos distintos para *defect* e *bug.* No estudo atual, o grupo que apresentou maior variedade de rótulos distintos foi o grupo *milestone*, com 274 rótulos, em sua maioria distintos devido à presença do número do *milestone* em questão.

Por fim, identificamos que 54% dos *issues* da amostra possuem rótulos, ou seja, os 304.033 rótulos identificados estão concentrados em 119.008 *issues*, demonstrando que nessa amostra é comum utilizar mais de um rótulo por *issue*. No artigo original, foram encontrados rótulos em apenas 30% da base de *issues* analisados.

4.4 RQ4 – Quem cadastra *issues*? Quantas destas pessoas fazem parte da equipe de desenvolvimento?

O objetivo dessa questão de pesquisa é discutir se a utilização da ferramenta de *issue tracking* é feita por colaboradores de um projeto que não são desenvolvedores ou se um colaborador que registra *issues* também participa do desenvolvimento. Para isso, em cada projeto classificamos os colaboradores como desenvolvedores, quando contribuem com *commits*, ou como *reporters*, quando apenas registram *issues*. Além disso, para cada projeto identificamos o número de colaboradores que executaram atividades de desenvolvimento (*commits*) e também fizeram o registro de *issues*.

Identificamos um total de 110.946 desenvolvedores nos projetos que possuem *issues* registrados (581.856 no artigo original), com uma média de 9,5 desenvolvedores por projeto e uma mediana de dois desenvolvedores. Já o número de *reporters* identificados na amostra é de 44.522 colaboradores (239.629 no artigo original), com uma média de 3,8 por projeto e uma mediana de um colaborador. Esses números consideram os colaboradores distintos por projeto e não para toda a amostra, ou seja, caso o mesmo colaborador tenha executado atividades de desenvolvimento em mais de um projeto, ele será contabilizado uma vez para cada projeto. O número de *reporters* segue o mesmo comportamento, porém no mesmo projeto esse colaborador não será contado mais de uma vez, independente da quantidade de ações realizadas.

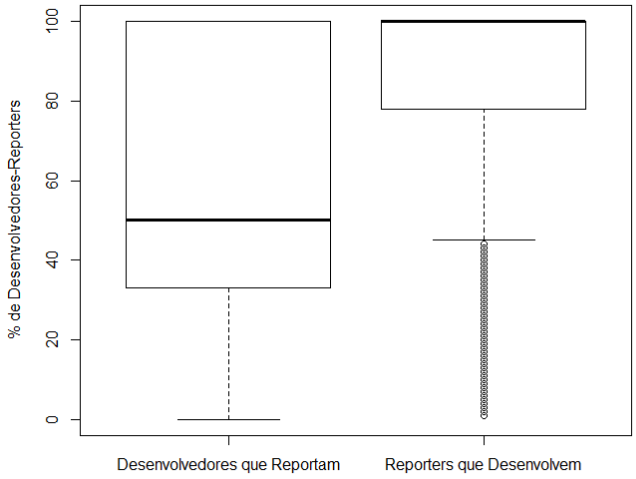

Figura 9 - Proporção de desenvolvedores entre os *issues reporters* e vice-versa. Observa-se que não existem grupos distintos de desenvolvedores e *reporters* na maior parte dos projetos.

Foi identificado que na média 38% dos desenvolvedores também atuam como *issue reporter*, com uma mediana de 33% (31% no artigo original). Por sua vez, em média 53% dos *reporters* também participam de alguma atividade de desenvolvimento, com uma mediana de 50% (42% no artigo original). A Figura 9 mostra a distribuição de desenvolvedores e *reporters* nos projetos. Assim como apresentado por BISSYANDE *et al.* (2013), o *boxplot* mostra que muitos desenvolvedores não registram *issues* em seus projetos, enquanto grande parte dos *reporters* também contribui para o código-fonte.

Essa questão de pesquisa está relacionada à hipótese H2, que é negada porque não foram observados grupos distintos de desenvolvimento e registro de *issues*, uma vez que um grande número de desenvolvedores contribui como *reporters* e um grande número de *reporters* também contribuem para a evolução do código-fonte do projeto.

4.5 RQ5 – Qual a relação entre a utilização de *issue tracking* e o sucesso do projeto?

Para essa questão serão utilizadas as métricas de forks e watchers de um projeto para indicar o seu sucesso e o nível de interesse despertado pelo projeto na comunidade de desenvolvimento que utiliza o GitHub. Com essa informação, será analisada a relação entre o sucesso do projeto e a utilização da ferramenta de issue tracking. Utilizaremos a amostra reduzida de projetos, contendo todos os projetos que possuem a ferramenta de issue tracking ativa e que possuem alguma linguagem de programação.

4.5.1 *Watchers*

Como primeira análise, observamos o número de *watchers* que "seguem" um projeto no GitHub para determinar a sua popularidade. A Figura 10 apresenta o *scatterplot* da relação entre o número de *issues* e o número de *watchers* de um projeto. A correlação de Spearman para essa relação é de 0,311, o que segundo COHEN (1992) representa uma relação moderada entre os dados. Esse resultado é inferior ao encontrado no artigo original, que observou o valor de 0,628, considerado forte segundo COHEN (1992). Concluímos assim, que existe uma relação moderada entre a quantidade de *watchers* e a quantidade de *issues* em um projeto, confirmando o resultado identificado por BISSYANDE *et al.* (2013), porém com um tamanho de efeito menor.

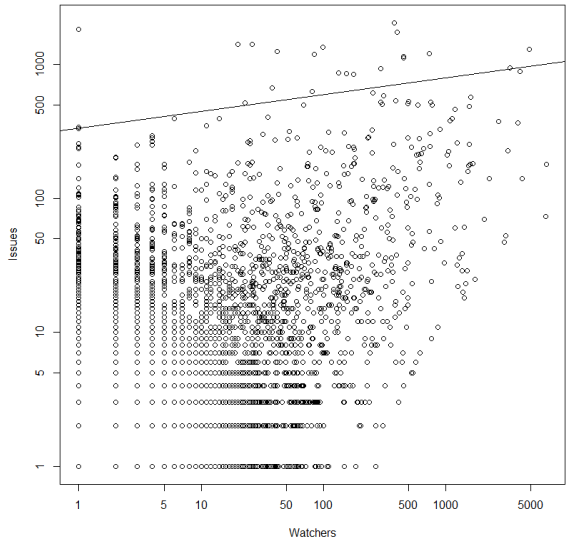

Figura 10 - A relação entre o número de *watchers* e número de *issues*. Observa-se uma correlação moderada entre o número de *issues* e *watchers* nos projetos selecionados.

Dando sequência à análise, observamos a relação entre o número de *watchers* e o número de usuários que reportam *issues* em um projeto. A Figura 11 apresenta o *scatterplot* para a relação entre as duas distribuições. A correlação de Spearman indica o valor de 0,323 para essa relação, moderada segundo COHEN (1992). Esse resultado é inferior ao encontrado no artigo original, onde a correlação observada foi de 0,789, forte segundo COHEN (1992). Concluímos que existe uma relação moderada entre a quantidade de *watchers* e a quantidade de pessoas que registram *issues* em um projeto, confirmando o resultado identificado por BISSYANDE *et al.* (2013), também com um tamanho de efeito menor.

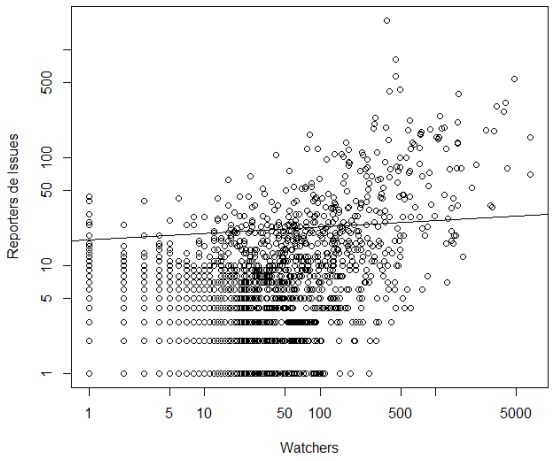

Figura 11 - A relação entre o número de *watchers* e número de *reporters*. Observa-se uma correlação moderada entre o número de *reporters* e *watchers* nos projetos selecionados.

4.5.2 *Forks*

Nesse tópico será analisada a relação entre o número de *issues* registrados e o número de *forks* de um projeto. Segundo BISSYANDE *et al.* (2013), o número de *forks* de um projeto é um indicador do envolvimento de desenvolvedores interessados naquele código que não pertencem ao time chave de um projeto, pois *forks* são ramificações de um código-fonte já existente, onde são realizadas alterações no código sem que o projeto origem daquele *fork* sofra alterações. São muito utilizados para a criação de versões independentes de um código-fonte base, por outros desenvolvedores que podem não estar envolvidos no desenvolvimento do projeto origem daquele *fork.*

A Figura 12 é o *scatterplot* da relação entre o número de *issues* e o número de *forks* de um projeto. Utilizando a correlação de Spearman encontramos o valor de 0,426, indicando uma relação moderada segundo COHEN (1992), entre *issues* e *forks*. Esse resultado é inferior ao valor de 0,669 encontrado no artigo original, que indicava uma relação forte entre o número de *issues* e de *forks*. Assim, podemos concluir que existe uma relação moderada entre o número de *forks* e o número de *issues* em um projeto, como descrito por BISSYANDE *et al.* (2013), porém com um tamanho de efeito menor.

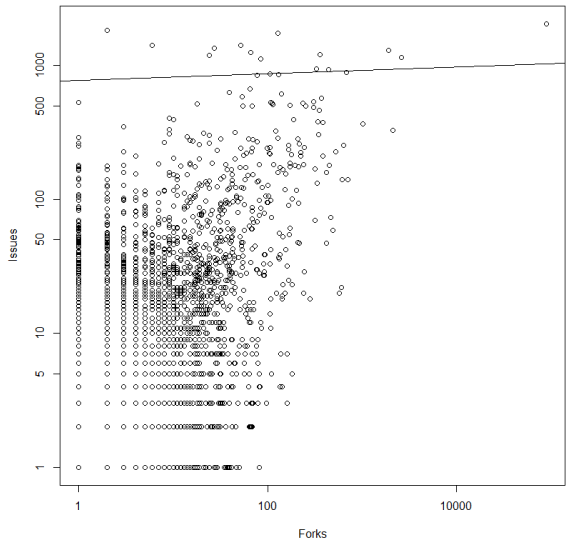

Figura 12 - A relação entre o número de *forks* e número de *issues.* Observa-se uma correlação moderada entre o número de *issues* e *forks* nos projetos selecionados.

Finalmente, a Figura 13 apresenta o scatterplot da relação entre o número de *reporters* e o número de *forks*. Utilizando a correlação de Spearman encontramos uma relação de 0,442, moderada segundo COHEN (1992). Esse resultado é inferior ao

encontrado no artigo original, onde foi encontrado o valor de 0,829, representando uma correlação forte. Podemos concluir que existe uma relação entre o número de *forks* e o número de colaboradores que registram *issues* em um projeto, assim como descrito por BISSYANDE *et al.* (2013), porém com um tamanho de efeito menor.

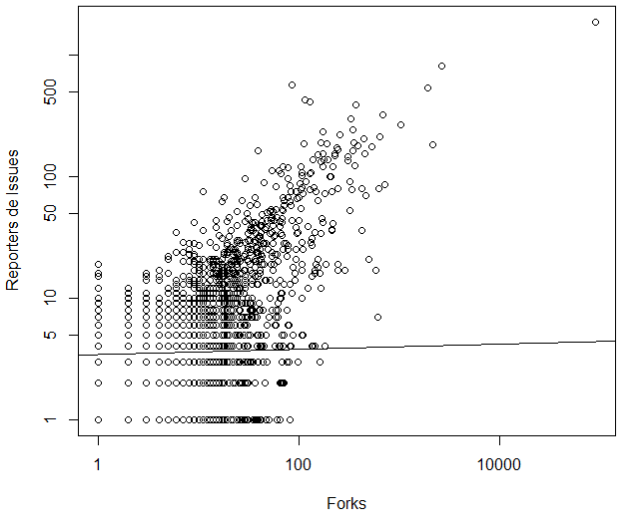

Figura 13 - A relação entre o número de *forks* e número de *reporters.* Observa-se uma correlação moderada entre o número de *reporters* e *forks* nos projetos selecionados.

Como conclusões dessa questão de pesquisa podemos citar que o número de *watchers* e *forks* de um projeto possuem relação com a utilização da ferramenta de *issue tracking* e com o número de usuários que utilizam essa ferramenta em um projeto. Assim como no artigo original, esses resultados apontam para a sustentação da hipótese H3, indicando que um ambiente colaborativo e distribuído aumenta o número de registros de *issues* por parte da equipe do projeto.

BISSYANDE *et al*. (2013) observaram a que o principal catalisador para essa relação pode ser o conjunto de funcionalidades semelhantes a uma rede social disponibilizadas pelo GitHub para aqueles usuários de seguem algum projeto ou outro usuário. Essas funcionalidades permitem que usuários possuam maior visibilidade dos eventos que ocorrem em projetos do seu interesse ou que são realizados por outros usuários que eles acompanhem, aumentando o interesse desses usuários pela colaboração.

4.6 RQ6 – O tamanho da comunidade de usuários afeta o tempo de correção de um *issue*?

Nessa questão de pesquisa serão investigados os aspectos que podem possuir relação com a velocidade com que os *issues* são corrigidos nos projetos presentes na amostra reduzida. O tempo de correção de um *issue* é a diferença em dias entre a data de abertura e encerramento. A Figura 14 apresenta o *scatterplot* para a relação entre o número de *reporters* de *issues* e a média do tempo de correção de *issues* em um projeto. Esse gráfico aparentemente não apresenta relação linear entre as duas distribuições, assim como no artigo original.

Ao calcularmos a correlação de Spearman entre as duas séries de dados encontramos o valor de 0,062, fraco segundo COHEN (1992). Esse resultado é próximo ao encontrado no artigo original (0,161), também fraco segundo COHEN (1992). Sendo assim, podemos concluir que o número de *reporters* possui uma relação fraca com o tempo para correção dos *issues* encontrados na amostra. Assim como no artigo original, a hipótese H4 não foi confirmada, levando em consideração que o resultado encontrado não demonstra forte correlação entre as distribuições.

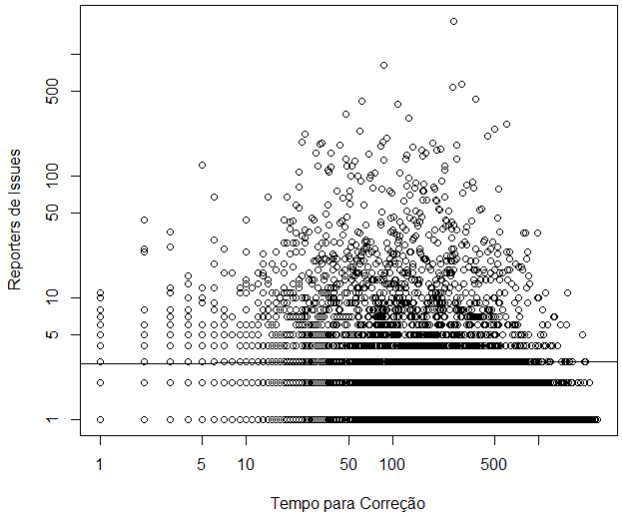

Figura 14 - A relação entre o número de *reporters* e o tempo para correção de *issues.* Observa-se a inexistência de uma relação forte entre o tempo de encerramento de um *issue* e o número de *reporters.*

Nesse ponto é encerrada a replicação do artigo de BISSYANDE *et al*. (2013). Chegamos à conclusão que os resultados apresentados no artigo de 2013 são semelhantes aos resultados encontrados nesse estudo e, consequentemente, continuam

válidos como uma avaliação do uso de ferramentas de *issue tracking*. Os próximos tópicos dessa questão de pesquisa não foram avaliados por BISSYANDE *et al*. (2013) e foram acrescentados nesse estudo visando identificar as características de *issues* que são encerrados rapidamente. Neste sentido, buscamos identificar as características existentes em um *issue* que podem influenciar no seu tempo de encerramento.

A primeira característica avaliada é se a existência de um rótulo influencia no tempo de encerramento de um *issue.* A média do tempo de encerramento para *issues* que possuem rótulos é de 22,52 dias, com mediana de zero dias, visto que 73% desses *issues* foram encerrados com menos de um dia. Para *issues* que não possuem rótulos, a média do tempo de encerramento é de 25,17 dias, com uma mediana de zero dias (62% desses *issues* foram encerrados em menos de um dia). Utilizando o teste WMW identificamos medianas significativamente diferentes para o tempo de correção entre *issues* que possuem rótulos e aqueles que não possuem (p-value < 0,001). O tamanho de efeito de Vargha & Delaney para esse conjunto de dados é de 56%, pequeno segundo COHEN (1992). Esse tamanho de efeito mostra que em 56% das vezes o tempo de correção será menor para os *issues* que possuem *rótulos.*

A Figura 15 apresenta um gráfico dessa distribuição utilizando a escala logarítmica, onde as ocorrências do número zero (ou seja, tempo de correção zero) foram retiradas dos intervalos. A escala logarítmica foi utilizada para tornar o gráfico compreensível, devido ao tamanho da amostra e aos *outliers*. Sem a escala logarítmica, os gráficos apresentariam um intervalo de valores muito grande em seus eixos e os dados mais relevantes (IQR) ficariam concentrados em pontos muito próximos, dificultado a compreensão da informação.

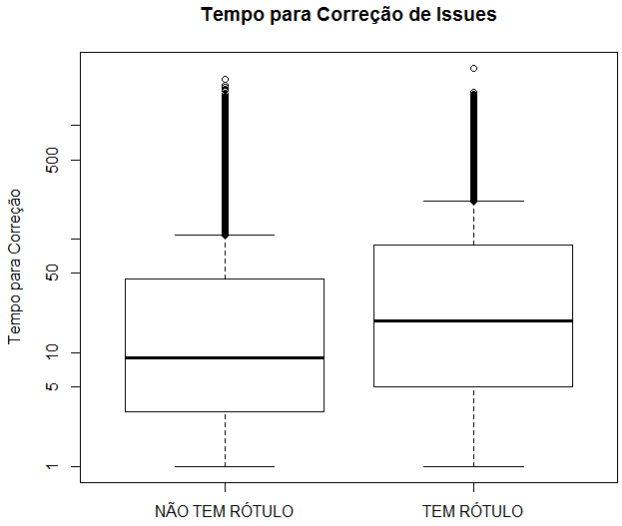

Figura 15 - Tempo de correção de *issues* e a existência de rótulos. Observa-se menor tempo para o encerramento para *issues* que possuem rótulos.

Investigamos também a relação entre o tempo necessário para o encerramento de um *issue* e a quantidade de comentários por ele recebidos. Calculando a correlação de Spearman, encontramos o valor de 0,065, que representa uma correlação praticamente nula (COHEN, 1992) entre o número de comentários de um *issue* e o tempo para sua correção. Sendo assim, não encontramos evidências de que o número de comentários influencia o tempo de correção de um *issue*. A Figura 16 apresenta o gráfico que descreve essa relação.

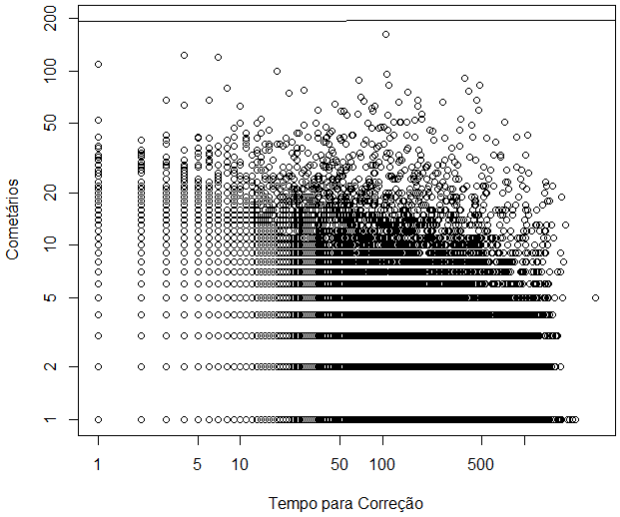

Figura 16 - A relação entre o tempo de correção para *issues* e o número de comentários. Observa-se que não existe correção entre o tempo de correção e o número de comentários recebidos por um *issue*.

Outro tópico analisado foi a relação entre o tempo de correção de um *issue* e se o mesmo possui algum dos rótulos presentes nos dez grupos de rótulos mais utilizados. A média do tempo de correção para os *issues* que possuem algum rótulo destes grupos é de 17,3 dias, com mediana de zero dia (80% desses *issues* foram encerrados em menos de um dia). Já para os *issues* que não possuem rótulos ou possuem rótulos fora dos dez grupos mais utilizados, a média do tempo de correção é de 28,80 dias, com mediana de zero dia (60% desses *issues* foram encerrados com menos de um dia). Utilizando o teste WMW, identificamos uma distribuição significativamente diferente para essas amostras (p-value < 0,001). O tamanho de efeito de Vargha & Delaney para esse conjunto de dados é de 60%, pequeno segundo COHEN (1992). Verificamos assim que em 60% das vezes os *issues* que possuem algum dos rótulos mais utilizados terão um tempo de correção menor do que aqueles que não possuem um desses rótulos. A Figura 17 apresenta um gráfico dessa distribuição utilizando escala logarítmica, onde as ocorrências do número zero foram retiradas dos intervalos.

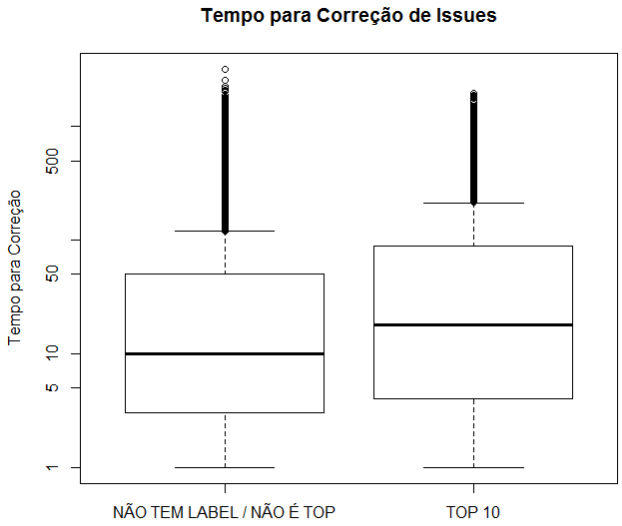

Figura 17 - Tempo de correção dos *issues* que possuem ou não os rótulos mais utilizados. Observa-se menor tempo para o encerramento em *issues* que possuem rótulos que participam dos 10 grupos de rótulos mais utilizados.

Nessa questão de pesquisa concluímos que *issues* que possuem algum rótulo tendem a ser corrigidos mais rapidamente do que aqueles que não possuem, com uma diferença de aproximadamente três dias em média. *Issues* que possuem rótulos presentes nos dez grupos de rótulos mais utilizados também possuem essa tendência, porém com uma diferença de nove dias em média. Verificamos também que não há uma relação entre a quantidade de comentários recebidos por um *issue* e o seu tempo de correção.

4.7 RQ7 – Quais as características em comum dos usuários que mais utilizam o *issue tracking system*?

Nessa questão de pesquisa serão investigadas as características presentes em um colaborador e a sua relação com a utilização da ferramenta de *issue tracking* da plataforma GitHub para fins de registro de novos *issues*. Para isso, foram coletadas informações sobre todos os colaboradores que participaram como desenvolvedores ou *reporters* dos projetos presentes na amostra reduzida. Desse total, retiramos os colaboradores que não puderam ter suas informações coletadas devido ao cancelamento de suas contas ou que possuem dados privados, restando 40.885 colaboradores na amostra que será analisada. Desse número, 28.177 registraram pelo menos um *issue* nos projetos da amostra.

4.7.1 Tempo de Criação da Conta

A primeira relação analisada é entre o número de *issues* registrados por um colaborador e o tempo, em dias, desde a criação da sua conta no GitHub. A Figura 18 apresenta o *scatterplot* desses dois conjuntos de dados, não apresentando nenhuma relação evidente entre o número de *issues* e o tempo desde a criação da conta. Calculando a correlação de Spearman encontramos um valor negativo de 0,101, o que representa uma correlação fraca (COHEN, 1992). Sendo assim, observamos que não existe uma relação evidente entre o número de *issues* registrados por um usuário e o seu tempo de utilização da plataforma GitHub.

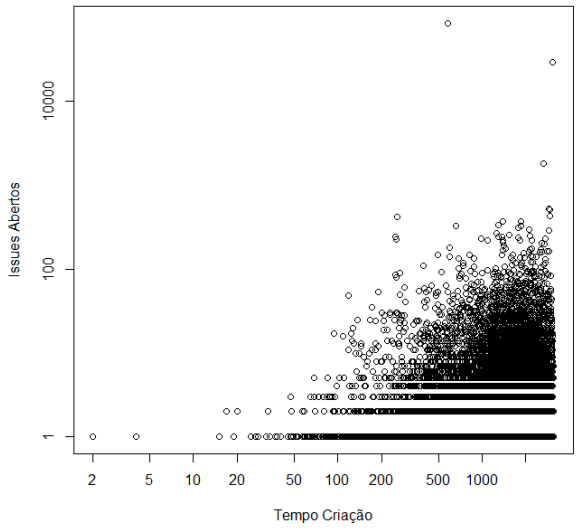

Figura 18 - Relação entre *issues* registrados e o tempo desde a criação da conta do usuário. Observa-se uma correlação bastante fraca entre o tempo da criação da contra de um usuário e número de *issues* registrados por ele.

Foi avaliado também se existe uma diferença estatisticamente significativa entre o tempo desde a criação da conta no GitHub para os colaboradores que registraram e não registraram *issues*. Identificamos que a média de tempo de criação para os colaboradores que registraram *issues* é de 1597,7 dias, com mediana de 1601 dias. Para os colaboradores que não registraram *issues* a média é de 1827,7 dias, com mediana de 1836 dias. Utilizando o teste WMW identificamos uma distribuição significativamente diferente (p-value < 0,001) entre o grupo de colaboradores que registrou ou não registrou *issues* com tamanho de efeito pequeno (59%), indicando que 59% dos colaboradores que não registraram *issues* são mais antigos. A Figura 19 apresenta um *boxplot* dessa distribuição.

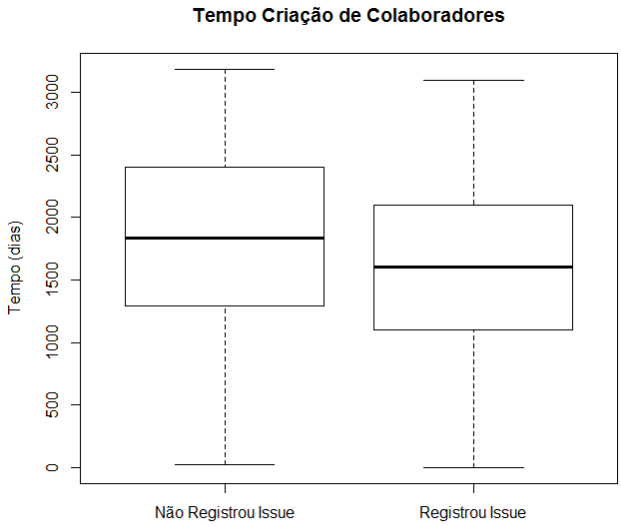

Figura 19 - Tempo de criação das contas dos colaboradores que registraram *issues* e que não registraram. Observa-se uma que colaboradores que registram *issues* tendem a ter menos tempo de criação de conta.

4.7.2 Número de *Commits* Registrados

Nesse tópico foi avaliada a relação entre o registro de *issues* e o registro de *commits* pelos colaboradores presentes na amostra. A Figura 20 apresenta um *scatterplot* desses dois conjuntos de dados. Calculando a correlação de Spearman identificamos um valor negativo de 0,185, indicando que existe uma correlação pequena e inversa entre o número de *commits* e *issues* registrados.

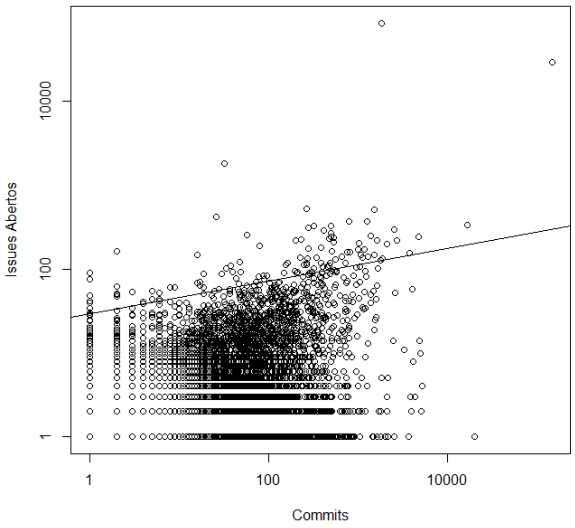

Figura 20 - Relação entre o número de *issues* e *commits* registrados por um colaborador. Observa-se uma correlação fraca entre o número de *commits* e *issues* registrados por um colaborador.

Observamos que colaboradores que registraram *issues* realizaram em média 30,83 *commits*, com mediana de zero (56% dos colaboradores não registraram nenhum *commit* nos projetos da amostra). Já para aqueles colaboradores que não registraram *issues* a média é de 99,6, com mediana de cinco *commits*. Utilizando o teste WMW identificamos uma distribuição significativamente diferente (p-value < 0,001) entre os grupos de colaboradores que registraram ou não registraram *issues*, com tamanho de efeito moderado de 74%, sendo este o percentual de vezes que um colaborador que registrou *issues* também terá registrado mais *commits*. A Figura 21 apresenta *boxplots* do número de *commits* realizados por colaboradores agrupados conforme o registro de *issues*. Os *boxplots* utilizam a escala logarítmica e foram eliminados colaboradores com zero *commits* (56% dos colaboradores que registraram *issues*).

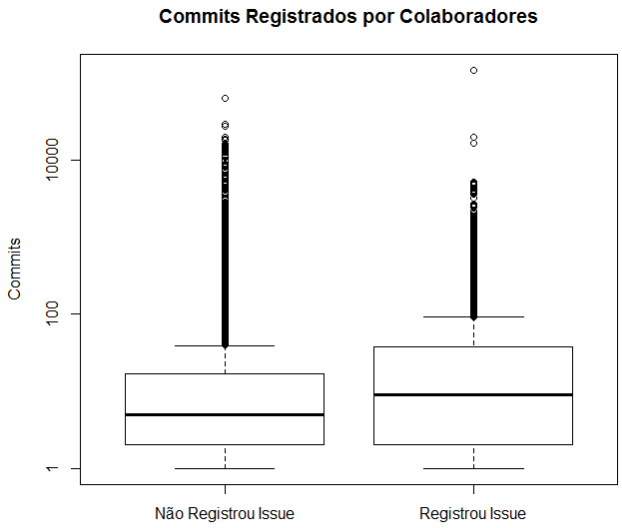

Figura 21 - *Commits* registrados por colaboradores que registraram *issues* ou não. Observa-se a tendência de que colaboradores que registram *issues* façam um menor número de *commits.*

Concluímos que, para a amostra analisada, os colaboradores que registram *issues* tendem a realizar menos *commits* do que aqueles que não registraram nenhum *issue*. Conforme apresentado anteriormente, um grande grupo de usuários apenas contribui para os projetos fazendo *commits*, como desenvolvedores, enquanto aqueles que contribuem com o registro de *issues* possuem uma participação menor na execução de *commits*, se limitando a relatar os problemas encontrados durante o uso do software.

4.7.3 Número de Seguidores

Nesse tópico avaliaremos a relação entre o número de *issues* registrados e a popularidade de um colaborador, medindo essa popularidade através do número de seguidores do colaborador. A correlação de Spearman entre estas séries de dados é negativa e muito fraca (–0,063) na amostra analisada, Esse índice mostra que não existe uma relação forte entre o número de seguidores e o número de *issues* registrados por um colaborador. A Figura 22 apresenta um *scatterplot* dessa relação.

Verificamos que a média de seguidores para os colaboradores que registraram *issues* é de 144,43, com mediana de sete seguidores (foi retirado dessa amostra um *outlier* que possuía mais de 20 milhões de seguidores; com ele, a média seria de 1057,27 seguidores). Para colaboradores que não registraram *issues* a média é de 157,04, com mediana de 15 seguidores. Utilizando o teste WMW identificamos uma distribuição significativamente diferente (p-value < 0,001) entre o grupo de usuários que registrou e

que não registrou *issues*, com tamanho de efeito pequeno (59%). Portanto, os colaboradores que registram *issues* tendem a possuir menos seguidores do que aqueles que não registram *issues*. Esse comportamento pode estar ligado ao fato de que desenvolvedores costumam ter mais seguidores do que *reporters.* A Figura 23 apresenta um *boxplot* dessa distribuição.

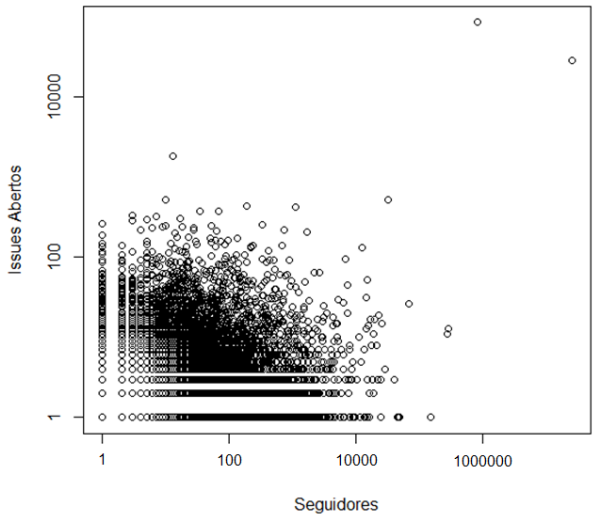

Figura 22 - *Issues* registrados por um colaborador e o número de seguidores. Observa-se uma correlação muito fraca entre o seguidores e o número de *issues* registrados por um colaborador.

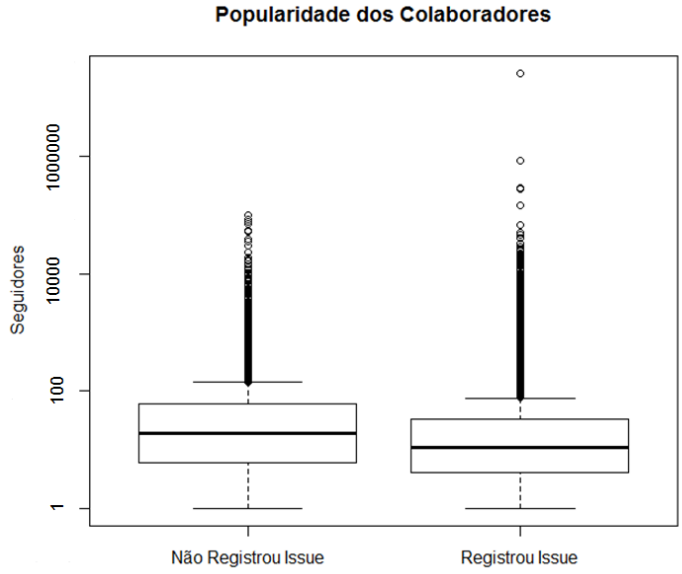

Figura 23 - Número de usuários seguidos por um colaborador que registrou *issues* ou não. Observa-se a tendência de que colaboradores que registram *issues* tenham menos seguidores.

4.7.4 Número de Seguidos

Nesse tópico será avaliada a relação entre o número de usuários seguidos pelos colaboradores e a quantidade de *issues* registrados pelos mesmos. A correlação de

Spearman entre esses conjuntos é muito fraca (-0,007), ou seja, não existe uma relação entre o número de usuários seguidos e o de *issues* registrados por um colaborador. Observamos que a média de usuários seguidos por colaboradores que registraram *issues* é de 437,56, com mediana de quatro usuários. Já para aqueles colaboradores que não registraram *issues* a média é de 42,11, com mediana de seis usuários seguidos. A Figura 24 apresenta um *scatterplot* sobre essa relação. Utilizando o teste WMW identificamos uma distribuição significativamente diferente (p-value < 0,001) entre o grupo de usuários que registrou e que não registrou *issues*, com tamanho de efeito pequeno (54%). Sendo assim, em 54% das vezes os colaboradores que registraram *issues* seguem mais usuários do que aqueles que não registraram *issues*. A Figura 25 apresenta um *boxplot* dessa distribuição utilizando a escala logarítmica, onde foram retiradas 26% dos colaboradores que registraram *issues* e 25% de colaboradores que não registraram *issue*, pois esses não seguiam nenhum usuário quando os dados foram coletados.

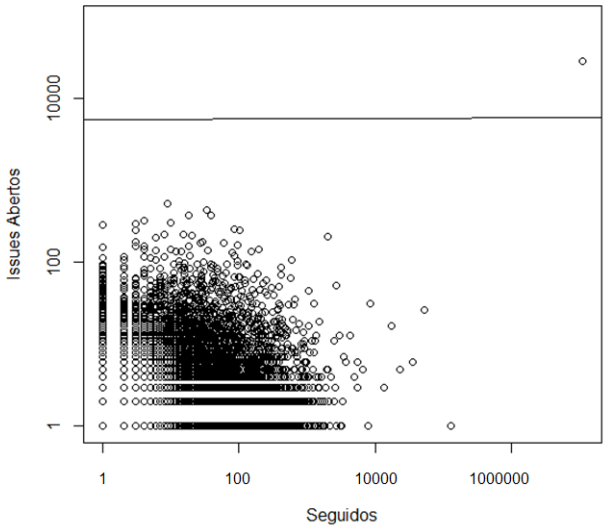

Figura 24 - Usuários seguidos por colaboradores e o número de *issues* registrados. Observa-se uma correlação muito fraca entre o número de seguidos e o número de *issues* registrados por um colaborador.

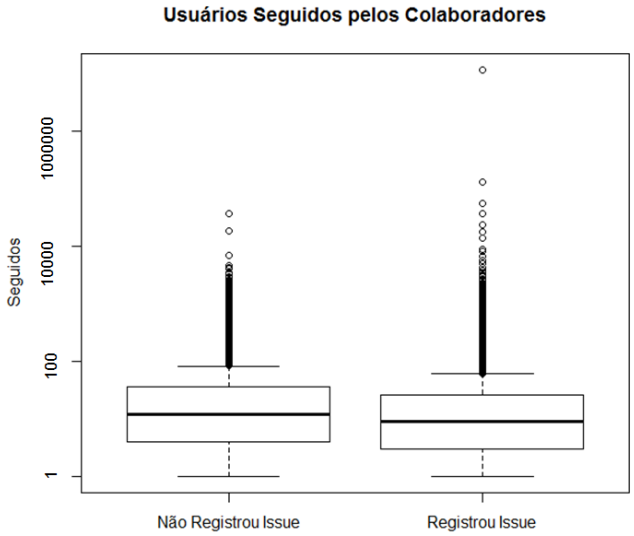

Figura 25 - Número de seguidos por colaboradores que registraram *issues* ou não. Observa-se a tendência de que colaboradores que registram *issues* tenham sigam um número maior de usuários do GitHub.

Concluímos que, para a amostra estudada, os colaboradores que registram *issues* tendem a seguir um número maior de usuários do que aqueles que não registram *issues*. Esse comportamento pode estar ligado ao fato de que esses colaboradores costumam acompanhar o andamento de projetos de outros usuários, contribuindo para estes projetos com o registro dos *issues* que encontram durante o uso do software.

4.7.5 Repositórios Públicos

Encerrando essa questão de pesquisa, avaliaremos a relação entre a quantidade de *issues* registrados por um colaborador e o número de repositórios públicos que possui no GitHub. Calculando a correlação de Spearman observamos uma relação muito branda (-0,024) entre o número de *issues* e o número de repositórios públicos de um colaborador. A Figura 26 apresenta o *scatterplot* dessa relação.

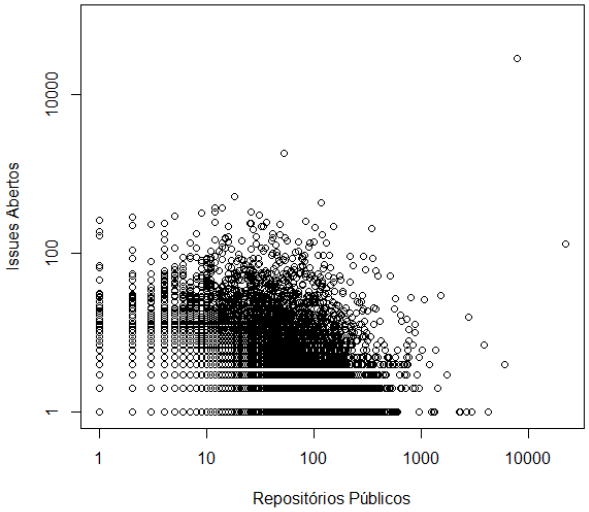

Figura 26 - A relação entre o número repositórios públicos e o número de *issues* registrados. Observa-se uma correlação muito fraca entre o número de repositórios públicos e o número de *issues* registrados por um colaborador.

Observamos que a média de repositórios públicos possuídos por colaboradores que registraram *issues* é de 38,6, com mediana de 19 repositórios. Já para aqueles colaboradores que não registraram *issues* a média é de 38,9, com mediana de 25 repositórios. Usando o teste WMW, identificamos uma distribuição significativamente diferente (p-value < 0,001) entre o grupo de usuários que registrou e que não registrou *issues*, com tamanho de efeito pequeno (55%). A Figura 27 apresenta um *boxplot* dessa distribuição utilizando a escala logarítmica, onde foram retiradas 3% das ocorrências de colaboradores que registraram *issues* e 1% de colaboradores que não registraram *issue*, pois esses não possuem nenhum repositório público.

Concluímos que usuários que registraram *issues* tendem a possuir um número ligeiramente menor de repositórios públicos do que aqueles que não registraram nenhum *issue* na amostra estudada. Esse comportamento pode estar ligado ao fato de que os donos de repositórios geralmente participam como desenvolvedores e executam *commits*, enquanto os *reporters* não precisam possuir repositórios para registrar *issues*, podendo fazer essa ação em repositórios de outros usuários.

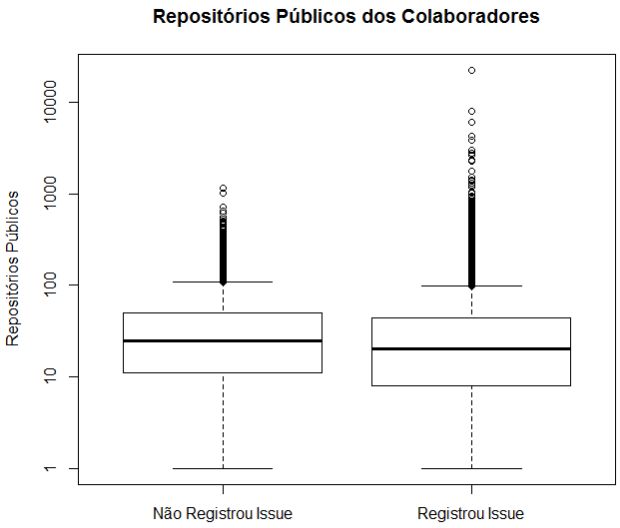

Figura 27 - Número de repositórios públicos de um colaborador que registrou *issues* ou não. Observa-se a tendência de que colaboradores que registram *issues* tenham menos repositórios públicos.

Com essa questão de pesquisa, concluímos que as características de um colaborador, como o tempo desde a criação da sua conta, o número de seguidores e usuários seguidos e o número de repositórios públicos têm pouca correlação sobre o número de *issues* registrados por ele. Dessa forma, a hipótese H5 pode ser negada, visto que não existe relação forte entre o tempo de criação da conta de um colaborador, seu número de seguidores, seguidos e repositórios públicos com a quantidade de *issues* reportados pelo mesmo. Já o número de *commits* realizados pelo colaborador tem uma correlação pequena com o número de *issues* registrados por ele. Identificamos que os usuários que registram *issues* tendem a realizar um número ligeiramente menor de *commits* que os demais colaboradores.

4.8 RQ8 – Qual a relação entre as características de um *issue* o seu encerramento por *commit*?

Nessa questão de pesquisa serão investigadas as diferentes características presentes em *issues* encerrados por *commit* em comparação com *issues* encerrados através da interface com o usuário da ferramenta de *issue tracking* do GitHub. Para isso, utilizaremos como amostra os 159.118 *issues* no estado “fechado” presentes entre os 226.340 *issues* da amostra. Desses, 7.000 *issues* foram encerrados através de *commit*.

A funcionalidade de encerramento de *issues* através de *commits* é importante pois pode ser utilizada para identificar a relação entre o código-fonte do projeto e os *issues* que foram abertos no mesmo. Assim, é uma forma simples de evidenciar qual alteração realizada em um projeto foi a responsável por encerrar um *issue* já identificado, possibilitando a rastreabilidade entre os módulos de código-fonte e os *issues.*

A média de *issues* encerrados através de *commit* para projetos com pelo menos um *issue* fechado é de 0,8 *issue*, representando uma média de 4% *issues* encerrados por *commit* em relação ao total de *issues* desses projetos. Para projetos com pelo menos um *issue* encerrado por *commit*, a média de *issues* encerrados desta forma é de 6,5 *issues*, indicando que em média 28% dos *issues* do projeto são encerrados por *commit*, com mediada de dois *issues*. As cinco linguagens que mais tiveram *issues* encerrados através de *commits* são JavaScript (com 1.509 *issues*, 4% dos *issues* registrados em projetos com essa linguagem), Java (com 1.377 *issues*, 3% dos *issues* registrados em projetos com essa linguagem), C++ (com 778 *issues*, 4% dos *issues* registrados em projetos com essa linguagem), Python (com 722 *issues*, 3% dos *issues* registrados em projetos com essa linguagem) e PHP (com 490 *issues*, 3% dos *issues* registrados em projetos com essa linguagem). Podemos observar também que essas linguagens estão entre as dez que mais possuem *issues* registrados.

4.8.1 Tempo de Correção

A primeira característica investigada é o tempo de correção de um *issue*. A média desse tempo para *issues* encerrados por *commit* é de 27,61 dias, com mediada de 2 dias. Para *issues* que não foram encerrados por *commit*, a média é de 23,8 dias, com mediana de zero dias (70% desses *issues* foram encerrados com menos de um dia desde sua abertura). Utilizando o teste WMW identificamos uma distribuição significativamente diferente do tempo de correção entre os *issues* encerrados por *commit* e os que não foram encerrados dessa forma (p-value < 0,001), com tamanho de efeito médio (65%) (COHEN, 1992), demonstrando que em 65% das vezes os *issues* fechados por *commit* terão maior tempo de vida até sua correção. A Figura 28 apresenta dois *boxplots* em escala logarítmica comparando estas distribuições.

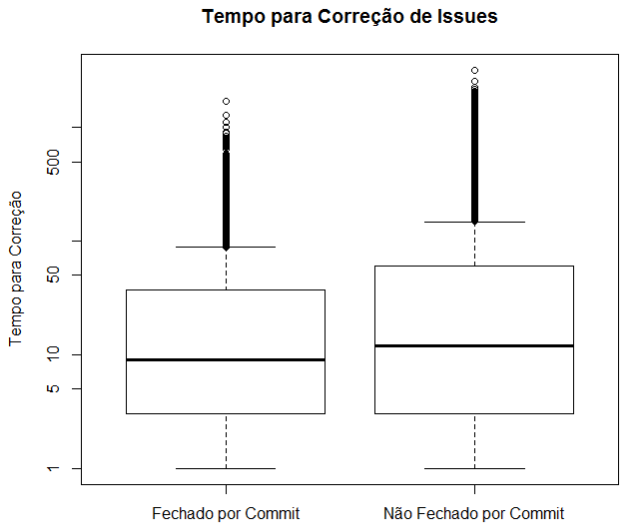

Figura 28 - Tempo de correção para *issues* encerrados por *commit* e encerrados manualmente. Observa-se a tendência de que *issues* encerrados por *commit* possuam maior tempo para encerramento.

4.8.2 Presença de Rótulos

Dos 7.000 *issues* encerrados por *commit,* apenas 4.219 possuíam rótulos. O teste chi-quadrado, utilizado para saber se os dados de dois grupos são estatisticamente diferentes, indica que o percentual de *issues* encerrados por *commit* é significativamente diferente entre os *issues* que possuem rótulos e os *issues* que não possuem rótulos, com tamanho de efeito pequeno (x^2 = 278.1, p-value < 0,001, Cramer-V (tamanho de efeito) = 4.18%). Apenas 4% dos *issues* que não possuem rótulos ou possuem rótulos fora do grupo dos dez rótulos mais utilizados foram encerrados através de commits, enquanto 6% dos *issues* que possuem rótulos entre os dez grupos de rótulos mais utilizados foram fechados via *commit.*

Utilizando novamente o teste chi-quadrado, verificamos que existe diferença significativa do porcentual de *issues* encerrados por *commit* entre os *issues* possuem algum rótulo entre os dez grupos de rótulos mais utilizados e aqueles que possuem rótulos fora destes grupos ou que não possuem rótulos. O tamanho de efeito para esse teste também é pequeno (x^2 = 114.34, p-value < 0.001, Cramer-V = 2.6%).

Os cinco rótulos mais utilizados em *issues* encerrados por *commits* pertencem aos seguintes grupos: BUG (com 1.768 *issues* encerrados por *commit* ou 19% dos *issues* registrados com rótulos de BUG); ENHANCEMENT (com 1.392 *issues* encerrados por *commit* ou 6% dos *issues* registrados com rótulos deste grupo); FEATURE (com 335

issues encerrados por *commit* ou 13% dos *issues* registrados com rótulos de FEATURE); PRIORITY (com 211 *issues* encerrados por *commit* ou 0,3% dos *issues* registrados com rótulos deste grupo); e USER INTERFACE (com 92 *issues* encerrados por *commit* ou 20% dos *issues* registrados com rótulos deste grupo). Os grupos BUG, ENHANCEMENT, FEATURE e PRIORITY, além de serem os que mais possuem *issues* encerrados via *commit*, também estão entre os dez grupos de rótulos com mais *issues* (em geral) registrados.

Observamos que a funcionalidade de encerramento de *issues* por *commits* é mais utilizada para *issues* relacionados a *bugs* ou melhorias de código, dando sustentação à hipótese H6. Assim, podemos concluir que essa funcionalidade é mais utilizada para encerramento de *issues* diretamente ligados ao código-fonte do projeto, enquanto que *issues* relacionados a outros aspectos são encerrados de forma manual, fora de *commits.*

4.8.3 Número de Comentários por *Issue*

Investigamos o número de comentários recebidos por *issues* fechados por *commit.* A média de comentários para os *issues* encerrados através de *commits* é de 1,7 comentário por *issue*, com mediana de zero (56% dos *issues* encerrados por *commit* não receberam nenhum comentário). Para *issues* que não foram encerrados por *commits* essa média é de 2,4 comentários por *issue*, com mediana de um comentário. Utilizando o teste WMW identificamos uma distribuição significativamente diferente do número de comentários para *issues* encerrados por *commit* e os que não foram encerrados dessa forma (p-value < 0,001), com tamanho de efeito pequeno (62%).

Verificamos então que o número de comentários recebidos por *issues* encerrados por *commit* é ligeiramente menor do que para os *issues* encerrados de forma manual. Um possível motivo para essa diferença é a necessidade de acessar a funcionalidade de *issue tracking* quando se fecha um *issue* manualmente, o que pode motivar os colaboradores a fazerem comentários, visto que essa opção fica muito próxima do comando de encerramento manual de *issue*. Já quando o *issue* é encerrado por *commit,* o colaborador não tem a opção de realizar comentários no *issue* no momento da realização do *commit*, ficando apenas registrado no *issue* que ele foi encerrado via *commit*. A Figura 29 apresenta um boxplot em escala logarítmica dessa distribuição.

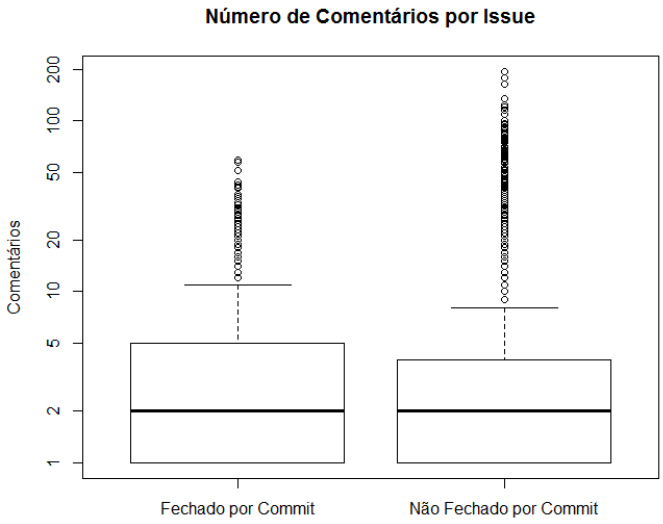

Figura 29 - Número de comentário para *issues* encerrados por *commit* e não encerrados por *commit*. Observa-se que o número de comentários recebidos por *issues* encerrados por *commit* tende a ser menor do que *issues* encerrados manualmente.

Nessa questão observamos que *issues* encerrados por *commits* tendem a possuir um maior tempo de correção do que aqueles que não foram encerrados por *commit*. Também observamos um comportamento diferente em relação ao encerramento por *commit* para *issues* que possuem rótulos em relação àqueles que não possuem, com pequena predominância de encerramento via *commit* para aqueles *issues* que possuem rótulos. Além disso, observamos que os *issues* com rótulos componentes dos dez grupos de rótulos mais utilizados no registro de *issues* em geral também possuem um porcentual ligeiramente maior de encerramento via *commit* do que aqueles que não possuem rótulos nestes grupos. Em relação ao número de comentários recebidos por um *issue*, verificamos que aqueles que foram encerrados por *commit* tendem a receber um número menor de comentários, possivelmente porque o encerramento via *commit* não possibilita a inclusão de comentários no *issue*.

4.9 RQ9 – Qual a característica dos desenvolvedores que encerram *issues* via *commit*?

Nessa questão de pesquisa serão avaliadas as características presentes em um colaborador e a sua relação com a utilização da funcionalidade de encerramento de *issues* através de *commits*. Para a realização dessa análise será utilizada a mesma amostra descrita na questão de pesquisa anterior, contendo 40.885 colaboradores. Avaliaremos apenas a relação entre o número de *issues* encerrados por *commit* e o número de *commits* registrados por um colaborador. Não avaliaremos a relação entre os

issues encerrados por *commit* e outras características do colaborador, como o tempo de criação da sua conta, o número de seguidores, seguidos e repositórios públicos, pois na sétima questão de pesquisa identificamos que essas relações não são relevantes, tendo encontrado tamanho de efeito pequeno para todas estas análises.

Nessa questão de pesquisa avaliaremos a relação entre o número de *commits* registrados por um colaborador e (a) o número de *issues* encerrados por *commit, (b)* o número de *issues* que possuem rótulos encerrados por *commit,* e (c) o número de eventos de encerramento de *issues* externos ao repositório por *commits.* O número de eventos de encerramento de *issues* externos por *commits* é descrito como o número de vezes que um usuário encerrou um *issue* através de um *commit*, porém o *issue* não pertencia ao repositório em que foi realizado o *commit.* Esse comportamento é observado quando o *issue* está presente no repositório pai do repositório onde o *commit* foi registrado.

4.9.1 *Issues* Encerrados por *Commits*

Nesta seção investigamos a correlação entre o número de *issues* encerrados por *commit* e o número de *commits* registrados por um colaborador utilizando Spearman's *rho*. Encontramos um índice de 0,211, que de acordo com COHEN (1992) aponta uma correlação pequena entre os conjuntos. A Figura 30 apresenta um *scatterplot* em escala logarítmica dos dois conjuntos de dados.

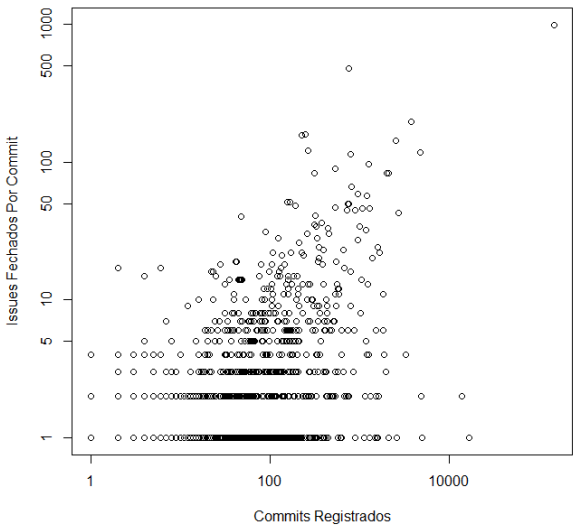

Figura 30 - *Issues* fechados por *commit* e o número de *commits* realizados pelo colaborador. Observa-se uma correlação fraca entre o número de *commits* e o número de *issues* encerrados por *commit.*

Avaliando o número de *commits* registrados, verificamos que a média para os colaboradores que encerraram ao menos um *issue* através de *commits* é de 270,41, com mediana de 42 *commits*. Já para aqueles que não encerraram *issues* através de *commits* a média é de 44,39, com mediana de um *commit*. O teste WMW identificou uma distribuição significativamente diferente (p-value < 0,001) entre o grupo de colaboradores que encerrou *issues* através de *commits* e aqueles que não o fizeram, com tamanho de efeito grande, segundo COHEN (1992), de 82%. A Figura 31 apresenta um boxplot da distribuição desses dois grupos.

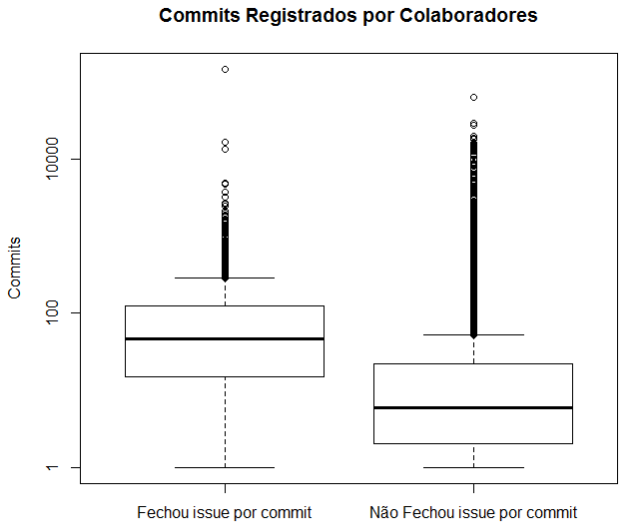

Figura 31 - Número de *commits* registrados por colaboradores que encerraram *issues* através de *commit* ou não. Observa-se que colaboradores que utilizam a funcionalidade de encerramento de *issues* através de *commits* tendem a realizar um maior número de *commits*.

Concluímos que o número de *commits* realizados por um colaborador possui correlação pequena com a quantidade de *issues que* ele encerrou através de *commits.* Verificamos também que colaboradores que utilizam a funcionalidade de encerramento *issues* através de *commits* tendem a possuir maior número de *commits* do que aqueles que não a utilizam.

4.9.2 *Issues* com Rótulos Encerrados por *Commit*

Utilizando Spearman's rho, verificamos que o índice de correlação entre o número de commits registrados por um colaborador e o número de issues com rótulos encerrados pelo mesmo através de commits é de 0,158, apontando para a existência de uma correlação pequena (COHEN, 1992). A Figura 32 apresenta o scatterplot para essa relação.

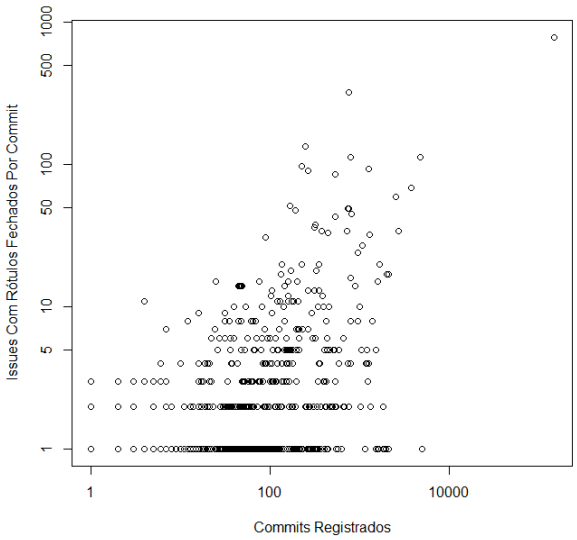

Figura 32 - Issues com rótulos fechados por *commit* e o número de *commits* realizados pelo colaborador que encerrou o *issue*. Observa-se uma correlação fraca entre o número de *commits* e o número de *issues* com rótulos encerrados por *commit.*

Verificamos que a média de *commits* realizados por colaboradores que encerraram *issues* com rótulos através de *commits* é de 369,49, com mediana de 53 *commits*. Já para aqueles que não encerraram *issues* com rótulos através de *commits* a média é de 45,98, com mediana de um *commit*. O teste WMW identificou uma distribuição significativamente diferente (p-value < 0,001) entre o grupo de colaboradores que encerrou *issues* com rótulos através de *commits* e aqueles que não o fizeram, com tamanho de efeito grande (82%), segundo COHEN (1992). A Figura 33 apresenta um *boxplot* da distribuição desses dois grupos.

Nesse tópico, assim como no anterior, encontramos uma correlação pequena entre o número de *commits* registrados e o número de *issues* com rótulos encerrados através de *commits.* Além disso, verificamos, conforme o esperado, que aqueles colaboradores que utilizam a funcionalidade de encerramento de *issues* por *commit*, inclusive para *issues* com rótulos, tendem a executar mais *commits* do que aqueles que não utilizam a funcionalidade.

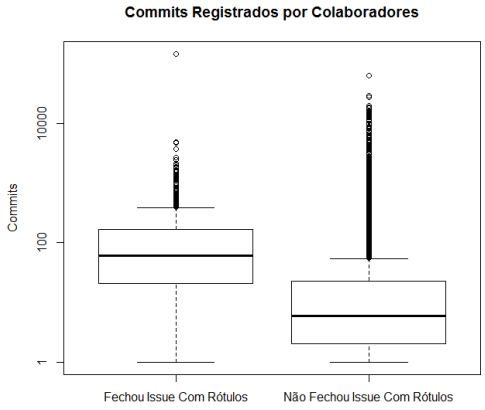

Figura 33 - Número de *commits* realizados por colaboradores que encerraram *issues* com rótulos através de *commit* ou não. Observa-se que colaboradores que utilizam a funcionalidade de encerramento de *issues* através de *commits*, inclusive para *issues* com rótulos, tendem a realizar um maior número de *commits*

4.9.3 *Issues* Externos Encerrados por *Commit*

Nesta seção analisamos a correlação entre o número de *commits* e o número de *issues* externos encerrados por *commit* por um colaborador, que totalizam 14.951 *issues*. Esse número diz respeito à quantidade de vezes que um colaborador utilizou a funcionalidade de encerramento de *issues* por *commit*, porém o *issue* não estava presente no repositório onde o *commit* foi realizado. *Issues* externos não tiveram seus dados coletados para análise. Somando os *issues* presentes nos repositórios (7.000) aos externos aos repositórios (14.951), temos um total de 21.951 *issues* encerrados por *commit*. Identificamos um índice de correlação de 0,155 (Spearman), que representa uma correlação pequena entre os dois conjuntos (COHEN, 1992). A Figura 34 apresenta um *scatterplot* para essa relação.

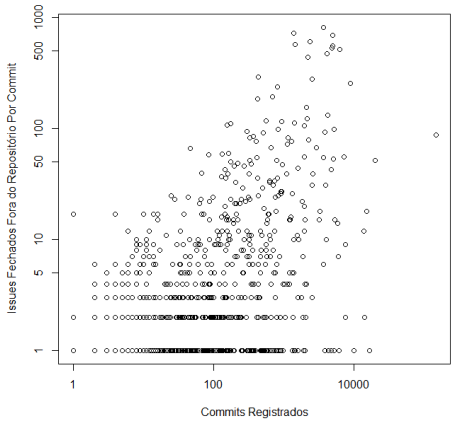

Figura 34 - *Issues* externos fechados por *commit* e o número de *commits* realizados pelo colaborador. Observa-se uma correlação fraca entre o número de *commits* e o número de *issues* externos encerrados por *commit.*

Verificamos que a média de *commits* realizados por colaboradores que encerraram *issues* externos através de *commits* é de 521,08, com mediana de 19 *commits*. Já para aqueles que não encerraram *issues* externos através de *commits* a média é de 39,07, com mediana de um *commit*. O teste WMW identificou uma distribuição significativamente diferente (p-value < 0,001) entre o grupo de colaboradores que encerrou *issues* externos através de *commits* e aqueles que não o fizeram, com tamanho de efeito pequeno (23%), segundo COHEN (1992). A Figura 35 apresenta um *boxplot* da distribuição dos grupos.

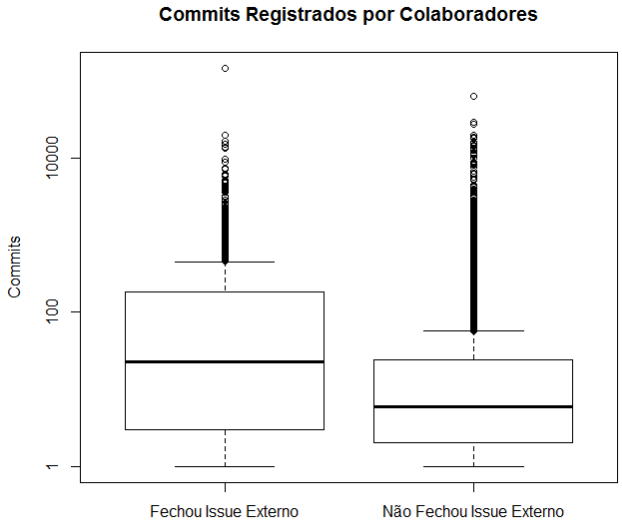

Figura 35 - Número de *commits* realizados colaboradores que encerraram *issues* externos através de *commit* ou não. Observa-se que colaboradores que utilizam a funcionalidade de encerramento de *issues* através de *commits* tendem a realizar um maior número de *commits*, inclusive para *issues* externos ao repositório

Concluímos que existe uma correlação fraca entre o número de *commits* realizados e o número de *issue* externos encerrados através de *commits*. Também confirmamos que os colaboradores que utilizam a funcionalidade de encerramento de *issues* por *commit*, inclusive para *issues* externos, tendem a realizar mais *commits* do que os colaboradores que não utilizam a funcionalidade de encerramento por *commit* para esse tipo de *issue*. Com esse resultado, temos evidências para a sustentação da hipótese H7.

4.10 Discussão e Sumário dos Resultados

Nesse capítulo foram apresentados as análises e os resultados encontrados para cada questão de pesquisa descrita no capítulo anterior. Entre primeira e a sexta questões de pesquisa, o estudo atual replicou as análises realizadas por BISSYANDE *et al*. (2013), buscando identificar se as conclusões alcançadas pelos autores se mantinham após três

anos desde a realização do estudo. Para essas questões, foram encontrados resultados que corroboram aqueles encontrados pelos autores do artigo original.

Assim como os autores do artigo original, concluímos que projetos *open source* não recebem um grande número de *issues*, visto que apenas 11% da amostra estudada possui *issues* registrados (H1 rejeitada). Também verificamos que não existem grupos bem definidos de desenvolvedores e *reporters* de *issues* nos projetos analisados, visto que grande parte daqueles que atuam em um papel também executa atividades relacionadas ao outro papel (H2 rejeitada). Por outro lado, verificamos que existe uma forte correlação entre o número de *watchers* e *forks* de um projeto e a quantidade de *reporters* de *issues* envolvidos e *issues* registrados, evidenciando assim que o ambiente distribuído e colaborativo tem uma influência positiva na utilização da ferramenta de *issue tracking* (H3 sustentada). Encerrada a replicação, concluímos também que a quantidade de *reporters* de *issues* e o tempo de correção não possuem uma correlação relevante (H4 rejeitada).

Outra característica interessante identificada durante a replicação é que o perfil de projetos que mais utilizam a ferramenta de *issue tracking* se manteve o mesmo daquele observado por BISSYANDE *et al*. (2013), sendo eles projetos mais antigos, com maior base de código (LOC), que possuem uma grande equipe de desenvolvimento e cujos donos são populares (ou seja, têm mais seguidores). Confirmamos que existe uma correlação fraca entre o número de *issues* e o número de linhas de código de um projeto. Identificamos as dez linguagens de programação que mais tiveram *issues* registrados e observamos a grande frequência de linguagens para programação *web* nessa relação. Também elencamos os rótulos mais utilizados para classificação de *issues* e verificamos a grande utilização de rótulos como BUG e ENHANCEMENT, relacionados à correção de erros e à melhoria do código-fonte.

A partir da sexta questão de pesquisa, o presente estudo buscou analisar novos pontos de vista referentes a utilização da ferramenta de *issue tracking*, agora não olhando apenas para as características do projeto, mas também as características dos colaboradores desses projetos e dos próprios *issues.* Com a análise dessas questões de pesquisa, chegamos a algumas conclusões, sendo uma delas a tendência de *issues* que possuem rótulos serem encerrados mais rapidamente do que aqueles que não possuem.

Por outro lado, verificamos que não há um relação significativa entre o número de comentários que um *issue* recebe e o seu tempo de correção.

Referente às características dos colaboradores, identificamos que os colaboradores que registram *issues* tendem a realizar um menor número de *commits* no código-fonte, ainda que previamente não tenhamos encontrado diferença significativa entre os papéis de colaborador e *reporter*. Por outro lado, não foram identificados resultados relevantes em relação às características de tempo de criação da conta do usuário, seu número de seguidores, o número de usuários que ele segue e seu número de repositórios públicos.

Concluímos também que o número de *commits* realizados por um colaborador tem relação com o número de *issues* que encerrou através de um *commit*, para *issues* internos ou externos ao projeto, com ou sem rótulos. Assim, verificamos que os colaboradores que mais registram *commits* tendem a utilizar a funcionalidade de encerramento de *issues* através de *commits* com maior frequência.

Já em relação às características dos *issues*, verificamos que aqueles encerrados através de *commits* tendem a possuir um tempo de vida maior do que os encerrados através da interface com o usuário e recebem um número menor de comentários. Também verificamos que a funcionalidade de encerramento por *commit* é bastante utilizada para fechar *issues* que possuem rótulos e que os principais rótulos de *issues* encerrados desta forma possuem relação direta com o código-fonte, como BUG, FEATURE, ENHANCEMENT e USER INTERFACE.

4.11 Considerações Finais

Nesse capítulo foram apresentadas as análises realizadas para responder às questões de pesquisa relacionadas no Capítulo 3. Ao final do capítulo, foi realizada uma discussão dos resultados identificados.

O próximo capítulo apresentará as conclusões finais do estudo realizado. Nele, serão discutidos os resultados encontrados após as análises realizadas e se esses resultados atenderam aos objetivos apresentados no começo desse estudo. Além disso, serão apresentadas as limitações desse trabalho e ideias para evoluções futuras.

5. Conclusão

Nesse capítulo, apresentaremos as conclusões finais dessa Dissertação, com base nos objetivos descritos no Capítulo 1. Em seguida, trataremos das limitações do estudo realizado e as possibilidades de trabalhos futuros vislumbradas ao final do estudo.

5.1 Contribuições

Nessa Dissertação apresentamos um estudo projetado para atender a dois objetivos principais. O primeiro objetivo foi replicar o trabalho realizado por BISSYANDE *et al*. (2013), buscando verificar se as conclusões alcançadas pelos autores continuam válidas após três anos do seu estudo. O segundo objetivo era analisar as características dos *issues* encerrados pela funcionalidade de encerramento de *issues* por *commits* que é disponibilizada pela plataforma GitHub. Podemos elencar as seguintes contribuições alcançadas após a realização desse estudo:

- Confirmação dos resultados alcançados no estudo replicado (BISSYANDE *et al*., 2013), acrescentando novas evidências às conclusões alcançadas pelos autores:
    - H1 rejeitada: projetos que recebem um maior número de registros de *issues* tendem a ser mais antigos, com maior base de código, equipe maior e donos mais populares. Porém, a utilização da ferramenta de *issue tracking* ainda não é muito difundida, se limitando a uma pequena fração dos projetos analisados;
    - H2 rejeitada: em geral, não existem grupos de usuários dedicados a desenvolver ou registrar *issues* em um projeto;
    - H3 sustentada: o ambiente colaborativo e distribuído de um projeto leva a um uso mais intensivo da ferramenta de *issue tracking;*
    - H4 rejeitada: não existe forte correlação entre o tempo de correção de *issues* e o número de colaboradores que registram *issues.*

- Realização de um estudo focado na utilização da funcionalidade de encerramento de *issues* por *commit*, com as seguintes conclusões a este respeito:
    - H5 rejeitada: a única característica de um colaborador relevante para estimar o uso da ferramenta de *issue tracking* é a quantidade de *commits* realizados por esse colaborador;
    - H6 sustentada: os issues encerrados por *commit* estão, em sua maior parte, relacionados à correção de erros e a melhorias no código-fonte do projeto;
    - H7 sustentada: desenvolvedores que utilizam a funcionalidade de encerrar *issues* através de *commits* tendem a realizar um número maior de *commits*.
- Conclusão de que a funcionalidade de encerramento de *issues* através de *commits* pode ser útil para identificar o relacionamento de um *issue* com o código-fonte do projeto, porém seu uso se limita a 3% ou 4% dos *issues* registrados em um projeto. Sendo assim, o gerente de projetos deve reforçar junto à equipe a necessidade de uso desta ferramenta, para que seja criada informação suficiente sobre as relações entre *issues* e código-fonte.

5.2 Ameaças à Validade

Ao longo do planejamento e execução deste estudo identificamos alguns pontos que devem ser citados como ameaças à validade dos seus resultados. Dentre estas ameaças, algumas existem devido a premissas assumidas durante a realização do trabalho e outras devido a características particulares do assunto mapeado. Nessa seção, as ameaças à validade são apresentadas e discutidas.

A primeira ameaça se refere ao número de *commits* existentes em cada projeto analisado. Visto que o segundo objetivo desse estudo buscou compreender as características relacionadas a utilização da funcionalidade de encerramento de *issues* via *commits*, seria mais correto analisarmos projetos que possuíssem uma grande quantidade de *commits*. Porém, ao utilizarmos para essa análise a mesma amostra coletada para a replicação do estudo de BISSYANDE *et al.* (2013) não filtramos os projetos pelo número de *commits* realizados, assim como o estudo original. Sendo assim, em nossa amostra existem projetos com apenas um ou nenhum *commit.*

Outra ameaça identificada está no método escolhido para identificar os colaboradores distintos. Nesse estudo, todos os registros de colaborador que possuíssem o mesmo

nome e e-mail foram considerados como apenas um colaborador. Quando o nome ou o e-mail era diferente foi tratado como um colaborador distinto. Essa forma de desambiguação não é a mais segura, visto que uma única pessoa pode possuir mais de um usuário cadastrado no GitHub. Neste caso, essa pessoa foi contabilizada duas vezes e suas contribuições com os projetos foram divididas entre esses dois registros.

Citamos também como ameaça a validade o fato de termos utilizado uma ferramenta diferente para contagem de linhas de código de um projeto durante a replicação do estudo, utilizando a ferramenta cLOC e não a ferramenta SLOCCount, conforme BISSYANDE *et al.* (2013). Essa escolha se deve ao fato da ferramenta utilizada no artigo original não estar disponível para ambiente Windows. Como as ferramentas podem possuir métodos distintos para contagem de linhas de código, podem ocorrer diferenças na contagem final do tamanho dos arquivos.

5.3 Limitações e Trabalhos Futuros

Nessa Dissertação identificamos como uma limitação que impediu a realização de análises mais específicas sobre o relacionamento entre o código-fonte e o *issue* o fato de não termos avaliado diretamente o conteúdo dos *commit* realizados. Não coletamos informações sobre os módulos e arquivos que foram alterados em cada *commit*[12] e assim não pudemos investigar a relação entre eles o os *issues* fechados através de *commit.*

Nesse quesito, propomos como trabalho futuro uma análise mais detalhada desse relacionamento, onde poderemos estudar tópicos como: (i) colaboradores que mais reportam *issues* relacionados a determinado módulo do código-fonte; (ii) o relacionamento entre rótulos de um *issue* e os módulos do projeto; (iii) colaboradores que mais realizam *commits* que encerram *issues* e envolvam determinado módulo, entre outras. Com essas análises, poderemos estudar meios de identificar os colaboradores mais aptos a solucionar ou registrar um *issue* relacionado a um módulo do projeto, devido ao seu histórico de resolução ou registro de *issues* semelhantes.

Outra limitação a ser explorada em trabalhos futuros é o fato de não termos avaliado *issues* externos aos repositórios coletados. Hoje, a funcionalidade de encerramento de *issues* por *commits* disponibilizada pelo GitHub permite o encerramento de um *issue* em

[12] O comando utilizado para gerar o log foi "git log --pretty=fuller". De acordo com pesquisas realizadas, o modo "fuller" traz a maior quantidade de detalhes sobre os commits, incluindo o *hash*, o autor, a data do commit e sua mensagem.

um repositório externo aquele em que o *commit* foi realizado. Nesse estudo, apenas contabilizamos o número de encerramento de *issues* externos, porém não nos aprofundamos na análise desses *issues* e dos projetos que os contém. Sendo assim, como trabalho futuro propomos uma análise mais detalhada das características dos *issues* que são encerrados por *commits* realizados em um repositório diferente do qual aquele *issue* está registrado.

Outra proposta de trabalho futuro é a replicação desse estudo, levando em consideração projetos privativos e que não possuem a intervenção de pessoas externas a equipe do projeto. Esse trabalho pode demonstrar se as características apresentadas nesse estudo, ainda que para projetos *open source*, são semelhantes ou não quando analisadas para projetos privativos.

Anexo I

GUIA PARA REPLICAÇÃO DO ESTUDO

Nesse apêndice será apresentado um guia para replicação do estudo realizado. Os tópicos abaixo apresentam as premissas e os métodos utilizados durante a replicação do estudo de BISSYANDE *et al.* (2013).

1) Seleção dos projetos para coleta.
   a. Utilizando a API do GitHub em conjunto com a biblioteca Egit, disponível para o Eclipse, colete o nome do dono do repositório e o nome do repositório, de todos os repositórios open-source existentes no GitHub até o momento;
   b. Armazene essas informações em um banco de dados;
   c. Aleatoriamente, faça a seleção da quantidade desejada de repositórios e enumere-os;
   d. Armazene o resultado em um arquivo texto ou banco de dados;

2) Guia para coleta dos dados
   a. Coleta dos Repositórios, Issues e Colaboradores:
      i. Utilizando o Egit através da plataforma Eclipse, construa código para coleta dos dados dos repositórios, *issues* e colaboradores. Esse código deve ser capaz de retornar as informações de um repositório, dois *issues* e colaboradores relacionados a ele, utilizando como chave para busca, o nome do dono do repositório e o nome do repositório;
      ii. Esse código também deve ser capaz de armazenar os dados coletados em um banco de dados ou arquivo texto. Gerando assim, uma base de dados onde as informações dos repositórios serão armazenadas.
      iii. Será necessário construir um mecanismo que receba o alerta sobre limites de requisições ultrapassadas e pare a coleta durante o prazo de 1 hora, retomando a coleta após esse prazo. Isso evitará erros e intervenção manual quando o limite de requisições for atingido;
      iv. Utilize os registros gerados na primeira seção, contendo o nome do dono do repositório e do próprio repositório, para acessar e coletar os dados dos repositórios sorteados;

v. Evite armazenar os dados coletados em memória. Sempre que os dados forem retornados pela API do GitHub, registre-o na base de dados que está sendo construída. Assim, caso haja algum erro no código, será possível reiniciar a coleta a partir do último repositório coletado;
vi. Todos os dados numéricos devem ser coletados em duas colunas, uma contendo o dado sem tratamento e outra, onde esse dado será modificado, caso o valor seja igual a 0, para o texto "NA". Esse tratamento nos dados é necessário, pois alguns testes estatísticos utilizados posteriormente não reconhecem dados que contenham o número 0. Sendo assim, todos os dados numéricos possuirão um registro sem alteração e um registro ajustado, após o tratamento. Alguns dados que sofrerão esse tratamento são: o número de *issues* do repositório, o número de desenvolvedores, o número de *followers* e *watchers;*
vii. Devem ser criados três registros de dados principais, um para os repositórios coletados, outro para armazenar os *issues* por repositório e outro para armazenar os colaboradores e suas interações para cada repositório;
viii. Deve ser criado um quarto registro para armazenar os dados de colaboradores distintos. Nesse registro serão consolidados os dados de todas as interações que um colaborador distinto realizou em todos os repositórios coletados. Nesse registro, devem ser somados o número de *commits*, *issues* abertos, fechados e fechados por *commits* em todos os repositórios nos quais o colaborador teve alguma ação. Além disso, nesse registro deve ser armazenado o tempo desde a criação da conta no GitHub, número de seguidores, de seguidos e de repositórios públicos.

b. Coleta dos *Commits* e Número de Linhas de Código
i. A coleta dos *commis* realizados pode ser realizada através do código construído com o Egit, porém consome muitas requisições, o que aumenta muito o tempo para coleta dos dados devido ao limite de requisições;
ii. Para acelerar a coleta das informações, os *commits* podem ser coletados utilizando a função de *log* disponibilizada pelo *git.* Para isso, construa um arquivo *Batch* (.bat) que repita a seguinte sequencia de ações para todos os repositórios selecionados para a amostra na sessão 1: (i) clonar do repositório para um diretório local; (ii) requisitar o *log* de *commits* do repositório; (iii)

armazenar esse *log* em um arquivo texto ou banco de dados; e (iv) excluir a cópia local do repositório;

iii. Para coletar a informação sobre o número de linhas de um repositório também é necessário utilizar o arquivo *Batch*, visto que a API do GitHub não permite o acesso a essa informação nativamente. Para isso, utilize a aplicação cLOC (Windows) ou SLOCCount (Linux) para fazer essa contagem. Com uma dessas aplicações, utilize o arquivo *Batch* gerado no item anterior para coletar a informação de linhas de código, incluindo os comandos necessários antes da exclusão da cópia local do repositório. Armazene essa informação em arquivo texto ou banco de dados.

iv. Visando consolidar as informações de repositórios, *issues* e colaboradores com as informações de *commits*, construa um código que faça as seguintes ações: (i) para cada repositório coletado, identificar os *commits* que pertencem a esse repositório; (ii) fazer a análise textual de cada *commit*, procurando pelas palavras que indicam o encerramento de um *issue*; (iii) atualizar o registro do repositório com a informação da quantidade de *issues* encerrados através de *commit;* (iv) atualizar o registro de colaboradores, contabilizando o número de *issues* que um colaborador encerrou através de *commit;* (v) Atualizar o registro de *issues*, indicando quando um *issue* foi encerrado através de um *commit; e* (vi) atualizar o registro de colaboradores distintos, consolidando as informações de interação de um colaborador em todos os projetos da amostra que ele participou.

v. Visando consolidar as informações de repositórios com a informação da quantidade de linhas de código de cada repositório, construa um código que faça as seguintes ações: (i) identificar o registro que contém a quantidade de linhas de código de um repositório; e (ii) atualizar o registro do repositório com a informação do número de linhas de código daquele repositório;

3) Guia para curadoria dos dados
   a. Após a realização da coleta, execute a curadoria usando as seguintes regras:
      i. Retire da amostra todos os projetos que não possuem a ferramenta de *issue tracking* disponibilizada pelo GitHub ativa;

ii. Retire da amostra todos os projetos que não possuem uma linguagem de programação principal, ou seja, onde a informação da linguagem do repositório está preenchida com “null”.

b. Essa curadoria pode ser realizada através da construção de um novo código, da utilização de *querys* de banco de dados ou da ferramenta Excel para trabalhar com planilhas.

4) Guia para análise

a. Durante esse estudo foram utilizados alguns testes e demonstrações estatísticas que possibilitaram a identificação dos resultados apresentados, foram eles:

i. Mann-Whitney(Wilcoxon) – Teste de hipótese para dois grupos sem distribuição normal, utilizado nas questões de pesquisa RQ1, RQ7, RQ8 e RQ9;

ii. Correlação de Spearman – Medida de correlação para grupos sem distribuição normal, utilizado nas questões de pesquisa RQ2, RQ4, RQ5, RQ6, RQ7, RQ8 e RQ9;

iii. Cramer´s V – Teste de tamanho de efeito para testes qualitativos, utilizado na questão de pesquisa RQ8;

iv. Vargha e Delaney’s – Teste de tamanho de efeito para o teste Mann-Whitney, utilizado nas questões de pesquisa RQ1, RQ7, RQ8 e RQ9;

v. Boxplot – Gráfico para demonstração da distribuição de dados, utilizado nas questões de pesquisa RQ1, RQ2, RQ4, RQ6, RQ7, RQ8 e RQ9. Utilize a escala logarítmica para possibilitar a melhor visualização de um grande volume de dados. Essa escala não pode ser utilizada em intervalo de dados que possuam o valor “0”. Por esse motivo, utilize os registros ajustados, onde o valor “0” foi trocado por “NA”, conforme dito anteriormente;

vi. Scatterplot – Gráfico para demonstração da correlação entre duas distribuições de dados, utilizado nas questões de pesquisa RQ2, RQ5, RQ6, RQ7, RQ7 e RQ9;

vii. Tabelas e Gráficos de Barras – Utilizados nas questões de pesquisa RQ1, RQ2 e RQ3, pra demonstração de dados, percentuais e comparações.

Referências Bibliográficas

[1] ANDREOU, A.S., PAPATHEOCHAROUS, E. “Towards a CBSE Framework for Enhancing Software Reuse: Matching Component Properties Using Semi-Formal Specifications and Ontologies” In: *10th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE'15)*, pp. 98 – 121, 2015

[2] BIOLCHINI, J., MIAN, P. G., NATALI, A. C. C., CONTE, T., TRAVASSOS, G. H. “Scientific research ontology to support systematic review in software engineering” In: *Advanced Engineering Informatics 21*, pp 133 – 151, 2007

[3] BISSYANDE, T. F., LO, D., LINGXIAO, JIANG, REVEILLERE, L., LE TRAON, Y. “Got Issues? Who Cares About It? A Large Scale Investigation of Issue Trackers from GitHub”. In: *2013 IEEE 24th International Symposium on Software Reliability Engineering*, pp. 188 – 197, Pasadena, Nov. 2013.

[4] CABOT, J., IZQUIERDO, J.L.C., COSENTINO, V. “Exploring the Use of Labels to Categorize Issues in Open-Source Software Projects” In: *Software Analysis, Evolution and Reengineering (SANER)*, pp. 550 – 554, 2015

[5] COHEN, J. “A Power Primer” In: *Psychological Bulletin*, pp. 155 – 159, 1992

[6] DABBISH, L., STUART, C., TASY, J., HERBSLEB, J. “Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository” In: *ACM 2012 Conference on Computer Supported Cooperative Work*, pp. 1277 – 1286, 2012.

[7] FARAH, G., CORREAL, D. “Analysis of intercrossed open-source software repositories data in GitHub” In: *8th Computing Colombian Conference (8CCC)*, pp. 1 – 6, 2013

[8] FEJZER, M., WIJTYNA, M., BURZANSKA, M., WISNIEWSKI, P., STENCEL, K. “Open Soucer is a Continual Bugfixing by a Few” In: Manolopoulos, Y.,

Trajcevski, G., Kon-Popovska, M. (eds), *Advances in Databases and Information Systems*, chapter 12, Ohrid, Macedonia, 2014

[9] GIGER, E., PINZGER, M., GALL, H.C. "Predicting the fix time of bugs" In: *2nd International Workshop on Recommendation Systems for Software Engineering*, pp. 52 – 56, 2010

[10] GOUSIOS, G. "The GHTorent dataset and tool suíte" In: *IEEE International Working Conference on Mining Software Repositories*, pp. 233 – 236, 2013.

[11] GOUSIOS, G., VASILESCU, B., SEREBRENIK, A., ZAIDMAN, A. "Lean GHTorrent: GitHub data on demand" In: *11th Working Conference on Minig Software Repositories - MSR 2014*, pp. 384 – 387, 2014.

[12] HELLER, B., MARSCHNER, E., ROSENFELD, E., HEER, J. "Visualizing collaboration and influence in the open-source software community", In: *8th Working Conference on Mining Software Repositories – MSR '11,* pp. 223, 2011.

[13] IZQUIERDO, J.L.C., COSNETINO, V., ROLANDE, B., BERGEL, A., CABOR, J. "GiLA: GitHub label analyzer" In: *International Conference on Software Analysis, Evolution and Reengineering (SANER),* pp. 479 – 483, 2015

[14] KALLIAMVAKOU, E., BLINCOE, K., SINGER, L., DAMIAN, D. "The Promises and Perils of Mining GitHub" In: *11th Working Conference on Mining Software Repositories*, pp. 92 – 101, 2014

[15] KENMEI, B., ANTONIOL, G., DI PENTA, M. "Trend Analysis and Issue Prediction in Large-Scale Open Source Systems" In: *12th European Conference on Software Maintenance and Reengineering*, pp. 73 - 82, 2008

[16] KIKAS, R., DUMAS, M., PFAHL, D. "Issue Dynamics in Github Projects" In: *16th Intl. Conference on Product-Focused Software Process Improvement (PROFES 2015), Springer*, pp. 295 – 310, 2015

[17] KO, A.J., CHILANA, P.K. "How power users help and hinder open bug reporting" In: *28th International Conference on Human Factors in Computing Systems - CHI '10*, pp. 1665 – 1674, 2010

[18] LEE, M. J., FERWERDA, B., CHOI, J., HAHN, J., MOON, J., KIM, J. "GitHub developers use rockstars to overcome overflow of News" In: *Human Factors in Compunting System*, pp. 133, 2013.

[19] LEVENSHTEIN, V. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”, *Soviet Physics Doklady* v. 10, pp. 707, 1966.

[20] MARKS, L., ZOU, Y., HASSAN, A.E. “Studying the fix-time for bugs in large open source projects” In: *7th International Conference on Predictive Models in Software Engineering - Promise '11,* pp. 1 – 8, 2011

[21] RUSK, D., COADY, Y. “Location-Based Analysis of Developers and Technologies on GitHub” In: *International Conference on Adavanced Information Networking and Applications Workshops,* pp. 681 – 685, 2014.

[22] THUNG, F., BISSYANDE T.F., LO, D., JIANG, L. “Network structure of social coding in GitHub” In: *European Conference on Software Maintenance and Reengineering (CSMR),* pp. 323 – 326, 2013

[23] VENKATARAMANI, R., ASADULLAH, A., BHAT, V., MUDDU, B. “Latent Co-development Analysis Based Semantic Search for Large Code Repositories” In: *IEEE International Conference on Software Maintenance,* pp. 372 – 375, 2013

[24] WEICHENG, Y., BEIJUN, S., BEN, X. “Mining GitHub: Why Commit Stops -- Exploring the Relationship between Developer's Commit Pattern and File Version Evolution” In: *Asia-Pacific Software Engineering Conference (APSEC),* pp. 165 – 169, 2013.

[25] WEISS, C., PREMRAJ, R., ZIMMERMANN, T., ZELLER, A. “How Long Will It Take to Fix This Bug?” In: *Fourth International Workshop on Mining Software Repositories*, 2007

[26] WEST, A. G., LEE, I. “Towards content-driven reputation for collaborative code repositories” In: *Eighth Annual International Symposium on Wikis and Open Collaboration – WikiSym ’12,* 2012.

[27] XAVIER, J., MACEDO, A., MAIA, M. A. “Understanding the popularity of reporters and assignees in the Github” In: *26th International Conference on Software Engineering and Knowledge Engineering*, pp. 484 - 489, 2014

[28] YONG, E. “Bad Copy” In: *Nature 485*, pp 298 – 300, 2012

[29] YU, L., MISHRA, A., MISHRA, D. “An Empirical Study of the Dynamics of GitHub Repository and Its Impact on Distributed Software” In: *5th International*

Workshop on Information Systems in Distributed Environment, pp. 457 – 466, 2014