UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

BACHARELADO EM ENGENHARIA DE SOFTWARE

JACQUES NIER CÂMARA MARINHO

ESTUDO SOBRE A CONTRIBUIÇÃO DE DESENVOLVEDORES BRASILEIROS EM BUILDS FALHOS DE PROJETOS JAVA

QUIXADÁ

2019

JACQUES NIER CÂMARA MARINHO

ESTUDO SOBRE A CONTRIBUIÇÃO DE DESENVOLVEDORES BRASILEIROS EM BUILDS FALHOS DE PROJETOS JAVA

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Orientador: Prof. Me. Carlos Diego Andrade de Almeida

QUIXADÁ

2019

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M29e Marinho, Jacques Nier Câmara.
Estudo sobre a contribuição de desenvolvedores brasileiros em builds falhos de projetos java / Jacques Nier Câmara Marinho. – 2019.
42 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2019.
Orientação: Prof. Me. Carlos Diego Andrade de Almeida.

1. Integração contínua. 2. Gerência de configuração. 3. Análise de dados. I. Título.

CDD 005.1

JACQUES NIER CÂMARA MARINHO

ESTUDO SOBRE A CONTRIBUIÇÃO DE DESENVOLVEDORES BRASILEIROS EM BUILDS FALHOS DE PROJETOS JAVA

Monografia apresentada no curso de Engenharia de Software da Universidade Federal do Ceará, como requisito parcial à obtenção do título de bacharel em Engenharia de Software. Área de concentração: Computação.

Aprovada em: _____/_____/_____

BANCA EXAMINADORA

Prof. Me. Carlos Diego Andrade de Almeida (Orientador)
Universidade Federal do Ceará – UFC

Prof. Dr. Lincoln Souza Rocha
Universidade Federal do Ceará - UFC

Prof. Dr. Regis Pires Magalhães
Universidade Federal do Ceará - UFC

Dedico este trabalho a Deus, que sempre me deu forças para continuar, e cuidou de mim por todos esses anos, mandando anjos, familiares e amigos para me apoiarem.

AGRADECIMENTOS

Agradeço aos meus pais, José Amado Marinho e Teresinha de Oliveira Câmara Marinho, pelo apoio durante esses anos de graduação que nunca deixaram me faltar nada e sempre foram as pessoas que mais me deram força em toda minha caminhada. Aos meus irmãos Jaime Virgínio Câmara Neto e Milkia Janne Câmara Marinho por todos os conselhos e apoio que me deram não só na minha graduação mas também em todos os momentos de dificuldade enquanto morei em Quixadá.

Agradeço também a minha noiva Sara Cilea Mendes Freire Marinho por todos os dias em que esteve me apoiando mesmo à distancia, sempre me dando forças e incentivando quando as coisas não davam certo.

Agradeço ao Prof. Me. Carlos Diego Andrade de Almeida, por toda a sua dedicação e excelente orientação. Aos professores participantes da banca examinadora, Prof. Dr. Lincoln Souza Rocha e Prof. Dr. Regis Pires Magalhães, pelas contribuições que foram imprescindíveis para a construção deste trabalho. E aqui estendo os agradecimentos a todos os professores que tive em todos os anos de estudos, pois eu sei que todos tiveram alguma participação na minha realização acadêmica.

Agradeço aos meus amigos e colegas de turma, principalmente a galera do AP301, Bruno Barreto, Amarildo Barros e Carlos Matheus pelo incentivo, horas de estudo e principalmente por tudo que eu aprendi com eles. Tenho certeza que essa amizade é da faculdade para a vida!

Agradeço a Deus por me conceder força, fé e determinação para nunca desistir do curso e nem dos meus sonhos. Obrigado meu Deus por cada uma das pessoas que passaram pela minha vida, pois eu sei que elas de alguma forma contribuíram e me ajudaram a conseguir realizar mais um sonho.

"Como é feliz o homem que acha a sabedoria, o homem que obtém o entendimento, pois a sabedoria é muito mais proveitosa do que a prata e rende mais do que o ouro."

(Provérbios de Salomão)

RESUMO

O mercado brasileiro de software está em plena ascensão, apresentando taxas de crescimento maiores que a média mundial, e inserindo o Brasil como um país de destaque em vendas de hardware, software e serviços de tecnologia. Tal ascensão tem aumentado a necessidade de mais profissionais qualificados na área, motivando algumas pesquisas para estudar a qualidade e quantidade dos profissionais de TI brasileiros. Seguindo esta tendência, este trabalho tem como objetivo realizar um estudo sobre a contribuição de desenvolvedores brasileiros em builds falhos em projetos Java, comparando com desenvolvedores do resto do mundo, com a finalidade de identificar se a contribuição de brasileiro em falhas está em conformidade com valores globais ou não. O estudo foi realizado utilizando a base de dados do TravisTorrent que contém logs de builds de projetos de código aberto armazenados no Github e que utilizam a ferramenta de integração contínua TravisCI. Os resultados obtidos neste trabalho mostraram que os brasileiros correspondem a 2% dos desenvolvedores válidos, este resultado está em conformidade com o levantamento realizado pela ABES (2019), sugerem que os brasileiros realizaram mais commits por build falho e contribuíram em menos builds falhos por projeto em comparação com desenvolvedores estrangeiros.

Palavras-chave: Integração Contínua. Gerência de Configuração. Análise de Dados

ABSTRACT

The Brazilian software market is on the rise, with growth rates higher than the world average, and has entered Brazil as a leading country in sales of hardware, software and technology services. Such a rise has increased the need for more qualified professionals in the field, motivating some research to study the quality and quantity of Brazilian IT professionals. Following this trend, this work aims to investigate the participation of Brazilian developers in failed builds in Java projects, comparing with developers from the rest of the world, in order to identify if the Brazilian contribution in failures is in compliance with global values or not. The study was conducted using the TravisTorrent database that contains open source project build logs stored in Github and using the TravisCI continuous integration tool. The results obtained in this work showed that Brazilians correspond to 2% of the valid developers, this result is in accordance with the survey carried out by ABES (2019), suggest that Brazilians performed more commits per failed build and participated in fewer failed builds per project compared to foreign developers.

Keywords: Continuous Integration. Configuration Management. Data analysis.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O mercado brasileiro de software vem ganhando cada vez mais destaque no mundo, apresentando um crescimento de 9.8% em 2018 com previsão de crescimento de 10.5% em 2019. Atualmente o mercado brasileiros de software representa 2.1% das vendas globais de hardware, software e serviços de tecnologia, tornando o Brasil líder na América Latina e nono lugar no ranking mundial de vendas neste setor (ABES, 2019).

A ascensão do mercado brasileiro de software tem aumentado a demanda por profissionais capacitados na área, segundo a Associação Brasileira de Empresas de Tecnologia da Informação e Comunicação (Brasscom), até 2020 o mercado brasileiro necessitará de mais 750 mil profissionais de Tecnologia da Informação (TI) (BRASSCOM, 2015).

Com o crescimento do mercado brasileiro de software e a necessidade de profissionais na área, alguns trabalhos foram realizados para estudar o mercado e os profissionais na área de TI no Brasil. A pesquisa realizada pela ABES (2019), Associação Brasileira das Empresas de Software, estudou o mercado brasileiro de software, apresentando dados sobre investimentos na área de TI, apontando tendências de mercado e áreas com maior previsão de crescimento. Já o levantamento realizado pela SOFTEX (2019) analisa a formação acadêmica, tempo de experiência, e idade dos profissionais de TI brasileiros. Podemos citar também o trabalho realizado por ONECHOICE.COM.BR (2017) que estuda a qualidade dos profissionais de TI brasileiros, que através de testes técnicos online constatou que 8 de cada 10 brasileiros foram reprovados nos testes técnicos.

Este trabalho segue a linha de estudos dos três trabalhos citados anteriormente. Neste trabalho foi realizado um estudo sobre a contribuição de desenvolvedores brasileiros em builds falhos, foram analisados apenas builds falhos de projetos java de código aberto que utilizam integração contínua.

A integração contínua deriva da gerência de configuração de software, e consiste em um conjunto de práticas em que trabalhos realizados diariamente por cada membro de uma equipe possam ser integrados após realização de testes, relatando os erros rapidamente, aumentando assim a coesão do software (FOWLER; FOEMMEL, 2006). Daí surgiu o TravisTorrent, que dispõe de uma grande base de dados contendo os logs de builds de integração realizados em projetos que utilizam a ferramenta de integração contínua TravisCI. O TravisTorrent foi matéria-prima para o desafio da 14ª Conferência Internacional de Mineração de Repositórios de Software (MSR'17).

Os estudos deste trabalho foram realizados na base de dados do TravisTorrent e se deu em 4 etapas: (i) Definição das questões de pesquisa; (ii) Investigação inicial e refinamento das questões de pesquisa; (iii) Coleta dos dados; e (iv) Análise dos dados. Na etapa (i) foram definidas as questões de pesquisa que iriam guiar os estudos realizados neste trabalho; Na etapa (ii) foi realizado o estudo da base de dados disponibilizada pelo TravisTorrent, e refinamento das questões de pesquisa; Na etapa (iii) foram coletados commits, builds falhos e participantes de projetos java, foi obtido também o autor de cada commit coletado e a nacionalidade de cada participante; Na etapa (iv) foi realizada a identificação dos brasileiros, builds falhos com brasileiros e projetos Java com brasileiros.

O objetivo principal deste trabalho é estudar os tipos de contribuição de desenvolvedores brasileiros, buscando identificar builds falhos com responsabilidade ou participação de brasileiros. Outro objetivo deste trabalho é medir a qualidade dos desenvolvedores brasileiros em relação aos desenvolvedores estrangeiros, levando em consideração a contribuição em builds falhos de projetos Java.

Os estudos realizados neste trabalho poderá servir de referência para a realização de outros estudos semelhantes, levando em consideração desenvolvedores de outras nacionalidades, análises sobre builds bem-sucedidos e projetos desenvolvidos em outras linguagens. Os resultados obtidos neste trabalho auxiliam no estudo sobre qual o impacto de desenvolvedores brasileiros em seus projetos, apresentando análises sobre builds falhos e commits em projetos Java e comparando com dados globais.

Este trabalho está organizado da seguinte forma: na Seção 2 são expostas as contribuições científicas que se relacionam com o tema desenvolvido neste trabalho; Na Seção 3 são apresentados os conceitos principais que fundamentaram a construção deste trabalho; Na Seção 4 são apresentados os procedimentos metodológicos executados para a realização deste trabalho; Na Seção 5 são apresentados e discutidos os resultados obtidos na realização neste trabalho; No Seção 6 são discutidos os resultados encontrados e suas implicações; Na Seção 7 são apresentadas algumas dificuldades encontradas durante a realização deste trabalho; Por fim na Seção 8 apresenta as contribuições, limitações, conclusões e possíveis trabalhos futuros.

2 TRABALHOS RELACIONADOS

A seguir serão apresentados os trabalhos que auxiliaram no desenvolvimento deste estudo, serão discutidas as semelhanças e diferenças com relação a este trabalho.

Em MAQSOOD; ESHRAGHI; ALI (2017) foram utilizados os dados de 5000 projetos open source nas linguagens Java, PHP, JavaScript, C/C++ e HTML, coletados a partir da GHTorrent. Nesse trabalho investigou-se quais os principais motivos de falhas e sucessos nos projetos. Como resultado foi identificado que popularidade, quantidade de estrelas, quantidade de forks e quantidade de assinantes são os fatores que mais influenciam para o sucesso de um projeto *open source* e que os projetos JavaScript têm uma maior taxa de sucesso devido a sua comunidade grande e ativa.

Este trabalho assemelha-se ao desenvolvido por MAQSOOD; ESHRAGHI; ALI (2017), quando propõe estudos a taxa de falhas em projetos de código aberto armazenados no Github, obtendo informações através do GHTorrent. O diferencial deste trabalho neste caso é o estudo de falhas em projetos Java levando em consideração builds falhos.

Em REBOUÇAS et al. (2017) foram analisados builds de projetos contidos na base de dados do TravisTorrent, com o objetivo de investigar a influência de desenvolvedores casuais e não casuais em sucesso ou falha de builds. Os resultados mostraram que não há diferença representativa entre o sucesso ou falha de builds entre desenvolvedores casuais e não casuais, e que um desenvolvedor casual não é um forte indicador para criar builds com falha.

O trabalho de REBOUÇAS et al. (2017) assemelha-se a este trabalho, pois são realizados estudos com o uso da base de dados do TravisTorrent e projetos de código aberto com o objetivos de identificar a influência de desenvolvedores em builds com falha. O diferencial deste trabalho neste caso é o estudo da contribuição de desenvolvedores brasileiros em builds falhos. A seguir a tabela 1 mostra a comparação deste trabalho com os trabalhos relacionados discutidos nesta seção.

Tabela 1 – Comparação deste trabalho com trabalhos relacionados

Trabalho	Utiliza a base de dados do TravisTorrent	Análises em projetos open source	Análise de builds falhos	Contribuição de desenvolvedores brasileiros
MAQSOOD; ESHRAGHI; ALI (2017)		X		
REBOUÇAS et al. (2017)	X	X	X	
Trabalho proposto	X	X	X	X

Fonte – Próprio autor

3 FUNDAMENTAÇÃO TEÓRICA

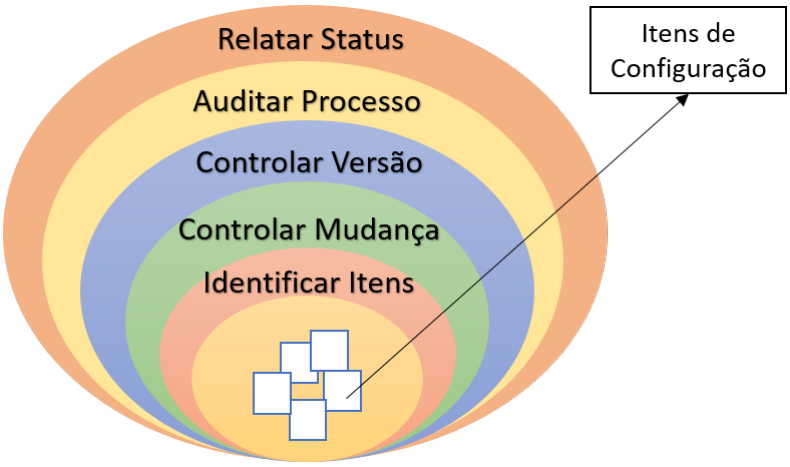
Nesta seção serão apresentados com mais detalhes os conceitos e definições para o entendimento deste trabalho. Na Seção 3.1 serão abordados os principais conceitos sobre Gerência de configuração. Na seção 3.2 serão abordados os principais conceitos sobre integração contínua, com visão mais detalhada sobre o sistema de controle de versão Git, seu repositório remoto Github, e o sistema de integração contínua TravisCI. Na seção 3.3 serão abordados os conceitos do TravisTorrent.

3.1 Gerência de Configuração

A Gerência de Configuração(GC) é um conjunto de atividades que visam a integração dos componentes de um sistema, disponibilizando aos desenvolvedores o acesso ao código fonte, documentos do projeto, e históricos de alterações realizadas. (SOMMERVILLE, 2011).

A Gerência de Configuração possui 5 fases (PRESSMAN; MAXIM, 2016): identificação dos itens de configuração, controle de mudança, controle de versão, auditoria de configuração e relato de status. A Figura 1 ilustra como é a disposição dessas fases.

Figura 1 – Fases da Gerência da Configuração



Fonte – (PRESSMAN; MAXIM, 2016)

Na Fase de identificação dos itens de configuração devem ser definidos quais itens de configuração deverão ser controlados e gerenciados.

Na fase de Controle de Versão é definido o conjunto de ferramentas de controle de versão e os mecanismos para gerenciar e armazenar as versões dos itens de configuração durante o processo de desenvolvimento de software, auxiliando na construção de novas versões para

o software. Dentre as sistemas de controle de versão mais conhecidos estão o Subversion[1] e Mercurial[2].

A fase de Controle de Mudanças é realizada pela combinação de elementos humanos e ferramentas automatizadas. Esta fase se dá pela identificação e solicitação de uma mudança, avaliação da solicitação da mudança pelos responsáveis do projeto e realização da mudança proposta e avaliada. Dentre alguns sistemas de controle de mudanças mais conhecidos estão Redmine[3] e Jira[4].

Na fase de Auditoria é feita a revisão do processo e dos itens de configuração, avaliando se tudo está sendo executado como deveria. A auditoria é a busca de defeitos visando melhorias no gerenciamento de configuração.

Na fase de Relato de status é feita a comunicação das mudanças a todos do projeto. É informado o que mudou, quem fez a mudança, qual impacto e quando foi modificado.

A Gerência de Configuração é muito importante no desenvolvimento de software, pois todo projeto de software está sujeito a mudanças, que é um fator de risco em projetos de software (PRESSMAN; MAXIM, 2016). Uma das práticas utilizadas dentro da Gerência de configuração para a redução de risco ao implementar mudanças é a integração contínua (DUVALL; MATYAS; GLOVER, 2007), que será debatida mais detalhadamente na próxima seção.

3.2 Integração Contínua

Segundo FOWLER; FOEMMEL (2006) a Integração Contínua é uma prática em que os membros da equipe integram seus trabalhos frequentemente, e para cada integração é realizada um conjunto de verificações automáticas, chamadas de compilações automatizadas, a fim de encontrar erros de integração mais rapidamente, e assim minimizar problemas de integração e maximizar a coesão do software produzido.

Para a obtenção de uma prática de IC bem ajustada, são necessários alguns elementos (SEPPALA, 2010):

- **Disciplina dos desenvolvedores**: o desenvolvedor é responsável pela conclusão de sua tarefa, realização de uma compilação local e se bem sucedida, submissão das alterações

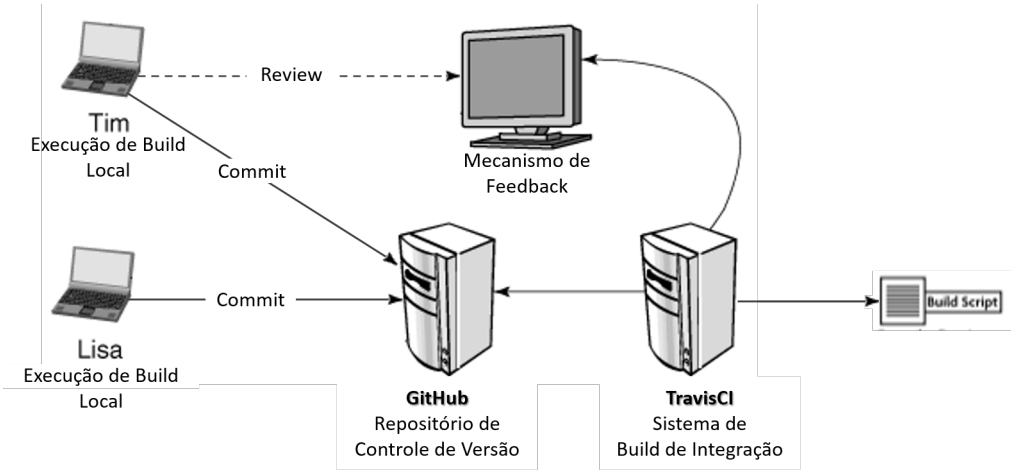
[1] https://www.subversion.apache.org/
[2] https://www.mercurial-scm.org/
[3] http://www.redmine.org/
[4] https://www.atlassian.com/software/jira

para o repositório.

- **Sistema de controle de versão**: é utilizado como repositório para documentos e códigos fonte, mantendo também os históricos das versões e das alterações de cada conteúdo armazenado.
- **Script de build**: é a definição de como se dará a compilação automatizada de integração, com regras de compilação, testes, inspeção e implantação do software.
- **Sistema de Integração Contínua**: é no sistema de integração contínua que são feitas as compilações automatizadas de integração após a submissão de alguma alteração no repositório.
- **Mecanismo de feedback**: o sistema de feedback deverá ser utilizado para informar os participantes do projeto qual o resultado das compilações automatizadas de integração.

Para uma boa execução da prática de IC dentro de um projeto é necessário que esta deva ser bem entendida por todos os integrantes. Para isso é necessário um processo de integração contínua (DUVALL; MATYAS; GLOVER, 2007). A seguir a figura 2 apresenta como os elementos se relacionam.

Figura 2 – Processo de integração contínua genérica



Fonte – (DUVALL; MATYAS; GLOVER, 2007)

O processo de integração contínua genérico se dá em 5 passos. (1) Os desenvolvedores obtém uma cópia do projeto a partir do repositório de controle de versão. (2) As alterações são realizadas e os desenvolvedores realizarão uma compilação local sendo necessário uma compilação bem sucedida para prosseguir para o passo seguinte. (3) É realizado o commit das alterações no repositório de controle de versão. (4) Após a submissão das alterações no repositório, sistema de integração contínua seguindo as regras contidas no script de

build, verifica as alterações e executa um build de integração. (5) O desenvolvedor receberá o resultado da execução do build através do mecanismo de feedback. Caso o build não seja bem sucedido, o responsável deverá corrigir o código fonte, e o processo retornará para o passo 2, se o build for bem sucedido então o processo será concluído (DUVALL; MATYAS; GLOVER, 2007). Este processo pode ser adaptado de acordo com a necessidade do projeto, atualmente as duas variações mais difundidas deste processo é integração contínua manual e a integração contínua automatizada, que serão debatidos a seguir.

Na integração contínua manual apenas um desenvolvedor pode realizar check-ins no repositório durante um intervalo de tempo, ou seja, o desenvolvedor tem a responsabilidade de realizar os builds de integração, e portanto neste momento nenhum outro desenvolvedor poderá realizar um build de integração. Desta forma o processo de integração contínua manual garante maior confiabilidade nas integrações, maior consistência dos itens do repositório, e manutenção da estrutura do repositório consistente (MENEZES, 2011). Diferentemente da integração contínua manual, a integração contínua automática através do uso de uma máquina de integração contínua, possibilita vários check-ins simultâneos no repositório, gerando vários builds de integração ao mesmo tempo. A responsabilidade de execução dos builds é do sistema de integração contínua que gera um build privado para cada check-in realizado no repositório (MENEZES, 2011). Um passo opcional para integração contínua automatizada é a implantação, sendo realizada através de um provedor de implantação contínua e um sistema de controle de versão para registrar a nova versão que será implantada(FOWLER; FOEMMEL, 2006). Um dos sistemas de controle de versões mais conhecidos é o Git que iremos discutir detalhadamente a seguir.

3.2.1 Sistema de controle de versão: Git e GitHub

O git é um sistema de controle de versão criado em 2005, compatível com diversos sistemas operacionais e linguagens de programação, que opera como um pequeno sistema de arquivos, gerenciando o armazenamento e modificações dos arquivos do projeto de software (BELMONT, 2018). Os projetos que utilizam Git serão chamado neste trabalho de projetos Git. O Git possui algumas características essenciais (SWICEGOOD, 2008):

- **Repositório remoto**: Repositório remoto é onde ficam armazenados os diretórios, arquivos, históricos de modificações e versões do projeto Git.
- **Repositório Local**: é onde o desenvolvedor possui uma cópia do projeto Git.

- **Commit**: é o pacote de modificações realizadas pelo desenvolvedor. As seqüências de mudanças são representadas como uma série de commits.
- **Branch**: são versões alternativas criadas a partir da versão do projeto Git, permitindo que a equipe trabalhe em várias modificações ao mesmo tempo.
- **Diff**: é a ferramenta de comparação entre versões do projeto Git, arquivos ou diretórios. O diff irá descrever visualmente o que foi adicionado, removido e alterado entre versões do mesmo projeto Git.
- **Merge**: é a mesclagem das alterações de uma branch em outra.

A maioria dos projetos Git utilizam o Github como repositório remoto (GITHUB, 2015). O Github será debatido com mais detalhes a seguir.

3.2.1.1 Repositório Remoto do Git: GitHub

O Github atualmente possui mais de 30 milhões de usuários ativos, contribuindo em mais de 100 milhões de projetos Git hospedados. O Github fornece ferramentas e serviços para rastreamento de problemas, revisão de código entre outros. Os projetos mais conhecidos que utilizam Git e Github são o WordPress[5], GNU/Linux[6] e Atom[7] (CHACON; STRAUB, 2014).

Além das caracteristicas do Git, o Github contém 3 estágios principais, são eles: Pull, revisão e Deploy (GITHUB, 2015). O fluxo de trabalho do Github trabalha integrado com o controle de versão do Git. A figura 3 a seguir mostra graficamente como funciona o fluxo básico de trabalho Github.

Figura 3 – fluxo básico de trabalho do git com Github

Fonte – (GITHUB, 2015)

[5] https://www.wordpress.com/
[6] https://www.linux.org/
[7] https://www.atom.io/

O fluxo de trabalho do Github integrado com o Git funciona da seguinte maneira: inicialmente o desenvolvedor cria uma *branch* a partir da versão principal do projeto Git e copia o projeto git do repositório remoto para o repositório local. Em seguida realiza as mudanças através dos *commits*. Após a realização de todas as mudanças, então, o desenvolvedor realizará o *pull*, que é o envio dos *commits* com as modificações para o repositório remoto. Em seguida os *commits* serão revisados pelo desenvolvedor ou por uma equipe de revisão com o auxilio da ferramenta *Diff*. Se aprovados, será realizado o *deploy* que é a implantação das modificações para a realização de testes. Caso não haja conflitos então as modificações serão mescladas com o repositório remoto (CHACON; STRAUB, 2014).

O Github também dispõe de um marketplace onde são disponibilizadas ferramentas que podem se integrar ao github, agregando mais recursos a plataforma e auxiliando na construção, segurança, qualidade e testes dos projetos Git. Alguns dos serviços que podem ser integrados ao Github são: Codacy[8], Slack[9] e TravisCI[10] (GITHUB, 2015). O TravisCI será debatido mais detalhadamente a seguir.

3.2.2 Sistema de integração Contínua: TravisCI

O TravisCI é um serviço de construção de software que auxilia no processo de desenvolvimento, construindo e testando automaticamente as modificações do código, com feedback rápido sobre o sucesso da modificação. O travisCI é compatível com 34 linguagens de programação, projetos de software de código fechado ou aberto, e GitHub (BELMONT, 2018).

3.2.2.1 Processo de construção de software

O TravisCI fornece um ambiente de construção padrão e um conjunto padrão de fases para cada linguagem de programação. As configurações das regras e do processo de construção de software devem ser realizadas no arquivo travis.yml. Logo a construção de Software se dá na execução de uma compilação seguindo as regras inseridas no arquivo travis.yml (COUTERMARSH, 2014).

O processo de construção de software se dá em 3 passos: (1) Ao identificar alguma modificação no código fonte do projeto, o travisCI inicia uma compilação, (2) o TravisCI clona o repositório do projeto no GitHub em um ambiente virtual e executa uma série de tarefas,

[8] https://www.codacy.com/
[9] https://www.slack.com/
[10] https://www.travis-ci.org/

previamente configuradas no arquivo travis.yml, para criar e testar o código, (3) são realizados os testes do código fonte, com quatro tipos de resultados possíveis: erro, falha, cancelado e aprovado. Se houver erros nas configurações inseridas no arquivo travis.yml, o resultado será de erro, neste caso a compilação é interrompida imediatamente. Se uma ou mais tarefas falhar, o resultado será de falha. Neste caso a compilação poderá ser interrompida ou não, dependendo das configurações inseridas no arquivo travis.yml. Se o usuário cancela a compilação antes de concluir, o resultado será de cancelado. Se nenhuma das tarefas falhar, o resultado será de aprovado (TRAVISCI, 2013).

O TravisCI pode automatizar outras partes do fluxo de trabalho de um projeto. Podem ser automatizados os serviços de implantação do código em um servidor da Web ou host de aplicativo, notificações de configuração, preparação de implementações após compilações, entre outras tarefas.

3.3 TravisTorrent

O TravisTorrent é um projeto que através do uso da API do GitHub e TravisCI disponibiliza um grande conjunto de dados de projetos abertos. Contendo dados de log de compilação, repositório dos projetos e seus commits, testes e builds, atualmente o TravisTorrent está em fase de protótipo analisando continuamente projetos desenvolvidos nas linguagens Java, Javascript e Ruby (BELLER; GOUSIOS; ZAIDMAN, 2017).

O objetivo do projeto é a partir da uniformidade e combinação dos dados gerados pelo travisCI e GitHub, disponibilizar uma base de dados consistente para aplicação da mineração de dados a fim de extrair informações relevantes sobre o impacto de se fazer IC na prática de engenharia de software (BELLER; GOUSIOS; ZAIDMAN, 2017).

O projeto contém dados de 1283 projetos de código aberto, sendo 886 na linguagem Ruby e 393 na linguagem Java e 4 na linguagem Javascript. Os dados estão organizados em uma tabela de 2,87 Gigabytes com 3,702,595 linhas, em que cada linha representa uma execução de *build*. Neste trabalho o TravisTorrent foi utilizado para a coleta dos dados de projetos Java.

4 PROCEDIMENTOS METODOLÓGICOS

Os procedimentos metodológicos deste trabalho foram realizados em quatro etapas: (i) Definição de questões de pesquisa, (ii) investigação inicial e refinamento das questões de pesquisa, (iii) coleta dos dados e (iv) Análise dos dados coletados. As etapas e os passos serão discutidos detalhadamente a seguir.

4.1 Definição das questões de pesquisa

Esta etapa teve como objetivo definir as questões de pesquisa do trabalho levando em consideração os objetivos deste trabalho. Para os estudos foram definidos duas perspectivas: uma de projeto Java e outra de desenvolvedores brasileiros. As seguintes questões de pesquisa foram levantadas inicialmente:

- Perspectiva de Projeto Java.

  Quantos projetos Java foram impactados por brasileiros?
- Perspectiva de desenvolvedores brasileiros:

  Qual a proporção de brasileiros em relação ao mundo?

  Quantos builds falhos e bem-sucedidos tiveram participação de brasileiros?

  Quantos builds falhos e bem-sucedidos tiveram responsabilidade de brasileiros?

Após a definição das questões de pesquisa concluída, foi iniciada a etapa de investigação da base de dados e refinamento das questões de pesquisa.

4.2 Investigação da base de dados e refinamento das questões de pesquisa

Esta etapa teve o objetivo de investigar a base de dados do TravisTorrent com o propósito de conhecer os dados contidos no banco de dados e investigar a viabilidade de algumas questões de pesquisa. Esta etapa foi importante, pois serviu para conhecer a base de dados e identificar quais estudos era possível realizar a partir da base de dados disponibilizado pelo TravisTorrent. Esta etapa se deu em 2 passos: Investigação da base de dados do TravisTorrent; E refinamento das questões de pesquisa. A seguir os passos desta etapa serão discutidos detalhadamente.

4.2.1 Investigação da base de dados do TravisTorrent

Este passo se deu pela investigação da base de dados do TravisTorrent, com a finalidade de conhecer melhor os dados disponibilizados. Este passo auxiliou na identificação de quais dados seriam seriam coletados para responder as questões de pesquisa, e no estudo de viabilidade das questões de pesquisa. Para a investigação foi utilizada a versão online da base de dados do TravisTorrent que é disponibilizada na plataforma BigQuery[11] da Google. A investigação foi realizada com foco nas questões de pesquisa definidas.

Após o passo de investigação dos dados contidos na base de dados do TravisTorrent foi iniciado o passo de refinamento das questões de pesquisa.

4.2.2 Refinamento das questões de pesquisa

Nesta etapa foi realizado o refinamento das questões de pesquisa após o estudo da base de dados do TravisTorrent. As questões de pesquisas definidas neste passo foram utilizadas como referência para guiar as próximas etapas do procedimento metodológico realizados neste trabalho.

Através do estudo de viabilidade foi verificado que não seria possível estudar os builds bem sucedidos devido ao tamanho muito grande da base de dados, e o tempo para a conclusão deste trabalho. Contudo a partir da exclusão dos builds bem sucedidos foi possível a realização da análise mais detalhada dos builds falhos, com isso foi realizados os estudos sob a perspectiva de projetos Java com builds falhos.

Para os estudos sobre projetos Java com builds falhos foram realizados os seguintes refinamentos:

- QP1:Quantos projetos Java foram impactados com builds falhos com contribuição brasileiros?
- QP2: Qual o nível de engajamento dos desenvolvedores estrangeiros e brasileiros em seus projetos?
- QP3: Qual a contribuição de desenvolvedores brasileiros e estrangeiros em builds falhos?
- QP4: Quantos commits são necessários para o desenvolvedor brasileiros produzir um build falho?

Para o estudos sobre a contribuição de desenvolvedores brasileiros em projetos Java

[11] https://cloud.google.com/bigquery/

com builds falhos foram realizados os seguintes refinamentos:

- QP5:Qual a proporção de brasileiros em relação ao mundo?
- QP6: Quantos builds falhos tiveram participação de brasileiros?
- QP7: Quantos builds falhos tiveram responsabilidade de brasileiros?

Importante salientar a diferença de brasileiro participante de um build falho e brasileiro responsável por um build falho. O brasileiro participante de build falho é o brasileiro que é autor de um commit que não iniciou o build falho, mas compõe o conjunto de commits participantes do build falho. O brasileiro responsável por um build falho é o brasileiro que é autor de um commit que é integrante único de build falho ou é autor do commit que iniciou a execução de um build falho. Ao concluir esta etapa foi iniciada a etapa de coleta dos dados.

4.3 Coleta dos dados

Esta etapa tem o objetivo de coletar os dados relevantes e pertinentes ao trabalho. Os dados coletados nesta etapa foram usados como insumo para responder as questões de pesquisa levantadas neste trabalho. A execução desta etapa se deu em 3 passos: (1) coleta dos projetos, builds e commits, (2) coleta dos participantes dos projetos, (3) coleta dos autores dos commits. A seguir os passos desta etapa serão discutidos detalhadamente.

4.3.1 Coleta dos projetos, builds e commits

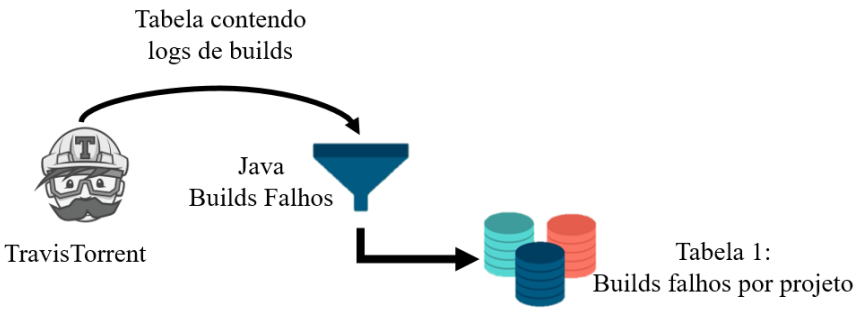
Neste passo foram realizadas consultas no banco de dados a partir de critérios de seleção a fim de obter apenas os dados relevantes para este trabalho. Os critérios de seleção utilizados foram: (I) projetos desenvolvidos na linguagem Java e (II) projetos que contenham algum build falho. Os resultados das consultas sobre projetos, builds e commits foram armazenados em uma tabela que chamamos de "Builds falhos por projeto".

Após a realização deste passo, os dados sobre projetos, builds e commits foram coletados e foi iniciado o passo de coleta dos participantes dos projetos. A seguir a figura 4 mostra graficamente o processo utilizado neste passo.

4.3.2 Coleta dos participantes dos projetos

Este passo se seu na obtenção de dados sobre os participantes de cada projeto. Os dados foram coletados a partir de um script desenvolvido pelo próprio autor na linguagem Python,

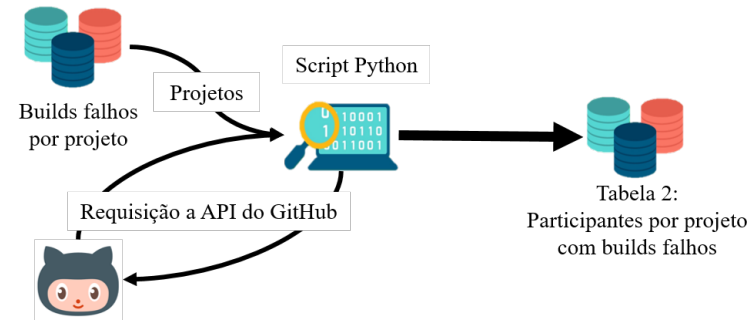
Figura 4 – Processo de coleta dos projetos, builds e commits



Fonte – Próprio Autor

que realizava requisições à API do GitHub. O script utilizava o dado "nome do projeto"contido na tabela 1, e recebia como resultado da requisição à API do Github, o link do perfil de cada participante do projeto. Por fim, o script criou uma nova tabela chamada de "Participantes por projeto com build falho", contendo o nome do projeto, link do perfil de cada participante do projeto e a quantidade de participantes em cada projeto. A figura 5 mostra como se deu a coleta dos participantes dos projetos.

Figura 5 – Processo de coleta dos participantes dos projetos



Fonte – Próprio Autor

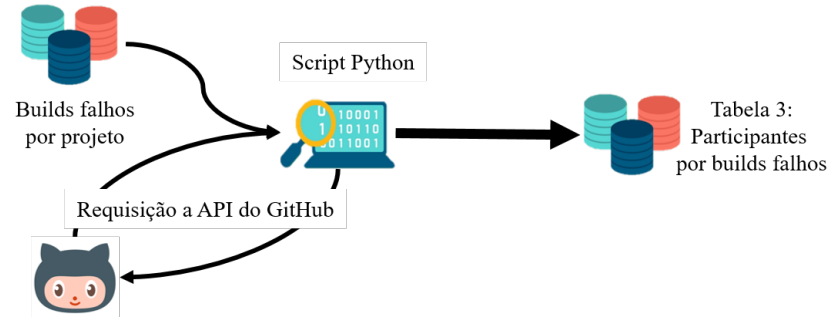
Após a conclusão deste passo, iniciou o passo de coleta dos autores dos commits.

4.3.3 Coleta dos autores dos commits

Este passo se deu na obtenção dos autores de cada commit coletado na etapa de coleta de projetos, builds e commits. Os dados foram coletados através de um script desenvolvido pelo próprio autor na linguagem Python. O script utilizava a chave de cada commit participante do build falho contido na tabela Builds falhos por projeto para realizar uma requisição para a API do GitHub, que retornava os dados referentes ao commit. A partir da obtenção dos dados dos commits, o script coletou o login ou nome do autor do commit. Em seguida o script armazenou

os dados coletados em uma nova tabela que chamamos de "Participantes por builds falhos", contendo o nome dos projetos, suas builds e commits, e o nome ou login do autor de cada commit. A figura 6 mostra o processo de coleta dos autores dos commits.

Figura 6 – Processo de coleta dos autores dos commits



Fonte – Próprio Autor

Após a conclusão deste passo foi concluída a etapa de coleta dos dados e iniciou-se a etapa de investigação dos dados coletados.

4.4 Análises dos dados coletados

Esta etapa tem o objetivo de realizar análises sobre os dados coletados na etapa anterior, visando responder as questões de pesquisas levantadas neste trabalho. Nesta etapa foram analisados os dados coletados com a finalidade de identificar usuários brasileiros, projetos com participantes brasileiros, builds falhos com commits brasileiros e o impacto de usuários brasileiros em builds falhos. Esta etapa se deu em 2 passos: (I) identificação dos brasileiros e seus projetos e (II) identificação dos builds com commits brasileiros. Os passos desta etapa serão discutidos detalhadamente a seguir.

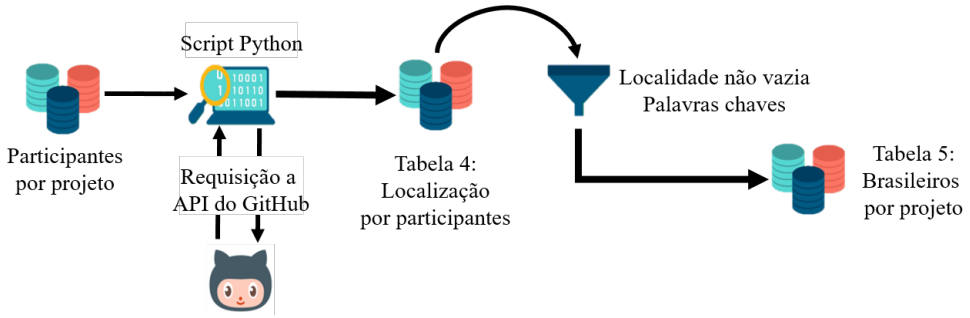
4.4.1 Identificação dos brasileiros e seus projetos

Este passo tem o objetivo de identificar os brasileiros e os projetos contendo brasileiros. Para a realização deste passo foi utilizado um script desenvolvido pelo próprio autor na linguagem Python, e a tabela participantes por projeto contendo informações sobre projetos Java e seus participantes. O script utilizava o link do perfil de cada participante dos projetos para realizar requisições à API do GitHub que retornava todas as informações públicas do participante. O script coletou o conteúdo do dado de localização do participante chamada de

location. Em seguida o conteúdo foi armazenado numa nova tabela chamamos de "Localização por participantes", que armazenava o nome do projeto, o nome e login do autor e o conteúdo do dado *location*. Caso o conteúdo do dado *location* fosse vazio ou inválido o conteúdo do dado *location* seria preenchido como *None*.

Para a identificação dos brasileiros foi realizada uma filtragem por palavras chaves e exclusão das localizações com o conteúdo *None*. As palavras chaves selecionadas para a filtragem foram os nomes dos estados brasileiros com e sem acentuação e suas respectivas abreviaturas, e as palavras Brasil e Brazil e sua abreviatura. A lista de palavras utilizadas neste passo está disponível no Apêndice A deste trabalho. Em alguns casos a abreviatura de algum estado coincidiu com abreviaturas de cidades e estados que não eram brasileiros ou até de outros países. Nestes casos os participantes foram retirados da lista de brasileiros. Os resultados desta filtragem foram armazenados em uma nova tabela que chamamos de "Brasileiros por projeto", que armazenava o nome do projeto e os participantes brasileiros. A figura 7 a seguir mostra o processo realizado neste passo para a coleta do conteúdo do dado *location* dos participantes e a identificação dos brasileiros e seus projetos.

Figura 7 – Processo de identificação dos brasileiros e seus projetos



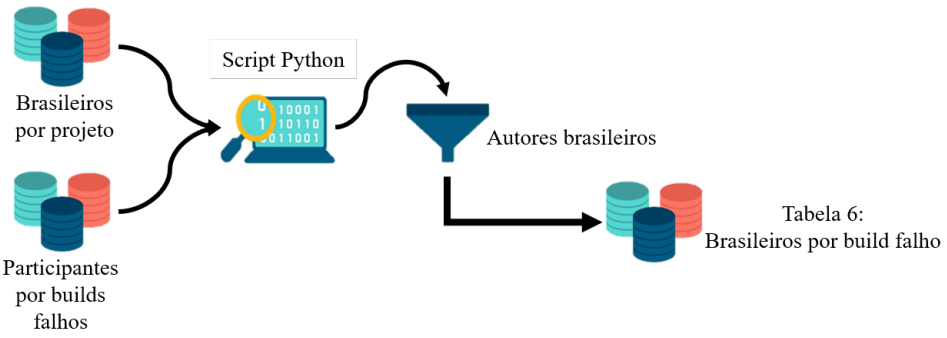
Fonte – Próprio Autor

4.4.2 Identificação dos builds com commits de brasileiros

Este passo tem o objetivo de identificar os builds falhos que contém commit de autor brasileiro. Para a execução deste passo foi utilizado um script desenvolvido pelo próprio autor deste trabalho na linguagem Python, e as informações da tabela 5 e tabela 3. O script utilizou o dado nome e login de brasileiros contidos na tabela 5, e as informações sobre builds falhos contidos na tabela 3. O script comparava o nome e login do autor de cada commit contidos na tabela 3, com o nome e login dos brasileiros contidos na tabela 5. Ao identificar o commit com

autoria de brasileiro, o script armazenava o nome e login do autor do commit, e as informações do build falho em uma nova tabela, chamamos "Brasileiros por build falho". A tabela 6 armazenava apenas as informações referentes aos builds falhos com commits de autoria de brasileiro. A seguir a figura 8 mostra graficamente como se dará o processo que será executado neste passo.

Figura 8 – Processo de identificação de builds com commits de brasileiros



Fonte – Próprio Autor

5 RESULTADOS

Esta seção apresenta os resultados obtidos através da execução dos procedimento metodológicos discutidos anteriormente neste trabalho. Serão discutidos os resultados de acordo com cada etapa do procedimento metodológico.

5.1 Investigação inicial dos dados

Ao investigar o banco de dados disponibilizado pelo projeto TravisTorrent foi identificada uma tabela com 61 colunas e 3.702.595 linhas, na qual cada linha correspondia a uma build executada pelo TravisCI e as colunas correspondiam aos dados obtidos a partir da execução da build.

A partir dos objetivos do trabalho foram definidos que as análises seriam realizadas apenas em projetos java com builds falhos e portanto toda a coleta de dados se deu neste contexto. Os dados foram definidos de acordo com a sua relevância para responder as questões de pesquisa, então os dados definidos foram: nome do projeto (gh_project_name), linguagem de programação dominante no projeto (gh_lang), commits participantes da build (gh_commits_in_push), commit que acionou a build (git_trigger_commit), e resultado final do build (tr_build_sucessful).

Após a definição dos dados relevantes para a pesquisa, iniciou-se a coleta dos dados a fim de obter uma base de dados menor e conveniente com a pesquisa realizada neste trabalho.

5.2 Coleta dos Dados

Esta seção apresenta os resultados da execução da etapa de coleta dos dados, apresentando os dados coletados dos projetos, builds, commits, participantes dos projetos e autores dos commits. A seguir os resultados desta etapa serão discutidos detalhadamente.

5.2.1 Coleta de Projetos, Builds e commits

Para coletar os dados sobre projetos, builds e commits, foram realizadas 2 consultas no banco de dados do TravisTorrent, a primeira foi realizada com o objetivo de coletar os projetos java, e a segunda foi realizada para coletar todos os builds falhos dos projetos java coletados na primeira consulta.

Na primeira consulta realizada foram identificados 1.283 projetos no TravisTorrent,

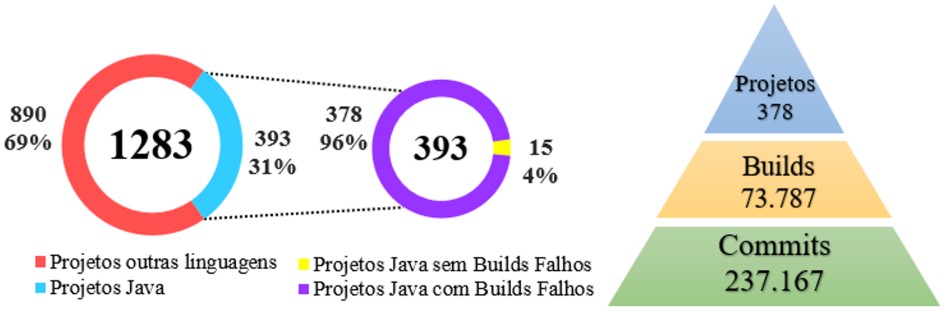
destes 1.283 projetos, 393 eram desenvolvidos em java e os outros 890 desenvolvidos em outras linguagens. Como o foco deste trabalho são os projetos java com builds falhos, então foi realizada uma filtragem a fim de identificar quais projetos java possuíam builds com falhas, ao final foram coletados 378 projetos java com pelo menos um build falho.

A segunda consulta foi realizada para coletar todos os builds falhos dos 378 projetos java com build falho identificados, nesta consulta foram selecionados 73.787 builds com um total de 237.167 commits participantes.

O resultado da segunda consulta foi armazenado em uma tabela no formato .csv chamada de "Builds falhos por projeto", esta tabela possui 73.787 linhas, em que cada linha representa uma build, e 4 colunas com os dados: nome do projeto, commits participantes de cada build, commit que acionou a build e resultado final da execução da build.

Ao gerar a tabela builds falhos por projeto percebeu-se que o dado commits participantes deveria ser tratado, já que estava disponibilizada em forma de uma string com as chaves de todos os commits participantes da build separado pelo caractere '#' causando dificuldade de entendimento do conteúdo deste dado. O tratamento se deu na separação de cada commit por vírgula para melhor identificação. A figura 9 mostra graficamente a síntese dos resultados obtidos nesta etapa.

Figura 9 – Proporção de Projetos java com e sem builds e quantidade de projetos, builds e commits



Fonte – Próprio Autor

Após a coleta dos dados sobre projetos, builds e commits, iniciou-se a etapa de coleta dos participantes dos 378 projetos Java identificados.

5.2.2 Coleta dos participantes dos projetos e dos autores dos commits

Nesta etapa foram coletados os participantes de cada um dos 378 projetos java. Ao executar esta etapa foram coletados 15.789 participantes. As informações sobre os participantes de cada projeto foram armazenados em uma tabela chamada "participantes por projeto com builds falhos", na qual possuía informações sobre nome do projeto, participantes do projeto e numero de commits dos projetos. A figura 10 a seguir mostra como os dados estavam dispostos na tabela Participantes por projeto.

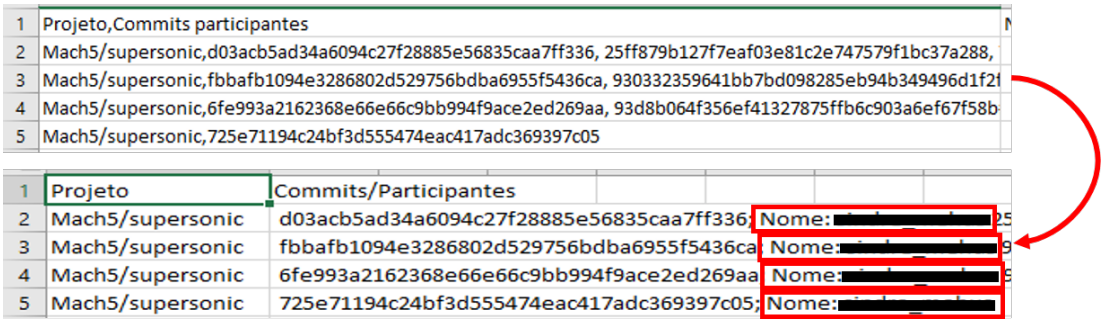
Figura 10 – Tabela Participantes por projeto

	Projeto	Participantes	Quantidade de Commits
2	cbeust/testng	https://api.github.com/users/MicahLC	4543
3	dreamhead/moco	https://api.github.com/users/caimaoy	2345
4	Netflix/Hystrix	https://api.github.com/users/timbozo	2106
5	stanfy/enroscar	https://api.github.com/users/roman-n	1486
6	Mach5/supersonic	https://api.github.com/users/timorein	429
7	puniverse/quasar	https://api.github.com/users/pron htt	2459

Fonte – Próprio Autor

Na etapa de coleta dos autores dos commits foram analisados os 237.167 commits participantes dos builds falhos, para cada commit foi coletado o login e nome do autor. Os novos dados foram armazenados em uma nova tabela chamada "participantes por builds falhos", de forma que após a chave de cada commit foi inserido o nome ou login do seu autor. A figura 11 a seguir mostra a diferença da tabela builds falhos por projeto e a tabela participantes por builds falhos.

Figura 11 – tabela builds falhos por projeto e a tabela participantes por builds falhos.



Fonte – Próprio Autor

5.3 Análises dos Dados coletados

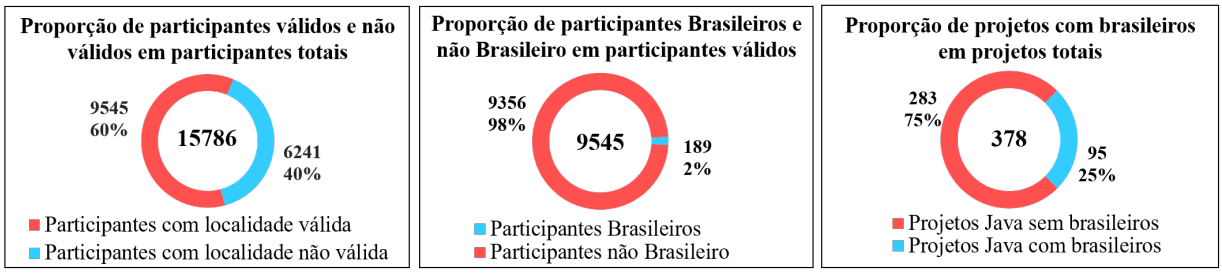
Esta Seção apresenta as análises sobre os dados coletados nas etapas anteriores, respondendo as questões de pesquisas levantadas neste trabalho. Os resultados e análises serão discutidos detalhadamente a seguir.

5.3.1 Identificação dos brasileiros e seus projetos

Nesta etapa foram executados os passos de identificação dos brasileiros a partir dos 15.786 participantes em projetos java com builds falhos. Ao executar esta etapa foi detectado que uma parte dos participantes não havia preenchido o campo 'location' que era primordial para a identificação da nacionalidade de cada participante. Os participantes que não preencheram o campo 'location' foram marcados com o valor 'None', estes participantes chamaremos de participantes não válidos, os demais participantes, ou seja aqueles que preencheram o campo 'location', foram selecionados para a realização das análises e chamaremos de participantes válidos.

Nos participantes válidos foi realizada uma filtragem a fim de identificar quais dos participantes eram brasileiros. Foi utilizada a ferramenta excel para a filtragem dos participantes válidos. Foram identificados 9.545 participantes que preencheram o dado 'location' e 6.241 participantes que não preencheram o dado 'location' e foram marcados com o valor 'None'. Dos 9.545 participantes válidos 189 foram identificados como brasileiros e 9.356 foram identificados como não brasileiros. Foi identificado que os 189 brasileiros estavam distribuídos em 95 projetos java com builds falhos, e que estes 95 projetos possuíam um total de 27.000 builds falhos com 63.089 commits. A figura 12 mostra graficamente a síntese dos resultados obtidos nesta etapa.

Figura 12 – Síntese dos resultados obtidos na etapa de Identificação dos brasileiros e seus projetos.



Fonte – Próprio Autor

5.3.2 Identificação dos builds com commits de brasileiros

Neste passo foram identificados os builds falhos com commits brasileiros. Ao executar este passo foram identificados 631 commits brasileiros. Os brasileiros contribuíram com 85 commits responsáveis por builds falhos e 546 commits participantes de builds falhos. Foram identificados 87 builds falhos com commits de brasileiros. Os brasileiros foram responsáveis por 76 builds falhos, e participantes de 11 builds falhos.

Após a identificação dos builds falhos com commits brasileiros, foram identificados 24 brasileiros que participaram ou foram responsáveis de builds falhos em seus projetos, sendo 3 participantes de algum build falho, 19 responsáveis por algum build falho e 2 foram participante e responsável por algum build falho. Os brasileiros contribuíram em 21 Projetos. Os brasileiros foram responsáveis por builds falhos em 16 projetos, participantes de builds falhos em 3 projetos, em 2 projetos houve brasileiros que foram participantes e responsáveis por builds falhos. A figura 14 a seguir mostra os dados obtidos dos projetos java com brasileiros participantes de builds falhos.

Figura 13 – Resumo projetos java com brasileiros participantes de builds falho.

Projeto	Brasileiros	BF. com brasileiros	Qtd. BF. com responsabilidade de brasileiros	Qtd. BF. com participação de brasileiros	BF. Totais	% BF. Brasileiros por BF. total	LOC
jmxtrans/jmxtrans	1	1	1	0	77	1,298%	9.872
timmolter/XChange	1	1	1	0	146	0,684%	88.879
SonarSource/sonarqube	1	1	1	0	3556	0,028%	224.469
DSpace/DSpace	1	1	1	0	145	0,689%	219.033
satyan/sugar	1	29	29	0	46	63,043%	2.384
caelum/vraptor4	4*	9	9	0	146	6,164%	12.187
psi-probe/psi-probe	1	1	1	0	14	7,142%	14.426
caelum/mamute	2	3	3	0	23	13,043%	15.513
xetorthio/jedis	1*	7	6	1	445	1,573%	18.016
jcabi/jcabi-http	1	2	2	0	157	1,273%	3.369
MrTJP/ProjectRed	1	1	1	0	234	0,427%	23.486
geoserver/geoserver	1	2	2	0	3142	0,063%	335.090
grails/grails-core	2	3	3	0	534	0,561%	72.416
querydsl/querydsl	1	1	1	0	748	0,133%	51.652
yegor256/rultor	1**	1	1	0	312	0,320%	20.997
gwtbootstrap/gwt-bootstrap	1**	12	12	0	103	10,619%	15.356
sparklemotion/nokogiri	1	3	1	2	1877	0,159%	12.380
caelum/vraptor	1	1	1	0	7	14,285%	14.498
thinkaurelius/titan	1	1	0	1	397	0,251%	28.285
gradle/gradle	1	6	0	6	299	2,006%	209.562
dynjs/dynjs	1	1	0	1	67	1,492%	34.048

(*) O Participante "*"em projetos diferentes
(**) O Participante "**"em projetos diferentes

Fonte – Próprio Autor

5.3.3 Resposta as questões de pesquisa

Esta etapa tem o objetivo de sintetizar os resultados obtidos anteriormente com a finalidade de responder as questões de pesquisa definidas neste trabalhos. Em algumas questões de pesquisa será realizado uma breve explicação sobre os cálculos para a obtenção dos resultados.

Para as questões de pesquisa sobre projetos Java com builds falhos foram encontradas as seguintes respostas:

- QP1: Quantos projetos Java foram impactados com builds falhos com contribuição brasileiros?

  Para responder a QP1 foram coletados 95 projetos java com brasileiros, porém apenas 21 projetos tiveram brasileiros participantes de builds falhos, ou seja em 74 projetos não foram encontrados brasileiros em builds falhos. Os 74 projetos que não tiveram brasileiros participantes de builds falhos foram considerados nos cálculos para desenvolvedores estrangeiros e desconsiderados nos cálculos para desenvolvedores brasileiros. Então os brasileiros contribuíram com builds falhos em 21 projetos Java
- QP2: Qual o nível de engajamento dos desenvolvedores estrangeiros e brasileiros em seus projetos?

  Para responder a QP2 foi utilizada a métrica chamada de taxa de commit por projeto, que calcula a qantidade de commits realizados por projeto, ou seja quanto maior essa taxa, maior quantidade de modificações que foram realizadas pelos desenvolvedores em seus projetos.

  Para os desenvolvedores estrangeiros foram coletados 213.486 commits participantes de builds falhos em 357 projetos java. Então os desenvolvedores estrangeiros realizam em média 598 commits contribuinte de builds falhos por projeto. Para os desenvolvedores brasileiros foram utilizados apenas os commits dos projetos Java que haviam brasileiros contribuindo com builds falhos, ou seja foram utilizados os commits e builds falhos de 21 projetos. O cálculo foi realizado pela quantidade de commits brasileiros contribuinte de builds falhos por quantidade de projetos com brasileiros contribuinte de builds falhos. Então foram encontrados 631 commits brasileiros em builds falhos em 21 projetos. Portanto os desenvolvedores brasileiros realizam uma média de 30 commits contribuintes de builds falhos por projeto.
- QP3: Qual a contribuição de desenvolvedores brasileiros e estrangeiros em builds falhos?

Para responder a QP3 foi utilizada uma métrica chamada de taxa de builds falhos por projeto, que calcula a contribuição de desenvolvedores em builds falhos nos seus projetos, ou seja, quanto maior essa taxa, maior a contribuição dos desenvolvedores em builds falhos em seus projetos.

Para os desenvolvedores estrangeiros foram encontrados 61.312 builds falhos em 357 projetos java. Então os desenvolvedores estrangeiros produzem em média 171 builds falhos por projeto. Para os desenvolvedores brasileiros as análises foram realizadas através dos resultados dos 21 projetos Java que tiveram brasileiros participantes de builds falhos. Neste caso foram encontrados 87 builds falhos com brasileiros em 21 projetos Java com participantes brasileiros. Então os desenvolvedores brasileiros produzem em média 4 builds falhos por projeto.

- QP4: Quantos commits são necessários para o desenvolvedor brasileiros produzir um build falho?

  Para responder a QP4 foi utilizada uma métrica chamada de taxa de commits por build falho, que calcula a quantidade de commits realizados por builds falhos.

  Para os desenvolvedores estrangeiros foram coletados 213.486 commits estrangeiros em 61.312 builds falhos. Então os desenvolvedores estrangeiros produzem em média 3,4 commits por build falho. Para os brasileiros foram encontrados 631 commits brasileiros em 87 builds falhos com participantes brasileiros. Então os desenvolvedores brasileiros produzem em média 7 commits por build falho.

Para as questões de pesquisa sobre a contribuição de desenvolvedores brasileiros em projetos Java com builds falhos foram encontradas as seguintes respostas:

- QP5:Qual a proporção de brasileiros em relação ao mundo?

  Para responder a QP5 foram considerados apenas os participantes de projetos com localização válida. Então foram identificados 9545 participantes válidos, sendo que 189 participantes eram brasileiros. O brasileiros representam 2% dos desenvolvedores globais.

- QP6: Quantos builds falhos tiveram participação de brasileiros?

  Para responder a QP6 foram considerados commits brasileiros que não eram participates únicos de builds falhos e commits brasileiros que não iniciaram um build falhos. Foram coletados 546 commits brasileiros participantes de 11 builds falhos. Então os brasileiros foram participantes de 11 builds falhos.

- QP7: Quantos builds falhos tiveram responsabilidade de brasileiros?

Para responder a QP6 foram considerados commits brasileiros que eram participates únicos de builds falhos ou commits brasileiros que iniciaram um build falhos. Foram coletados 85 commits brasileiros participantes de 76 builds falhos. Então os brasileiros foram responsáveis por 76 builds falhos.

Ao analisar os resultados obtidos da taxa de builds falhos por projetos foi possível concluir que os desenvolvedores estrangeiros são mais engajados em seus projetos em comparação com desenvolvedores brasileiros. A análise da taxa de commits por projeto mostra que os desenvolvedores estrangeiros contribuem com mais builds falhos em seus projetos do que os desenvolvedores brasileiros. Já a taxa de commits por build falho mostra que os desenvolvedores brasileiros contribuem com mais commits para que possa produzir um build falho.

Ao comparar a taxa de builds por projeto com a taxa de commits por projeto dos desenvolvedores brasileiros e estrangeiros foi identificado que os brasileiros por participarem menos dos seus projetos também contribuem menos com builds falhos, já os estrangeiros é identificado que por participarem mais dos seus projetos produzem mais builds falhos.

Ao comparar a taxa de commits por build falho de desenvolvedores estrangeiros e brasileiros é possível concluir que os brasileiros precisam contribuir com mais commits para produzir um build falho.

6 DISCUSSÃO

Os resultados apresentados neste trabalho mostraram que os brasileiros correspondem a 2% dos participantes válidos globais, este resultado está em conformidade com resultados obtidos em ABES (2019) que mostram que o mercado brasileiro de software corresponde a 2,1% do mercado globais de software.

Foi constatado que a maioria dos builds falhos com brasileiros foram produzidos por responsabilidade de brasileiros. Acredita-se que esses resultados podem ter acontecido por causa que a maioria dos projetos java com brasileiros coletados eram de pequeno porte, com poucos participantes, ou seja projetos não profissionais, aumentando a possibilidade de encontrar mais builds com apenas um commit participante.

Os resultados da comparação dos desenvolvedores estrangeiros e brasileiros, mostram que os brasileiros são menos engajados e possuem menos contribuições em builds falhos. Acredita-se que isso se deve porque a maioria dos projetos com brasileiros são pequenos, e por causa da pequena amostragem de builds falhos e commits de desenvolvedores brasileiros. Outro resultado é que os desenvolvedores brasileiros precisam contribuir com mais commits para produzir um build falho, em comparação com desenvolvedores estrangeiros, esse resultado sugere a qualidade maior do desenvolvedor brasileiro em relação ao desenvolvedor estrangeiro, entretanto o resultado obtido neste trabalho é apenas uma evidência, sendo necessário o estudo de outros fatores para a obtenção de um resultado mais concluso.

7 AMEAÇAS À VALIDADE

Nesta seção serão apresentados alguns fatores que dificultaram os estudos realizados neste trabalho e que possam ter influenciado os resultados obtidos neste trabalho.

As principais ameaças aos resultados deste trabalho serão debatidos a seguir:

- **Commit com autor que deletou seus perfis**: Foram encontrados 26 casos de desenvolvedores que são autores de commits participantes de builds falhos mas que deletaram seus perfis do Github, e portanto não foi possível identificar a nacionalidade. Essa dificuldade foi contornada com a exclusão desses commits e desenvolvedores do estudo.
- **Desenvolvedores que não preencheram a localidade**: 6.241 desenvolvedores não preencheram o dado 'location' e acredita-se que este é uma quantidade relevante de participantes que foram excluídos dos estudos realizados no trabalho. Para contornar esta dificuldade poderá ser realizada uma investigação dos códigos fonte e projetos desenvolvidos por esses desenvolvedores para encontrar evidências que possam comprovar a nacionalidade do desenvolvedor, por exemplo idioma utilizado pelo desenvolvedor, ou desenvolvedores que apenas participam de projetos com brasileiros.
- **A localidade pode não representar a nacionalidade do usuário**: O dado 'location' pode ser entendido como nacionalidade ou local onde o desenvolvedor está no momento, se caracterizando um dado subjetivo. Para contornar esta dificuldade poderá ser realizada uma investigação dos códigos fonte e projetos desenvolvidos por esses desenvolvedores para encontrar evidências que possam comprovar a nacionalidade do desenvolvedor, por exemplo idioma utilizado pelo desenvolvedor, ou desenvolvedores que apenas participam de projetos com brasileiros.

8 CONSIDERAÇÕES FINAIS

Este trabalho realizou o estudo da contribuição de brasileiros em builds falhos com o objetivo principal de identificar o tipo de contribuição dos desenvolvedores brasileiro em seus projeto.

A partir da coleta dos dados sobre commits, builds falhos, projetos Java e desenvolvedores brasileiros foi possível identificar que a proporção de brasileiros corresponde a 2% dos participantes válidos, e que a maioria dos projetos com contribuição de brasileiros são de pequeno porte.

O objetivo principal deste trabalho foi atingido, através da identificação dos tipos de contribuição de brasileiros em builds falhos de projetos Java, foi identificada a quantidade de builds falhos com contribuição e participação de brasileiros. O objetivo de medir a qualidade dos desenvolvedores brasileiros em relação aos desenvolvedores estrangeiros foi parcialmente alcançado, pois os resultados obtidos neste trabalho são apenas uma evidência que os desenvolvedores brasileiros são melhores que desenvolvedores estrangeiros, necessitando de estudos mais profundos para resultados mais conclusos.

Os resultados obtidos neste trabalho podem ter sido influenciados pela baixa amostragem de builds falhos e commits de desenvolvedores brasileiros. Contudo este trabalho contém resultados relevantes sobre tipos de contribuição de brasileiros em builds falhos, quantidade de brasileiros em relação a comunidade global e visão geral dos projetos java com builds falhos brasileiros em relação a projetos globais.

Vale salientar que apesar das dificuldades enfrentadas durante este trabalho, os estudos realizados tem relevância no sentido de fornecer uma metodologia que busca estudar as contribuições de desenvolvedores de uma determinada nacionalidade em builds falhos, podendo servir de referência para a realização de outros estudos semelhantes.

Como trabalhos futuros, pretende-se fazer esta mesma análise para um conjunto maior de projetos, com desenvolvedores de outras nacionalidades, projetos de outras linguagens e builds bem-sucedidos. Além disso, pretende-se fazer análises mais profundas, a fim de encontrar alguma correlação entre os desenvolvedores e os builds falhos levando em consideração apenas projetos de grande porte.

REFERÊNCIAS

ABES. **Investimentos em TI no Brasil crescem 9,8% em 2018**. 2019. ABES,2018Disponível em: http://www.abessoftware.com.br/noticias/investimentos-em-ti-no-brasil-crescem-98-em-2018. Acesso em: 16 de jun. 2019.

BELLER, M.; GOUSIOS, G.; ZAIDMAN, A. **Travistorrent**: Synthesizing travis ci and github for full-stack research on continuous integration. *In*: Proceedings of the 14th working conference on mining software repositories. [S.I]:IEEE, 2017. 447-450 p. isbn:9781538615447.

BELMONT, J.-M. **Hands-On Continuous Integration and Delivery:** Build and release quality software at scale with Jenkins, Travis CI, and CircleCI. 1ª. ed. Birmingham: Packt Publishing, 2018. 416 p. isbn: 9781789130485

BRASSCOM. **Brasil precisa de 750 mil novos profissionais de TI até 2020**. 2015. Disponível em: https://dev-brasscom.primaestudio.com.br/brasil-precisa-de-750-mil-novos-profissionais-de-ti-ate-2020/. Acesso em: 21 de nov. 2018.

CHACON, S.; STRAUB, B. **Pro git**. 2ª. Ed. New York: Apress, 2014. 419 p. isbn:9781430218340

COUTERMARSH, M. **Heroku Cookbook**. 1ª. Ed. Birmingham: Packt Publishing, 2014. 232 p. isbn:9781782177944.

DUVALL, P. M.; MATYAS, S.; GLOVER, A. **Continuous integration:** improving software quality and reducing risk. 1ª. Ed. Londres: Pearson Education, 2007. 336 p. isbn:9780321336385

FOWLER, M.; FOEMMEL, M. **Continuous integration**. 2006. Disponível em: https://martinfowler.com/articles/continuousIntegration.html. Acesso em: 10 de fev. 2019.

GITHUB. **Github Documentation**. 2015. Disponível em: https://guides.github.com/. Acesso em: 16 de jun 2019.

MAQSOOD, J.; ESHRAGHI, I.; ALI, S. S. **Success or failure identification for github's open source projects**. *In*: Proceedings of the 2017 International Conference on Management Engineering, Software Engineering and Service Sciences. New York: ACM, 2017. (ICMSS '17), p. 145–150. isbn: 9781450348348.

MENEZES, G. **Ouriço**: Uma abordagem para manutenção da consistência em repositórios de gerência de configuração. Tese (Mestrado) — Dissertação de Mestrado, Universidade Federal Fluminense-UFF, 2011.

ONECHOICE.COM.BR. **Oito em cada dez profissionais de TI são reprovados em seleções de alto nível**. 2017. Disponível em: https://www.terra.com.br/noticias/dino/oito-em-cada-dez-profissionais-de-ti-sao-reprovados-em-selecoes-de-alto-nivel,8297a2c737ab5e66f918b8bd10e21084tcf1gso6.html. Acesso em: 25 de fev. 2019.

PRESSMAN, R.; MAXIM, B. **Engenharia de Software**: Uma abordagem profissional. 8ª. ed. Porto Alegre: AMGH, 2016. 940 p. isbn: 9788580555332.

REBOUÇAS, M.; SANTOS, R. O.; PINTO, G.; CASTOR, F. **How does contributors' involvement influence the build status of an open-source software project?** *In:* Proceedings of the 14th International Conference on Mining Software Repositories. [S.I]:IEEE PRESS, 2017. 475–478 p. isbn: 9781538615447

SEPPALA, A. **Improving software quality with continuous integration**. Tese (Mestrado) — Dissertação de Mestrado, Aalto University, 2010.

SOFTEX. **Persona TI**. 2019. Disponível em: https://softex.br/download/persona-ti/. Acesso em 21 de abr. 2019.

SOMMERVILLE, I. **Engenharia de software**. 9ª. ed. São Paulo: PEARSON BRASIL, 2011. 529 p. isbn: 9788579361081.

SWICEGOOD, T. **Pragmatic version control using Git**. 1ª. ed. Raleigh: Pragmatic Bookshelf, 2008. 184 p. isbn: 9781934356159

TRAVISCI. **TravisCI Documantation**. 2013.. Disponível em: https://docs.travis-ci.com/about. Acesso em: 12 de jun. 2019

APÊNDICE A – STRINGS UTILIZADAS PARA FILTRAGEM DE BRASILEIROS

Nomes País/Estados	País/Estado Maiúsculas	País/Estado Minúsculas	Siglas Maiúsculas	Siglas Minúsculas
Brasil	BRASIL	brasil	BR	br
Brazil	BRAZIL	brazil	BR	br
Acre	ACRE	acre	AC	ac
Alagoas	ALAGOAS	alagoas	AL	al
Amapá	AMAPÁ	amapá	AP	ap
Amazonas	AMAZONAS	amazonas	AM	am
Bahia	BAHIA	bahia	BA	ba
Ceará	CEARÁ	ceará	CE	ce
Espírito Santo	ESPÍRITO SANTO	espírito santo	ES	es
Goiás	GOIÁS	goiás	GO	go
Maranhão	MARANHÃO	maranhão	MA	ma
Mato Grosso	MATO GROSSO	mato grosso	MT	mt
Mato Grosso do Sul	MATO GROSSO DO SUL	mato grosso do sul	MS	ms
Minas Gerais	MINAS GERAIS	minas gerais	MG	mg
Pará	PARÁ	pará	PA	pa
Paraíba	PARAÍBA	paraíba	PB	pb
Paraná	PARANÁ	paraná	PR	pr
Pernambuco	PERNAMBUCO	pernambuco	PE	pe
Piauí	PIAUÍ	piauí	PI	pi
Roraima	RORAIMA	roraima	RR	rr
Rondônia	RONDÔNIA	rondônia	RO	ro
Rio de Janeiro	RIO DE JANEIRO	rio de janeiro	RJ	rj
Rio Grande do Norte	RIO GRANDE DO NORTE	rio grande do norte	RN	rn
Rio Grande do Sul	RIO GRANDE DO SUL	rio grande do sul	RS	rs
Santa Catarina	SANTA CATARINA	santa catarina	SC	sc
São Paulo	SÃO PAULO	são paulo	SP	sp
Sergipe	SERGIPE	sergipe	SE	se
Tocantins	TOCANTINS	tocantins	TO	to
Distrito Federal	DISTRITO FEDERAL	distrito federal	DF	df