UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

**JENNIFER IZABEL RODRIGUES DE SOUZA**

# COMO NOVATOS AGEM AO FAZER A SUA PRIMEIRA CONTRIBUIÇÃO COM TESTE

MONOGRAFIA

**CAMPO MOURÃO**
**2019**

**JENNIFER IZABEL RODRIGUES DE SOUZA**

# COMO NOVATOS AGEM AO FAZER A SUA PRIMEIRA CONTRIBUIÇÃO COM TESTE

Trabalho de Conclusão de Curso de Graduação apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Bacharelado em Ciência da Computação do Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Igor Scaliante Wiese

**CAMPO MOURÃO**
**2019**

## ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO

Às **14:00** do dia **28 de junho de 2019** foi realizada na sala **Mini-auditório EAD** da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do(a) acadêmico(a) **Jennifer Izabel Rodrigues De Souza** com o título **Como novatos agem ao fazer a sua primeira contribuição com teste**. Estavam presentes, além do(a) acadêmico(a), os membros da banca examinadora composta por: **Prof. Dr. Igor Scaliante Wiese** (orientador), **Prof. Dr. Marco Aurélio Graciotto Silva** e **Prof. Ms. Narci Nogueira da Silva**. Inicialmente, o(a) acadêmico(a) fez a apresentação do seu trabalho, sendo, em seguida, arguido(a) pela banca examinadora. Após as arguições, sem a presença do(a) acadêmico(a), a banca examinadora o(a) considerou _____ na disciplina de Trabalho de Conclusão de Curso **2** e atribuiu, em consenso, a nota _____ (_____). Este resultado foi comunicado ao(à) acadêmico(a) e aos presentes na sessão pública. A banca examinadora também comunicou ao acadêmico(a) que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor Responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações: _____

_____

_____

_____

_____

_____

Campo Mourão, **28 de junho de 2019**

**Prof. Dr. Marco Aurélio Graciotto Silva**
Membro 1

**Prof. Ms. Narci Nogueira da Silva**
Membro 2

**Prof. Dr. Igor Scaliante Wiese**
Orientador

**A ata de defesa assinada encontra-se na coordenação do curso.**

# Resumo

Souza, Jennifer. Como Novatos Agem Ao Fazer A Sua Primeira Contribuição Com Teste. 2019. 34. f. Monografia (Curso de Bacharelado em Ciência da Computação), Universidade Tecnológica Federal do Paraná. Campo Mourão, 2019.

**Contexto:** Um método muito utilizado para contribuir para um projeto de software livre é o de *pull requests*, sendo a escrita testes uma maneira de deixar mais confiável. Um novato em um determinado projeto pode não estar ciente disso e não se preocupar em escrever teste, fazendo que o seu *pull request* acabe sendo rejeitado.
**Objetivo:** Procurar entender como os novatos realmente se relacionam com testes.
**Método:** Coleta de dados dos *pull requests*, classificação de acordo com as características interessadas, como ser de um novato e ter teste. Retirar desses dados números que representam quanto teste os novatos fazem. Posteriormente, fazer uma análise manual de uma amostra de *pull requests* aleatórios desses novatos e tentar encontrar características comuns. Por fim, lançar um questionário para poder receber a opinião direta daqueles que foram um dia novatos.
**Resultados:** O uso de testes varia bastante de uma linguagem para outra, embora quando o teste existe, a taxa de aceitação dos *pull requests* é bem maior do que a taxa de aceitação geral dos *pull requests*. Embora sejam novatos nos projetos analisados, esses usuários não são na maioria das vezes, novatos no *GitHub*, já tendo contribuído anteriormente em outro projeto. Isso faz que a maioria dos novatos que escreveram testes, os escreveram por iniciativa própria, sem que outra pessoa precisasse pedir a ele fazer fazer.
**Conclusões:** Aqueles que realmente não sabem da importância de teste se mostraram como sendo um número relativamente pequeno dentre os novatos. Eles têm dificuldades com teste na primeira contribuição, mesmo que não seja a primeira de sua vida, pois cada projeto pode ter uma abordagem diferente.
**Palavras-chaves:** teste. *pull request*. *GitHub*. novatos

# Abstract

Souza, Jennifer. How Newcomers Act When Writing Their First Contribution With Test. 2019. 34. f. Monograph (Undergradute Program in Computer Science), Federal University of Technology – Paraná. Campo Mourão, PR, Brazil, 2019.

**Context:** A widely used method to contribute to a free software project is the pull requests. Writing tests is a way to make it more reliable. A newcomer in a project may not be aware of this, and will not write test, causing their pull request to be rejected.
**Objective:** Try to understand how newcomer actually relate to tests.
**Method:** Data collection of pull requests, classification according to the characteristics involved, such as being a newcomer and having test. Finding in this data, numbers that represent how much test the newcomers do. Subsequently, perform a manual analysis of a random sample of pull requests from these beginners and try to find common characteristics. Finally, to launch a questionnaire to receive the direct opinion of those who were once newcomer.
**Results:**The use of tests varies greatly from one language to another, although when the test exists, the acceptance rate of pull requests is much higher than the general acceptance rate of pull requests. Although they are new to the analyzed projects, these users are not, for the most part, newcomers to GitHub, having previously contributed to another project. Because of this, newcomers who wrote tests, in the majority, wrote their tests on their own, without another person having to ask him to do it.
**Conclusions:** Those who do not really know the importance of testing, proved to be a relatively small number among the newcomers. They have difficulty testing their first contribution, even if it is not the first of their life, as each project may take a different approach.
**Keywords:** test. pull request. GitHub. newcomer

# Lista de figuras

# Lista de tabelas

# Sumário

Capítulo 1

# Introdução

São vários os critérios que definem um projeto como sendo de código aberto. Também chamado de *software* livre, ele não pode cobrar pela sua distribuição, deve vir com o código fonte ou com uma maneira fácil de obtê-lo, permitir trabalhos derivados sob a mesma licença, entre outros (RAYMOND, 1999). Diversos projetos permitem a colaboração da comunidade disponibilizando o código fonte em plataformas que auxiliam no processo de contribuição. Um método de contribuição muito utilizado nessas plataformas é o *pull request* (solicitação de recebimento), na qual um usuário pode fazer uma proposta de mudança no código fonte, que pode ser aceita ou rejeitada por um colaborador do projeto após a revisão do código fonte (GITHUB, 2018b).

Para se fazer um *pull request* em uma plataforma social de desenvolvimento como o *GitHub*, é necessário escolher uma *issue* (tarefa) e tentar resolvê-la. O *pull request* em si, é o processo de submeter o código criado para resolver a *issue*, que é posteriormente avaliado pela comunidade. Assim como as *issues*, cada *pull request* possibilitam os desenvolvedores trocarem mensagem para realizar a revisão do código e/ou mudança proposta (GITHUB, 2018b).

Um usuário é considerado novato em um projeto quando ele está fazendo a sua primeira contribuição nesse projeto (STEINMACHER et al., 2013). É comum que usuários novatos tendo menos conhecimento do projeto, tenham mais dificuldade em interagir com a comunidade e encontrar uma *issue* adequada para tentar resolver (STEINMACHER et al., 2013). Desenvolvedores reportaram que a presença de testes em um *pull request* aumenta o seu grau de confiança, fazendo com que tenha mais chances de ser integrado ao projeto (PHAM et al., 2013), e os novatos podem não estar cientes disso, aumentando a possibilidade de seu *pull request* não ser aceito. Entretanto, a construção de testes é mais uma barreira que os novatos podem enfrentar nesse processo de submissão da sua primeira contribuição.

Um *pull request* rejeitado é ruim tanto para o usuário que investiu o seu tempo para

solucionar o problema e obteve uma rejeição como resposta, pois pode fazer com que ele não volte mais a tentar contribuir, como também é ruim para a comunidade que gastou tempo analisando um *pull request* que acabou por não ser útil para o projeto.

Considerando esse contexto, o objetivo deste trabalho consiste em entender como os contribuidores novatos escrevem testes nos *pull requests* que eles contribuem. Para isso, iremos responder as seguintes questões de pesquisa:

**QP1:** Qual o percentual de novatos que submetem *pull requests* com testes?
**QP2:** Como são escritos os testes feitos por novatos nos *pull requests*?
**QP3:** Quais são as dificuldades que os contribuidores enfrentam em relação a testes?

Como os projetos precisam dos contribuidores para se manterem sustentáveis, as respostas dessas questões poderão auxiliar na construção de soluções e ferramentas que ajudem os novatos a fazerem um *pull request* com mais chances de ser aceito, para assim, se tornarem contribuidores mais regulares.

Desse modo, o Capítulo 2 contará com o referencial teórico utilizado neste trabalho. No sessão 2.3 serão apresentados os trabalhos relacionados à este, bem como uma breve descrição sobre eles. O Capítulo 3 traz as informações referentes à metodologia que este trabalho segue, e todos os passos necessários para a sua realização, como a seleção dos projetos na Seção 3.1, a coleta e organização dos dados na Seção 3.3, a forma como serão respondidas as questões de pesquisa na Seção 3.4, e forma como se deu a análise dos dados na Seção 3.5. Os resultados obtidos se encontram no Capítulo 4, onde estão divido sem Seções de acordo com a questão de pesquisa respondida. Seção 4.1 para a questão de pesquisa 1, Seção 4.2 para a questão 2 e Seção 4.3 para a terceira questão de pesquisa. Esse capítulo também conta com a Seção 4.4 de ameaças à validade. O Capítulo 5 trás a conclusão e os possíveis trabalhos futuros.

Capítulo 2

# Referencial Teórico

Diversos conceitos foram utilizados como base para a realização desse trabalho. Abaixo são explicados cada um destes conceitos.

## 2.1. Contribuição em Projetos de Software Livre

O software livre, também chamado de código aberto, *open souce*, OSS, entre outras variações, diz respeito a um modelo de se fazer software que é diferente do software chamado proprietário. Enquanto um software proprietário é fechado e licenciado exclusivamente para o seu produtor, um software livre deve ser distribuído com o código fonte, ou com fornecer uma maneira fácil e gratuita de obtê-lo através da Internet. A sua licença de uso também não pode proibir a distribuição gratuita do código ou a sua alteração pelos mesmos termos de licença (KAVANAGH, 2004). Isso permite que a comunidade possa ajudar os projetos de software livre a evoluírem. Para auxiliar nesse processo, existem plataformas onde os códigos fontes estão disponíveis e que permitem a qualquer pessoa interessada, tentar fazer uma contribuição para esse projeto.

Um exemplo de plataforma que agrega muitos projetos de software livre diferentes é o GitHub [1]. Lançado em Outubro de 2007 em São Francisco, Califórnia, conta hoje com mais de 31 milhões de usuário e 96 milhões de repositórios hospedados (GITHUB, 2018a), sendo que muitos desses repositórios são de projetos abertos à contribuição da comunidade. Um desses projetos hospedado no GitHub é inclusive um guia sobre como contribuir, inicializado pelos mesmos autores do site, mas de código aberto à contribuição (GUIDE, 2018). Neste guia é possível encontrar uma descrição da anatomia de um projeto de software livre, como encontrar um projeto adequado para contribuir, e como fazer essa contribuição. Esses tópicos serão abordados nas Subseções 2.1.1 e 2.1.2.

[1] <https://github.com/>

### 2.1.1. Anatomia de um projeto

Um projeto é composto por muitos usuários de diferentes tipos, bem como documentações e ferramentas que os ajudam. Entre os usuários de um projeto estão os autores (*author*), que é a pessoa ou a organização que criou o projeto. O proprietário (*owner*) que é a pessoa que tem a propriedade sobre a organização ou o repositório. Mantenedores (*mainteners*) são contribuidores responsáveis por dirigir e organizar o projeto, sendo geralmente também um autor ou proprietário do projeto. Os contribuidores (*contributors*) são todos aqueles que contribuem de alguma forma para com o projeto. E por fim um membro da comunidade (*community member*) são todas as pessoas que utilizam o projeto e que podem ser membros ativos e expressar as suas opiniões sobre o projeto e seu desenvolvimento (GUIDE, 2018).

Sobre as documentações de um projeto, temos a licença (*LICENSE*) que por definição é obrigatório para projetos serem considerados como de software livre. O arquivo *leia-me* (*README*) é o manual de instruções que deve dar as boas vindas à novos membros da comunidade, bem como dizer para que o projeto serve, e dar instruções de como utilizá-lo. Já o arquivo *contribuindo* (*CONTRIBUTING*) é o documento que auxilia as pessoas a como contribuir para o projeto e é disponibilizado geralmente apenas naqueles projetos que estão dispostos a receber qualquer tipo de novo contribuidor. Código de conduta (*CODE_OF_CONDUCT*) também é mais frequente em projetos que querem atrair contribuidores, pois dita regras aos participantes a fim de tornar o ambiente mais amigável. Projetos maiores também podem contar outras documentações como tutoriais, por exemplo. (GUIDE, 2018).

Entre as ferramentas está o rastreador de questões (*issue tracker*) que é onde as pessoas iniciam conversas e discussões a cerca de problemas a serem resolvidos, e tarefas a serem realizadas no projeto. Uma dessas questões, a partir daqui referidas apenas como *issues*, pode receber uma solicitação de recebimento, que é chamada de *pull request*, e é o início do trabalho em uma solução, onde as pessoas podem discutir sobre e revisar a alteração que estiver sendo proposta. Alguns projetos também utilizam fóruns de discussão e lista de e-mails, e outros utilizam ainda outros tipos de canais de discussão, como o IRC e o Slack, por exemplo. (GUIDE, 2018).

### 2.1.2. Encontrando um projeto e contribuindo para ele

Uma boa maneira de começar a decidir em qual projeto iniciar é pensar em quais você usa geralmente, ou que gostaria de contribuir. Posteriormente deve-se verificar se o projeto em questão segue os parâmetros estabelecidos na Subseção 2.1.1, principalmente aos que dizem respeito às documentações. Não é necessário contribuir com código fonte imediatamente, já que 28% dos contribuidores casuais contribuem com documentações, reformatações, e traduções (PINTO et al., 2016). Para verificar se o projeto é apto a

receber novos contribuidores, pode-se checar sua licença, o tamanho da sua comunidade de contribuidores, e a data e a frequência das *issues* e dos *pull requests*, bem como se os mantenedores respondem e como respondem as perguntas realizadas a eles (GUIDE, 2018).

Verificada a viabilidade do projeto em receber novos contribuidores, pode-se escolher uma tarefa a que se considere apto à resolvê-la e submeter um *pull requests*. Caso encontre um problema novo e não puder resolvê-lo sozinho, abra uma *issue*, e permita que outras pessoas tentem resolvê-lo. Após se fazer um *pull request* várias coisas podem acontecer. Ele pode bem ser aceito, como também pode ser rejeitado, uma alteração na sua solução pode ser solicitada, bem como ele pode ser ignorado. Nesse último caso, convém pedir educadamente para alguém fazer uma revisão (GUIDE, 2018).

## 2.2. Teste de Software

Existe um grande crescimento do interesse dos desenvolvedores em relação à qualidade dos softwares produzidos, o que faz com que a atividade de testes seja vista como importante e indispensável (DELAMARO, 2007). O tempo gasto com testes corresponde a 50% do tempo gasto com toda a produção do software, ou seja, a mesma quantidade de tempo que se leva para escrever um código, se leva para testá-lo (MYERS, 2012). Testar um software geralmente é definido como um processo ou uma série de processos utilizado para garantir que o código de um programa está fazendo o que ele deveria estar fazendo. Essa definição pode gerar controvérsias já que é ideal se ter em mente quando está se escrevendo testes, que não se deve querer provar que o código não possui erros, mas sim, procurar possíveis erros que possam existir nesse código (MYERS, 2012). Para garantir que o código tenha alta cobertura de testes, são utilizadas diversas técnicas e ferramentas, como o teste unitário e TDD, que serão brevemente explicadas nas Subseções 2.2.1 e 2.2.2 respectivamente.

### 2.2.1. Teste unitário

É o teste de uma pequena e muito específica funcionalidade do código, para checar se ela está fazendo o que o programador queria que ela fizesse, como por exemplo, testar se ao executar um comando de remover um número de uma lista esse número é realmente removido. Neste momento não se está preocupado com a performance dos testes, apenas em provar que o código faz o que se é esperado dele, desse modo cada pequena funcionalidade é testada. Quando se faz um teste para uma funcionalidade, não se executa apenas o novo teste, pois deve-se garantir que o novo código, apesar de passar no teste escrito para ele, não cause um efeito colateral fazendo com que outros testes falhem (HUNT, 2004).

### 2.2.2. Desenvolvimento orientado a testes (TDD)

Nesta técnica, o desenvolvimento do código fonte é orientado aos testes previamente escritos, ou seja, só é escrita uma nova linha de código quando um teste automatizado falha. Para isso funcionar, deve-se primeiramente escrever códigos que testem pequenas funcionalidades de poucas linhas. Inicialmente esses testes falham e essa é realmente a primeira tarefa, chamada de tarefa vermelha. Deve-se então escrever o código que fará com que o teste passe rapidamente e fazer o *commit* dessa alteração. Esse passo é chamado de tarefa verde. O último passo do "Mantra do TDD"é a refatoração, que deixa apenas o código necessário para os testes funcionarem. Desse modo, só serão incluídos códigos que foram previamente testados e que passaram nestes testes. (BECK, 2003)

## 2.3. Trabalhos Relacionados

### 2.3.1. *When Testing Meets Code Review: Why And How Developers Review Tests*

O trabalho de Davide Spadini, Maurício Aniche, Margaret-Anne Storey, Magiel Bruntink e Alberto Bacchelli, publicado no ICSE'18 (SPADINI et al., 2018), tem como objetivo descobrir como a revisão de código é utilizada para melhorar a qualidade dos códigos de teste e o que os revisores pensam e fazem ao revisá-los. Para isso eles analisaram mais de 300.000 revisões de código, e posteriormente entrevistaram 12 desenvolvedores sobre revisão de arquivos de teste. Como um estudo preliminar, eles utilizaram um método de pesquisa proposto por Shane McIntosh (MCINTOSH et al., 2014), onde é avaliado se a participação e cobertura da revisão de código influenciam na qualidade do software. Diferentemente da pesquisa original, as métricas foram calculadas a nível de arquivo e não de pacote e descobriu-se que a decisão de rever um arquivo não deve ser associada ao fato de ter ou não teste, já que isso não interfere na presença de testes.

Foram definidos como critérios para seleção de estudo, projetos com teste de código, alta taxa de revisão de código e que utilizam a ferramenta de revisão de código Gerrit. Diante desses critérios, os projetos selecionados foram o Eclipse, Openstack e o Qt. Foi utilizado o Gerrit e selecionaram apenas revisões com menos de 50 arquivos e pelo menos um revisor diferente do autor. Para a avaliação foi considerado o número de comentários no arquivo, número de arquivos com comentários, número de revisores e o tamanho dos comentários. As revisões foram separadas em 3 categorias, sendo aquelas que pertenciam tanto aos arquivos de produção quanto aos de teste, os apenas de produção e os apenas de teste. Foram selecionados 600 comentários para serem analisados manualmente a fim de se descobrir sobre o que se

tratava o comentário, se era uma melhoria de código, entendimento, comunicação social, defeito, entre outros.

Concluiu-se então que para as revisões que tratam tanto de arquivos de teste, quanto de produção, os arquivos com teste possuem muito menos comentários do que os sem teste. Já para os somente de produção e os somente de teste, a porcentagem de com e sem comentários ficou muito próxima, não apresentando distinção significativa. Quanto ao que se discutiu nesses comentários, as análises manuais mostraram que a maioria sugere melhoramentos de código, seguido por questões de entendimento e comunicação social.

### 2.3.2. *Analyzing The Effects Of Test Driven Development In GitHub*

Neste trabalho dos autores Neil C. Borle, Meysam Feghhi, Eleni Stroulia1, Russell Greiner, e Abram Hindle, publicado na Springer (BORLE et al., 2018), analisa-se o custo-benefício do uso da técnica de *Test Driven Development*, em português, desenvolvimento orientado a testes. A técnica, mais conhecida pela sigla TDD, consiste na prática da metodologia ágil de escrever os testes antes de escrever o código. Estes códigos devem inicialmente falhar e o código fonte deve ser incrementado até que os testes passem. Os autores desejavam descobrir, entre outras coisas, se o uso dessa técnica afeta a colaboração dos desenvolvedores.

Primeiramente os projetos foram classificados como utilizando ou não a técnica de TDD. Para isso assume-se que sempre se utiliza a técnica seguindo o padrão de ter nomes similares entre os arquivos de teste e seus respectivos arquivos de código fonte. Por exemplo, o arquivo `OpenTest.java` corresponde ao arquivo de teste do `Open.java`, podendo existir variações como `Test-Open.java, TestingOpen.java`. Para conseguir então identificar os arquivos de teste, foi-se utilizada uma expressão regular para identificar se existe a palavra `test` no nome do arquivo. Para confirmar que realmente se tratava de TDD, foi também analisado se a criação do arquivo de teste foi feita antes do arquivo de código fonte.

Os projetos que utilizavam TDD foram comparados à projetos que não utilizavam esse técnica. Foi-se descoberto que aqueles projetos que utilizam TDD são realmente raros, o que mostra que a técnica não é muito popular, e ainda não se obteve benefícios observáveis no uso da técnica.

### 2.3.3. *Why Do Newcomers Abandon Open Source Software Projects?*

O trabalho de Igor Steinmacher, Igor Wiese, Ana Paula Chaves e Marco Aurelio Gerosa trata do fato de que novatos em projeto de software livre enfrentam muitos obstáculos e dificuldades quando tentam fazer uma contribuição, fazendo com que muitos deles acabem

por não continuar contribuindo (STEINMACHER et al., 2013). Apesar de existirem vários lugares onde esses novatos podem procurar por informações e ajuda, o volume de informações é bem grande, e existe uma falta de boas ferramentas que os ajude a encontrar aquelas informações que eles precisam. Mesmo quando eles tentam fazer perguntas em fóruns e lista de e-mails, nem sempre as respostas são esclarecedores e ajudam o novato no que ele realmente precisa. Desse modo, o trabalho buscou saber se a falta de respostas, de cordialidade e de utilidade das respostas dadas aos novatos, influenciam na sua decisão de continuar ou não a contribuir em um projeto.

Foram então coletados dados de 60 meses (5 anos, 2006-2010) de *issues* no Jira (ATLASSIAN, 2018), e de e-mails de discussão. Os usuários foram categorizados então entre *core members*, novatos e outros membros. Foi-se verificado se os novatos recebiam respostas para as suas perguntas, e quando sim, qual era a categoria do usuário que o respondia. O modo como essa resposta foi dada também foi avaliado, se a resposta ajudou ou não o novato ou se foi indiferente. Foram também enviados questionários à alguns dos novatos que não continuaram contribuindo.

Como conclusão, a taxa de retenção dos novatos é de 18% na lista de e-mails, e de 13% no Jira. Pelas respostas dos questionários descobriu-se que muitos realmente não desejavam se tornar um membro ativo do projeto, embora 46,15% dos que tenham desistido, relataram insatisfação com as respostas que receberam às suas perguntas. Também concluiu-se que novatos que eram respondidos por outros novatos tinham mais chances de sair do projetos, pois as respostas era mais insatisfatórias, e que alguns novatos que não desistiram, tornaram posteriormente membros muito ativos no projeto.

### 2.3.4. *Curating GitHub For Engineered Software Projects*

No trabalho de Nuthan Munaiah, Steven Kroh, Craig Cabrey e Meiyappan Nagappan, intitulado *"Curating GitHub for engineered software projects"*(MUNAIAH et al., 2017), com o objetivo de separar os repositórios contendo projetos que utilizam engenharia de software, dos repositórios contendo projetos pessoais, trabalhos de faculdade e afins, foi-se implementado um *framework* de código aberto chamado *reaper*, para ser capaz de fazer essa distinção. Utilizando a base do GHTorrent (GOUSIOS, 2013) e Boa (DYER et al., 2013) para adquirir os repositórios, utilizou-se um algoritmo de *Random Forest* para classificar cada repositório como utilizando ou não engenharia de software.

Para realizar a classificação, foram determinadas diversas métricas. Uma delas é a de *issues*, que é considerada pelos autores, como um ponto essencial para garantir a sustentabilidade de um projeto de software livre no GitHub. Essa métrica é dada pela frequência mensal do uso sustentado de *issues* em um determinado repositório através da

fórmula:

$$M_i(r) = \frac{1}{m}\sum_{i=1}^{m} s_i$$

Onde,
- $s_i$ é o numero de eventos de *issues* no mês $i$
- $m$ é o numero de meses entre o primeiro e último *commit* no repositório $r$

Uma outra métrica que foi determinada é a testes unitários, chamada de *test ratio*, que quantifica a extensão de esforço dos testes unitários. É definida pela fração entre o número de linhas de código em arquivos de teste e o número de linhas de código em todos os arquivos, definida pela fórmula:

$$M_u(r) = \frac{slotc}{sloc}$$

Onde,
- *slotc* é o número de linhas de código em arquivos de teste no repositório $r$
- *sloc* é o número de linhas de código em todos os arquivos do repositório $r$

Todos os dados obtidos formaram uma base de dados, que está disponível para download no site [2] do projeto.

### 2.3.5. Considerações Finais

No trabalho relacionado da Seção 2.3.1, foram selecionados 600 comentários de revisões de 3 projetos diferentes e analisou-se manualmente o impacto de que ser um arquivo de teste gera em relação à comentários. Para o trabalho que estamos realizando, foram selecionados todos os *pull requests* de 70 repositórios, de 7 linguagens diferentes (10 de cada linguagem), a fim de se identificar os perfis de usuários que realizam ou não, testes, e investigar melhor o papel exercido pelos novatos. Alguns *pull requests* também serão investigados manualmente e também haverá um questionário para alguns usuários.

Utilizamos o método do artigo da Seção 2.3.2 para selecionar se o arquivo era de teste ou não, mas não foi necessário observar se o arquivo era TDD através da data, apenas se possuía ou não testes. Com o trabalho da Seção 2.3.3 foi que surgiu o interesse em se investigar se o fato de um novato escrever testes ou não, o ajuda a ter seu *pull request* aceito. As métricas destacadas, da base resultante do trabalho citado na Seção 2.3.4, foram utilizadas para nos auxiliar na escolha dos projetos que seriam analisados.

[2] <https://reporeapers.github.io/results/1.html>

Capítulo 3

# Metodologia

Neste capítulo são apresentados os passos necessários para responder as questões de pesquisa desta pesquisa. A Seção 3.1 traz informações sobre como se deu o processo de seleção dos repositórios que foram coletados do modo descrito na Seção 3.3 para responder as questões descritas na Seção 3.4. A Seção 3.2 define o conceito de novato utilizado.

## 3.1. Seleção dos Projetos

Para selecionar os projetos a serem analisados, foi-se utilizada a base de dados da RepoReapers (MUNAIAH et al., 2017), que conta com diversas informações sobre repositórios no GitHub, tais como a linguagem de programação utilizada, quantidade de estrelas, tamanho, e também algumas métricas para dados como documentação e *issues*, que são especificadas em sua documentação. Um resumo de todo o processo de seleção pode ser observado na Figura 3.1.

Feito o *download* da base de dados, descobriu-se que ela conta com informações de 1.853.205 repositórios. Analisar todos eles seria inviável, dado a dificuldade de coleta de dados. A base possui dados de repositórios em 7 linguagens diferentes (*C, C++, C#, Java, Python, PHP, Ruby*), que inclusive estão entre as 10 mais utilizadas no GitHub (Figura 3.2 (ZAPPONI, 2014)). Decidiu-se então por coletar uma pequena quantidade de cada uma [1].

Pelo foco da pesquisa ser a relação entre os testes para a aceitação de *pull requests*, o primeiro passo para fazer a seleção dos projetos a serem analisados, foi separar apenas os repositórios onde a métrica referente à testes fosse maior que 0, o que significa que o repositório em questão possui testes. Para facilitar as próximas etapas, os dados da base original foram separado em 7 novos arquivos diferentes, de acordo com as linguagens de programação utilizadas. Cada arquivo foi ordenado a partir das maiores métricas para *issues*,

[1] Dados, *scripts* e etc, disponíveis em <https://github.com/JenniferIzabel/TCC-Files>

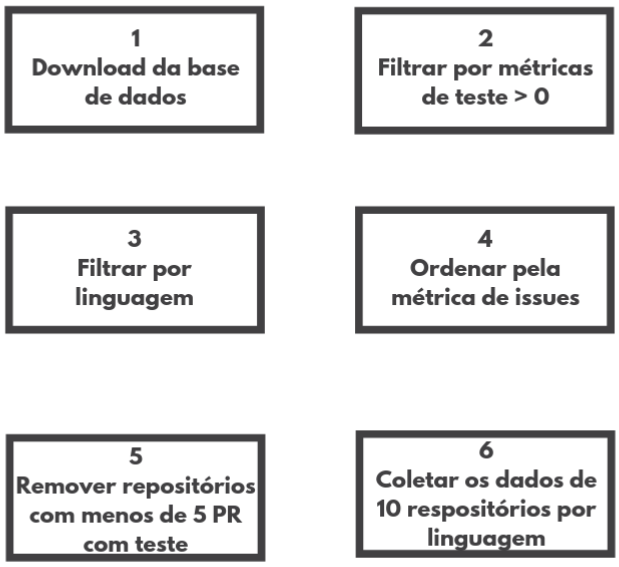


**Figura 3.1.** Processo de seleção dos repositórios

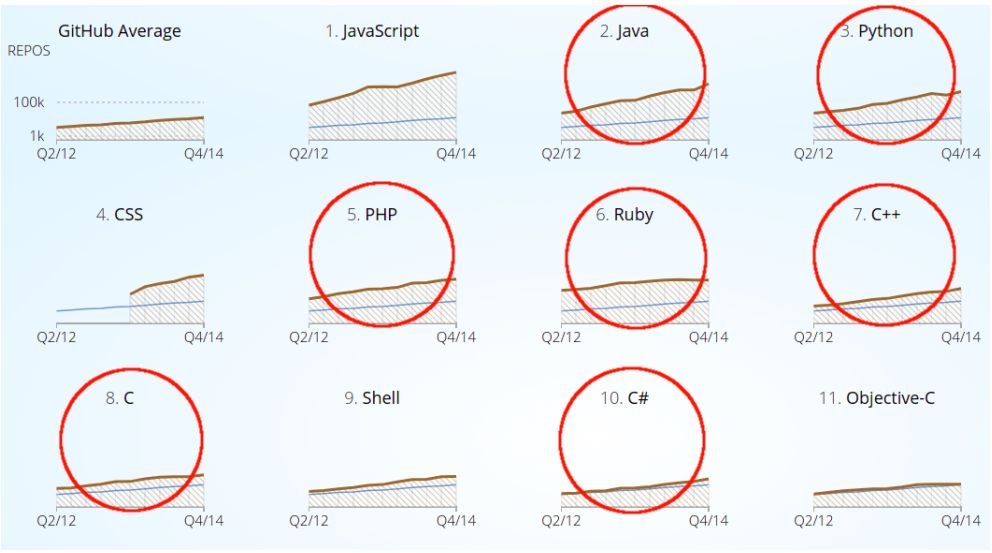


**Figura 3.2.** Linguagens mais utilizadas no GitHub.

e então começou a coleta.

Durante a coleta, percebeu-se que alguns repositórios, apesar do filtro feito na primeira etapa, acabaram por não ter nenhum, ou muito poucos *pull requests* para serem analisados, o que prejudicaria as análises. Estabeleceu-se então que para um repositório ser analisado, ele deveria ter pelo menos 5 *pull requests* com teste. Na Tabela 3.1 podemos observar que entre os 10 primeiros repositórios, apenas 2 puderam ser utilizados na pesquisa. Por esse motivo, decidimos limitar em 10 repositórios de cada linguagem.

**Tabela 3.1.** Repositórios selecionados em *Java*

| Número | Repositório | Foi selecionado? |
|---|---|---|
| 1 | **TEAMMATES/repo** | X |
| 2 | Craig-Macomber/WeatherOracle | |
| 3 | MassBayCS225/EMS | |
| 4 | ZooTypers/ZooTypers | |
| 5 | cs2103aug2014-W09-3j/main | |
| 6 | **wtud/TSAP** | X |
| 7 | huskysoft/403Interviewer | |
| 8 | champo/TPE-PAW | |
| 9 | cs2103aug2014-w14-2j/main | |
| 10 | cs2103aug2014-f11-1j/TaskCommander | |
| 11 | cs2103aug2014-t15-1j/main | |
| 12 | **realtalk403/realtalk** | X |
| 13 | turesheim/eclipse-utilities | |
| 14 | **fabric8io/fabric8** | X |
| 15 | **elasticsearch/elasticsearch** | X |
| 16 | **RostarSynergistics/ShinyExpenseTracker** | X |
| 17 | mff-uk/ODCS | |
| 18 | coldstar96/cse403 | |
| 19 | cs2103aug2014-t11-1j/main | |
| 20 | servicosgovbr/guia-de-servicos | |
| 21 | **CS340Group/hit** | X |
| 22 | **enonic/xp** | X |
| 23 | **mozilla/j2me.js** | X |
| 24 | **wildfly/wildfly** | X |
| 25 | 2014-Android-ITX4/RollingBALL | |

Diante disso, os repositórios não puderam ser selecionados previamente, pois precisávamos coletar alguns dados sobre ele para saber se ele cumpriria os critérios de seleção.

## 3.2. Contribuidor Novato

Para prosseguir com a pesquisa, nesse momento precisamos definir os critérios que usaríamos para classificar um usuário como novato. Decidimos que seriam considerados novatos todos aqueles que tivessem contribuído apenas uma vez com *pull request* em determinado projeto. Esse critério foi escolhido pois, para verificar o primeiro *pull request* de cada um dos usuários, exigiria muito processamento e tempo que não tínhamos disponíveis.

## 3.3. Coleta e Organização de Dados

Para realizar a coleta, foi utilizada a própria *API* do GitHub pois fazendo uma requisição autenticada, é possível obter uma grande quantidade de dados por hora. O processo,

que será detalhado a seguir e está simplificado na Figura 3.3, equivale para um repositório, e foi repetido para cada um dos repositórios selecionados em cada linguagem.

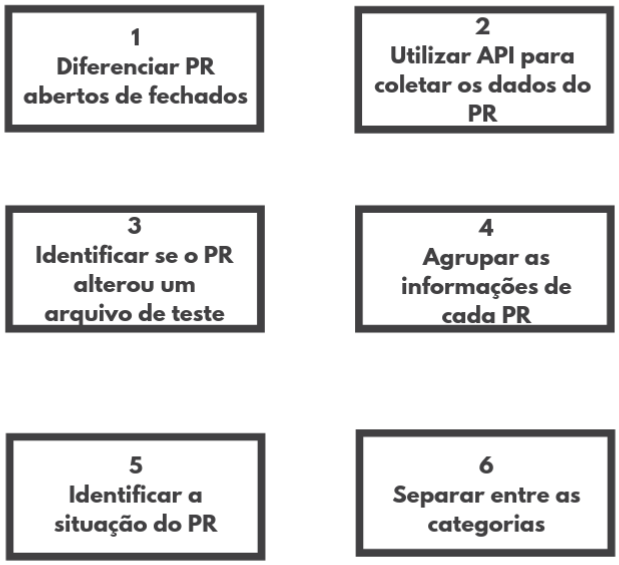


**Figura 3.3.** Processo de coleta e organização dos dados

Começamos então diferenciando os *pull request* em aberto dos fechados, pois cada um exige um link diferente para fazer a requisição pela *API*. Em cada requisição, podemos obter uma página de resultados, que possui tamanho variável. Definimos esse tamanho no valor máximo, que é 100, obtendo assim, os dados referentes aos 100 *pull requests* mais recentes. As páginas vêm no formato *JSON* e possuem apenas dados mais básicos como o número do *pull request*, seu estado (aberto ou fechado), link para os arquivos que foram alterados, assim como as datas de criação, alteração, fechamento e integração do *pull request*.

A partir desses dados é possível fazer novas requisições para obter mais dos dados necessários. Para cada *pull request* da primeira requisição, é gerada uma nova requisição para obter informações mais específicas. Dados como o *sha* (identidicador único) do *pull request* e uma lista de cada arquivo que esse *pull request* alterou, qual foi essa alteração, a quantidade de adições e exclusões, assim como o usuário que o realizou.

Borle utiliza entre outras heurísticas, a de que arquivos que contem casos de testes geralmente têm a palavra *"test"*, ou alguma variação, no nome do arquivo, do pacote, ou em qualquer outro lugar no caminho do arquivo dentro do repositório (BORLE et al., 2018). Utilizamos desse princípio para criar uma função capaz de identificar se um arquivo era de teste ou não. Ao final, se o *pull request* modificou algum arquivo de teste, todas as informações obtidas são armazenadas no arquivo referente aos *pull requests* com teste daquele

repositório. Caso contrário, as informações ainda são armazenadas, mas no arquivo referente aos *pull requests* sem teste.

Após coletados os dados dos repositórios de uma linguagem, começamos a agrupar os *pull requests* de mesmo número em uma só linha, já que no processo anterior, um determinado *pull request* apareceria n linhas diferentes, sendo n o número de arquivos que aquele *pull request* alterou. Essa linha única criada contém informações como qual o repositório, o número do *pull request*, quem foi o usuário que o realizou, se teve testes ou não, e se o *pull request* foi ou não integrado ao projeto.

Com as informações de um *pull request* reunidas, foi possível identificar, através das datas, a situação do *pull request*. Se tiver apenas a data de criação, significa que se trata de um *pull request* aberto (*Open*). Caso também possua data de fechamento, ele foi recusado (*Closed*). E se além disso, também possuir data de integração, é porque o código daquele *pull request* foi integrado ao projeto (*Merged*).

Como o nosso interesse foi apenas naqueles usuários que fizeram apenas uma contribuição no projeto, os separamos dos demais usuários. Eles foram identificados como aqueles cujo nome de usuário aparecia apenas uma vez na coluna referente ao autor do *pull request*. As informações referentes aos outros tipos de usuários foram mantidas para eventuais comparações.

Tendo obtido as principais informações, nós as organizamos em um arquivo do modo como pode ser visto na Tabela 3.2. Cada arquivo é referente a uma linguagem e possui o nome do repositório, o nome do usuário autor e o número do *pull request* em questão, bem como se possui um caso de teste e se foi aceito.

**Tabela 3.2.** Informações de alguns usuários novatos em *Python*

| **Repositório** | **Usuário** | ***Pull request*** | **Teste** | **Situação** |
|---|---|---|---|---|
| andresriancho_w3af.csv | 13rac1 | 4165 | TRUE | Closed |
| andresriancho_w3af.csv | 3rdDegree | 2292 | FALSE | Merged |
| andresriancho_w3af.csv | Ajeet-Yadav | 14800 | FALSE | Merged |
| andresriancho_w3af.csv | alexpantyukhin | 17194 | FALSE | Merged |
| andresriancho_w3af.csv | andersonDadario | 13574 | TRUE | Open |
| andresriancho_w3af.csv | andresriancho | 195 | FALSE | Closed |
| andresriancho_w3af.csv | atracy | 2252 | FALSE | Closed |
| andresriancho_w3af.csv | AverageS | 14022 | FALSE | Open |
| ..... | ..... | ..... | ..... | ..... |

## 3.4. Questões de Pesquisa

O objetivo principal deste trabalho procura entender como os contribuidores novatos escrevem testes em seus *pull requests*. Pra isso, respondemos às três questões de pesquisa da seguinte forma:

**QP1: Qual o percentual de novatos que submetem *pull requests* com testes?** Como estão fazendo a sua primeira contribuição no projeto, será que eles sabem que devem realizar teste? Para responder essa questão, analisaremos puramente a presença de teste nos *pull requests* realizados por novatos, bem como a aceitação desses.

**QP2: Como são escritos os testes feitos por novatos nos *pull requests*?** São várias as ações que um usuário pode realizar sobre um arquivo. É possível criar um arquivo de teste novo, adicionar testes a um arquivo já existente, modificar testes, etc. Nessa questão buscamos analisar manualmente alguns *pull requests* a fim de tentar responder essa questão. Foram selecionados 234 *pull requests* de 15 repositórios das 7 linguagens.

**QP3: Quais são as dificuldades que os contribuidores enfrentam em relação a testes?**

A melhor maneira de saber qual a dificuldade que um contribuidor enfrenta é perguntando isso diretamente a ele. Por esse motivo, foi elaborado um questionário que foi enviado a 700 indivíduos que fizeram apenas uma contribuição, e recebemos ao total 41 respostas.

## 3.5. Análise dos Dados

A análise dos dados foi realizada de duas formas distintas: Uma análise manual de uma amostra de *pull requests*, e uma análise das respostas recebidas por um questionário que foi enviado a autores de *pull requests* com teste feito por novatos.

### 3.5.1. Análise Manual

O processo realizado para a análise manual foi resumido na Figura 3.4. Para decidir quantos *pull requests* deveriam ser analisados manualmente, foi-se utilizada uma calculadora amostral [2], com os parâmetros de erro amostral em 5%, nível de confiança em 90% e distribuição da população como mais homogênea (80/20). Com a população total de 4808 *pull requests* com teste, feito por novatos, resultou em uma amostra de 234 indivíduos. Essa quantidade foi dividida entre todas as linguagens, de acordo com a porcentagem dela na população.

Tendo a quantidade necessária para cada linguagem, foi criado um *script* para a seleção de *pull requests* aleatórios. Posteriormente acessamos cada um deles e verificamos se eles realmente se encaixavam nos critérios aos quais tinham sido selecionados, como ter tido um arquivo de teste alterado por um novato no projeto. Os *pull requests* possuíam muitas características distintas, mas que se repetiam em outros *pull requests*. Desse modo, eles foram agrupados de acordo com essas características, podendo um estar em mais de uma categoria ao mesmo tempo. As características identificadas e utilizadas como categorias foram:

[2] Calculadora Amostral: <https://comentto.com/calculadora-amostral/>

- Usuários que tiveram a iniciativa de fazer teste;
- Usuários que fez o teste porque um membro o pediu;
- Usuários em que o *pull request* identificado como primeiro no projeto, foi também o seu primeiro no *GitHub*.
- Usuários em que o seu primeiro *pull request* no projeto, não foi o seu primeiro no *GitHub*;
- Usuários e *pull requests* que foram casos peculiares, como remover testes, por exemplo, ou quando o *commit* foi aceito mas o *pull request* foi fechado;
- Aqueles falso positivos onde a classificação automática dos dados coletados nos enganaram, e na verdade não era um *pull request* com teste, feito por um novato.

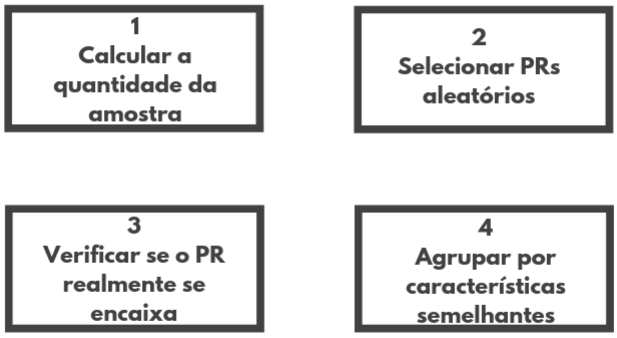


**Figura 3.4.** Processo de análise manual de *pull requests*

### 3.5.2. Questionário

Como resumido na Figura 3.5, o processo referente ao questionário foi iniciado com a de elaboração das questões, que foram pensadas de acordo como público alvo. Esperávamos receber as respostas de pessoas que já foram alguma vez um novato em um projeto de software livre do *GitHub*, e que tenha escrito testes nessa ocasião. As questões foram elaboradas de modo a tentar classificar os usuários nas categorias descobertas no processo de análise manual (seção 3.5.1). As questões colocadas no questionário foram as seguintes, traduzidas para o inglês:

1. Quantas das suas contribuições foram aceitas em projetos do *GitHub*?
   - 1 - 5;
   - 6 - 15;
   - $>$ 15;
2. Quantas das suas contribuições aceitas em projetos do *GitHub* possuíam teste?
   - 1 - 5;

   - 6 - 15;
   - > 15;
3. Você escreveu testes por conta própria, ou por que algum membro te pediu?
   - Por conta própria;
   - Algum membro me pediu;
4. Você conhecia *frameworks* de teste antes da sua primeira contribuição?
   - Sim;
   - Não;
5. O que te ajudou quando você estava escrevendo teste? *(mais de uma opção podendo ser selecionada)*
   - Documentação;
   - *Bots*;
   - *Travis/Jenkins* ou outra ferramenta de integração contínua;
   - Membro do projeto/revisor;
   - *Stack Overflow*;
   - Informação do *Reddit* ou *Hacker News*;
   - Outro *(descrever)*;
6. Da perspectiva de um novato, quais são os problemas/desafios de criar o primeiro teste? (Ex: saber o que testar, saber utilizar *frameworks* de tests, encontrar alguém para ajudar, etc)
7. Caso deseje receber os resultados da pesquisa, sinta-se livre à deixar o seu e-mail

Como a *API* do *GitHub*, que foi utilizada para coletar os dados iniciais (seção 3.3), não permitia coletar os endereços de e-mail, foi preciso fazer um *web scrapping* para pegar essa informação da página de perfil do usuário. Como nem todos os usuários possuíam essa informação visível em seu perfil, foram obtidos apenas cerca de 700 endereços de e-mail entre os 4808 usuários analisados (cerca de 14,56%). Foi criado então um *script* para enviar para esses usuários um e-mail com uma breve explicação sobre o processo ao qual eles estariam participando, e o link para responder o questionário. Foram recebidas, ao todo, 41 respostas (aproximadamente 5,8% dos e-mails enviados).

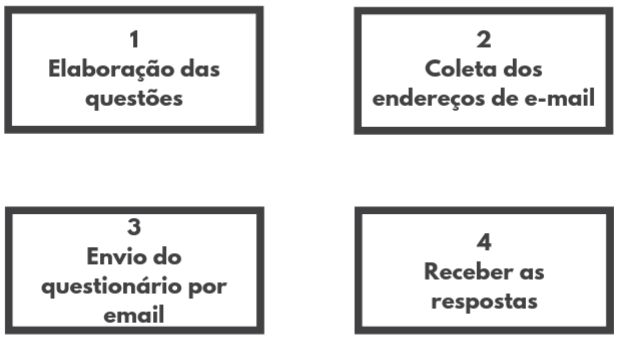


**Figura 3.5.** Processo de aplicação do questionário

Capítulo 4

# Resultados

A partir dos dados coletados, *pull requests* analisados e do questionário lançado foram analisados os resultados obtidos em cada etapa. Observando as informações coletadas, fomos capaz de responder as 3 questões de pesquisa, como será mostrado nas seções 4.1, 4.2 e 4.3.

## 4.1. QP1: Qual o percentual de novatos que submetem *pull requests* com testes?

A partir das dados coletados e organizados sobre cada *pull request* (seção 3.3), pudemos observar diretamente para as informações que nos são mais úteis para responder a questão de pesquisa. Considerando somente para os usuários novatos, percebemos uma diferença significativa em relação ao uso de teste entre os novatos que contribuíram para repositórios em uma linguagem, ou em outra. Usuários que contribuíram em repositórios Java, por exemplo, somaram 59,02% de uso de teste, enquanto aqueles repositórios que eram em Ruby, somaram apenas 11,75%. Diante da grande diferença entre uma linguagem e outra, a média entre todas acabou ficando em 31,61%. Essas informações podem ser observadas na Tabela 4.1.

Por outro lado, os *pull requests* que possuíam teste tiveram uma alta taxa de aceitação em todas as linguagens. O menor taxa foi para PHP, com 86,38%. Já a maior foi para Ruby com 96,88%. Ou seja, Ruby que é a linguagem onde os *pull requests* menos possuem teste, quando o têm, são os mais aceitos. Com todos os valores muito próximos, a média entre todas as linguagens ficou em 91,32%.

Com as coluna da aceitação dos *pull requests* com teste logo ao lado da coluna de aceitação total dos sem teste, fica visível a diferença. A taxa média de aceitação dos *pull requests* sem teste é de 78,67%, enquanto a de *pull requests* com teste é 91,32%. Isso mostra

**Tabela 4.1.** Porcentagens referentes aos novatos de cada linguagem.

| Linguagem | *Pull request* de novatos com teste | Com teste que foram aceitos | Sem teste que foram aceitos |
|---|---|---|---|
| C | 39,01 % | 92,44 % | 84,83 % |
| C++ | 14,64 % | 93,53 % | 71,34 % |
| C# | 48,80 % | 86,52 % | 89,76 % |
| Java | 59,02 % | 87,90 % | 85,84 % |
| PHP | 35,31 % | 86,38 % | 75,22 % |
| Python | 36,06 % | 92,01 % | 88,24 % |
| Ruby | 11,75 % | 96,88 % | 66,12 % |
| Média entre todas | 31,61 % | 91,32 % | 78,67 % |

que a presença de teste no *pull request*, com exceção de C#, aumenta as chances dele ser aceito.

## 4.2. QP2: Como são escritos os testes feitos por novatos nos *pull requests*?

Classificar os *pull requests* nas categorias citadas na seção 3.5.1 não foi uma tarefa fácil, pois muitas das informações necessárias estavam ocultas no meio dos códigos alterados e nos comentários das discussões. Procurando pelas informações, percebeu-se que diversos *pull requests* na verdade não se encaixavam nos critérios definidos para análise que eram:

- Ser de um usuário novato no projeto;
- Possuir teste;
- O teste ter sido feito pelo novato.
- O *commit* com teste ter sido integrado ao projeto (*merged*);

Levando esses requisitos em consideração, apenas 40,7% dos *pull requests* analisados manualmente se encaixaram nos critérios. Os outros 59,3% eram na verdade falsos positivos gerados pela análise automática dos *pull requests*. Alguns dos descartados, na verdade não possuíam teste em si, ou o teste não havia sido feito pelo novato, por exemplo. Os *pull requests* que se encaixaram em todos os critérios foram classificados de acordo com algumas categorias. O resultado dessa classificação pode ser visto nos gráficos das Figuras 4.1 e 4.2, onde a primeira mostra que apenas 21,6% dos novatos precisaram que um membro do projeto pedisse explicitamente para que ele fizesse testes. Já a segunda mostra que 81,1% dos usuários eram novatos apenas no projeto em questão, mas não era o seu primeiro no *GitHub*.

Perceba que a porcentagem de usuários que estavam no seu primeiro *pull request* no *GitHub* (18,9%) é muito próxima da porcentagem de novatos que necessitaram de um pedido para poder realizar o teste (21,6%). Esses fatos podem ser relacionados, já que desde

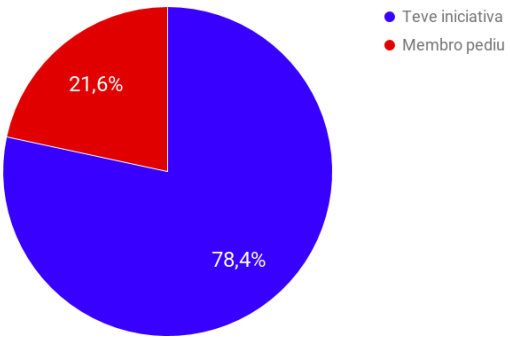


**Figura 4.1.** Iniciativa de fazer teste

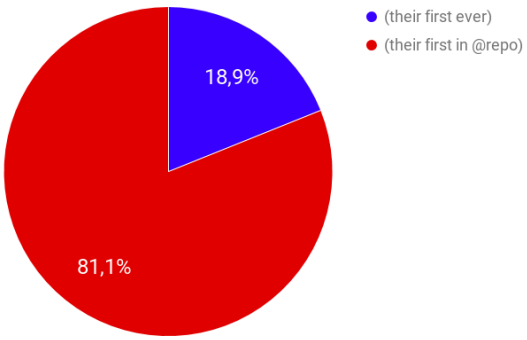


**Figura 4.2.** tentativa na vida

a primeira vez que o usuário faz *pull request* seja exigido testes para se considerar a aceitação, na sua próxima tentativa, ele já estará sabendo desse fato e fará o teste por ele mesmo, sem a necessidade de um outro usuário o avisar.

## 4.3. QP3: Quais são as dificuldades que os contribuidores enfrentam em relação a testes?

O questionário foi elaborado como descrito na seção 3.5.2, e as respostas foram recebidas pelo *Google forms*[1]. A partir das 4 primeiras questões, onde a pessoa podia escolher apenas uma das opções de resposta, foram gerados os gráficos das Figuras 4.3, 4.4, 4.5 e 4.6.

A questão 1 se referia a quantos dos *pull requests* feito pelo usuário foram aceitos. 51,2% disseram que tiveram mais de 15 aceitações. Isso se justifica pois, apesar do questionário ter sido enviado apenas para aqueles que foram identificados como novatos, ele era novato no projeto em questão, mas não necessariamente um novato no *GitHub*, podendo já ter feito *pull requests* em outros repositórios. Quando a pergunta foi em relação a quantos desses *pull requests* aceitos possuíam teste (questão 2), os valores se inverteram completamente, e 46,3% respondeu que menos de 5 desses *pull requests* possuíam teste. Esse fato é facilmente visto ao se observar os gráficos nas Figuras 4.3 e 4.4, onde no primeiro, a maior fatia se refere aqueles que tiveram mais de 15 *pull requests* aceitos, enquanto no segundo, a opção de mais de 15 *pull requests* possuírem teste é a menor das fatias.

A questão 3 perguntava ao usuário se, quando ele fez teste, foi por iniciativa própria, ou porque algum membro do projeto pediu para que ele assim fizesse. 65,9% responderam que tiveram a iniciativa em escrever o teste durante a sua primeira contribuição. É possível observar a incrível semelhança entre o gráfico da Figura 4.5, representando a resposta dos usuários no questionário, e o gráfico da Figura 4.1, representando as informações observadas e coletadas manualmente. Ambos se referindo a mesma questão, confirmam um ao outro. Quando perguntado aos usuários se eles já conheciam algum *framework* de teste antes da sua primeira contribuição (questão 4), apenas 14,6% relatou não ter esse conhecimento.

[1] Formulário: <https://forms.gle/UBZaUKF8KNbZwmdx9>

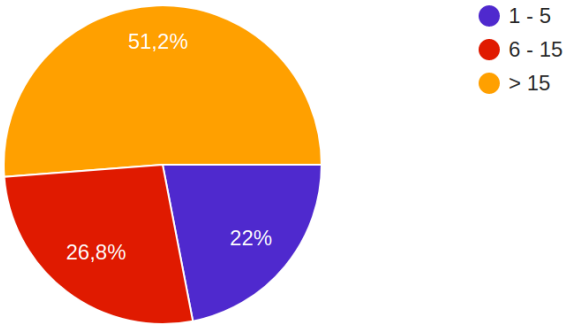


**Figura 4.3.** Questão 1:
Quantas das suas contribuições foram aceitas em projetos do *GitHub*?

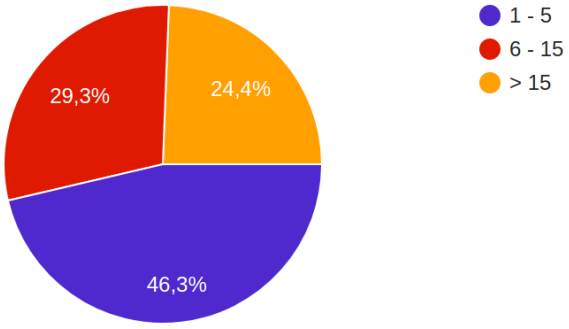


**Figura 4.4.** Questão 2:
Quantas das suas contribuições aceitas em projetos do *GitHub* possuíam teste?

Uma porcentagem bem menor dos usuários que não realizaram teste, afirmaram desconhecer *frameworks* de teste na ocasião de seu primeiro *pull request*. Ou seja, muitos, apesar de saber da existência de *frameworks* que auxiliam na escrita de teste, não os utilizaram.

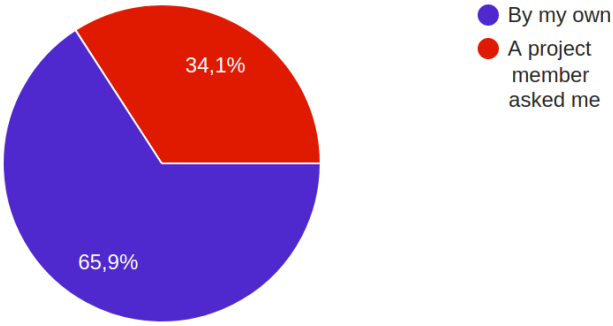


**Figura 4.5.** Questão 3:
Você escreveu testes por conta própria (em azul), ou por que algum membro te pediu(em vermelho)?

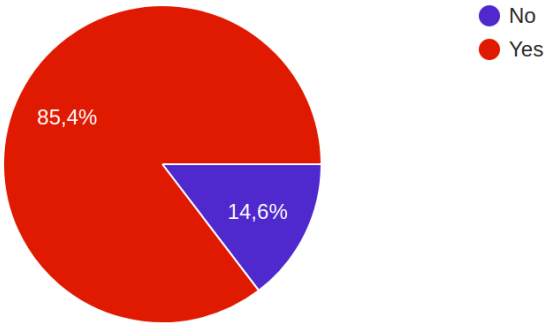


**Figura 4.6.** Questão 4:
Você conhecia *frameworks* de teste antes da sua primeira contribuição?
(sim:vermelho, não:azul)

A quinta questão, sobre o que ajudou na hora de escrever teste, permitia mais de uma opção de resposta ser escolhida. Como pode ser observado na Figura 4.7, 32 dos 41 usuários que responderam ao questionário, disseram que algo que os ajudou a teste foi a documentação. Em segundo lugar temos a integração contínua, seguida pela ajuda de membros/revisores do projeto e de discussões no *Stack Overflow*. A opção outros recebeu diversas respostas que puderam ser agregadas no tópico Testes existentes. *Bots* e outras opções receberam 2 votos cada um e o *Reddit* não recebeu nenhuma.

A sexta e última questão de fato, pedia para o usuário deixar a sua opinião sobre quais os problemas e desafios um usuário enfrenta ao fazer o seu primeiro teste. Apesar de essa questão não ser obrigatória, 34 dos 41 participantes deixaram uma resposta. Diversos usuários relataram que uma das dificuldades é saber o que testar. *"Entender os propósitos do testes, sabendo o que testar"*[traduzida]. Outro usuário ainda completou que *"É fácil encontrar um*

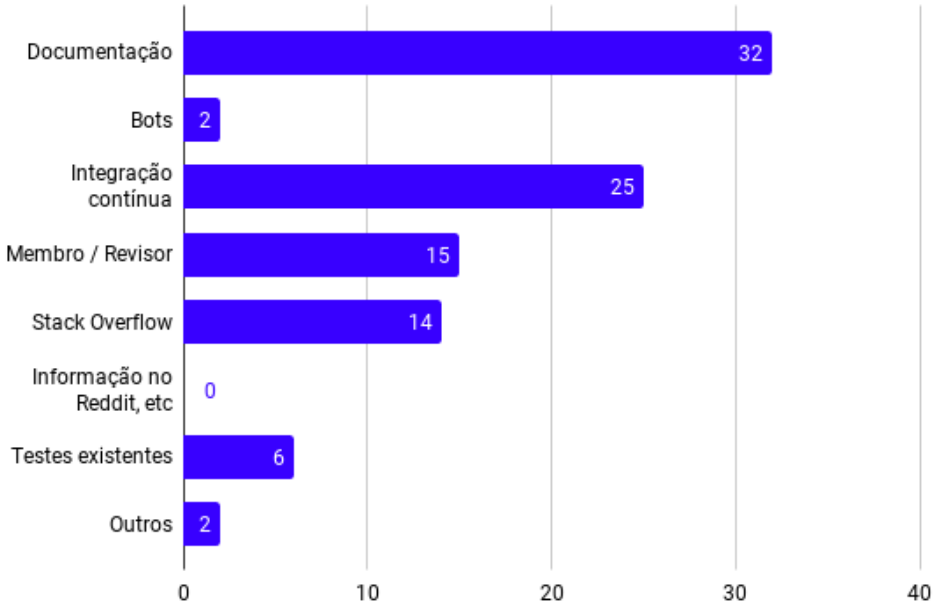


**Figura 4.7.** Questão 5: O que te ajudou quando você estava escrevendo teste? *(mais de uma opção podendo ser selecionada)*

*arquivo de teste, copiar e escrever coisas nele [...]. É difícil ter certeza se você está testando tudo que deveria"*[traduzida]. O assunto *framework* também foi lembrado por um usuário que disse que *"[...]uma das dificuldades foi entender como eles funcionam".*[traduzida].

## 4.4. Ameaças à Validade

Uma ameaça à validade interna que este trabalho apresenta é que assumimos que os repositórios utilizam a convenção de especificar que o arquivo é de teste já no seu nome, então não analisamos o conteúdo dos arquivos. Desse modo, podemos deixar passar vários testes, pois dependemos por repositórios utilizarem essa convenção. A análise manual dos *pull requests* também pode inserir vieses, pois além de ser um ser humano sujeito à falhas, ele procura por algumas coisas específicas nas análises, podendo ocultar da visão, possíveis informações úteis. Utilizar opiniões pessoais também pode não ser uma forma muito segura de coletar dados. Fica cada vez mais difícil controlar o ambiente e assegurar a precisão das respostas.

Capítulo 5

# Conclusões

Diante dos resultados obtidos, podemos concluir que o uso de teste varia bastante de uma linguagem para outra, embora quando há a presença de teste em um *pull request*, as suas chances de ser aceito aumentam. Muitos dos novatos nos projetos não são novatos no *GitHub*, pois já contribuíram em outros projetos anteriormente, de forma que provavelmente já passaram pela experiência de escrever o seu primeiro teste. Isso implica que a maioria do novatos na verdade escreve o teste por iniciativa própria, sem precisar que algum membro ou revisor peça a ele para fazê-lo.

De forma geral, a prática de teste provou-se não ser tão comum como deveria. A resposta da primeira questão de pesquisa (QP1), para qual o percentual de novatos que submetem *pull requests* com teste, nos mostrou que apesar dos *pull requests* com teste serem a minoria na maior parte das linguagens, quando presentes, a aceitação desse *pull request* é altíssima. Isso prova que a presença de teste realmente influencia. Na segunda questão (QP2), sobre como são escritos os testes feitos por novatos nos *pull requests*, vimos que somente uma minoria dos usuários estavam escrevendo o *pull request* pela primeira vez no *GitHub*, sendo novatos apenas aquele projeto. Da mesma forma, a grande maioria doa usuários escreveu o teste por iniciativa própria. Ou seja, os novatos no projeto geralmente já contribuíram em outros projetos antes. Já tendo passado por esse experiência antes, já sabiam que deveriam escrever testes, e assim o fizeram. Em relação às dificuldades que os contribuidores enfrentam em relação a testes, referente à terceira questão de pesquisa (QP3), a partir das respostas dos próprios usuários, pudemos observar que muitos deles na verdade escrevem menos teste do que deveriam, mesmo tendo consciência da existência de *frameworks* de teste que os ajudam nessa tarefa. Em relação as suas dificuldades, muitos sobre com o dilema de saber o que testar, e de qual a melhor maneira de fazer isso. A maioria busca esse auxílio em documentações, mostrando a importância de se manter uma boa documentação de um projeto.

Para trabalhos futuros, é possível aprofundar mais nas razões do por que os

usuários sentem tanta dificuldade em saber o que testar. Também será válido investigar as documentações dos projetos, de forma a perceber o quanto elas podem ajudar um usuário que a procure em busca de instruções, sobre como realizar testes no projeto. Um outro trabalho futuro poderia ser tentar encontrar uma forma de fazer com que mais *pull requests* passem a ter testes desde a sua primeira concepção, aumentando assim, as chances de ser aceito, bem como desperdiçar menos tempos dos membros e revisores do projeto, bem como do próprio usuário.

# Referências


ATLASSIAN. *Jira Software*. 2018. Disponível em: <https://www.atlassian.com/software/jira>.

BECK, Kent. *Test-driven development : by example*. Boston: Addison-Wesley, 2003. ISBN 978-0321146533.

BORLE, Neil C; FEGHHI, Meysam; STROULIA, Eleni; GREINER, Russell; HINDLE, Abram. Analyzing the effects of test driven development in GitHub. v. 23, p. 1931–1958, 2018. Disponível em: <https://doi.org/10.1007/s10664-017-9576-3>.

DELAMARO, Marcio. *Introducao ao teste de software*. Rio de Janeiro: Elsevier, 2007. ISBN 978-85-352-2634-8.

DYER, Robert; NGUYEN, Hoan Anh; RAJAN, Hridesh; NGUYEN, Tien N. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In: *Proceedings of the 35th International Conference on Software Engineering*. [S.l.: s.n.], 2013. (ICSE'13), p. 422–431.

GITHUB. *About*. 2018. Disponível em: <https://github.com/about>.

GITHUB. *GitHub Glossary*. 2018. Disponível em: <https://help.github.com/articles/github-glossary/\#pull-request>.

GOUSIOS, Georgios. The ghtorrent dataset and tool suite. In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE Press, 2013. (MSR '13), p. 233–236. ISBN 978-1-4673-2936-1. Disponível em: <http://dl.acm.org/citation.cfm?id=2487085.2487132>.

GUIDE, GitHub. *How to Contribute to Open Source*. 2018. Disponível em: <https://opensource.guide/how-to-contribute>.

HUNT, Andrew. *Pragmatic unit testing : in Java with JUnit*. Raleigh, N.C: Pragmatic Bookshelf, 2004. ISBN 0-9745140-1-2.

KAVANAGH, Paul. 1 – Open Source Software: Definitions and History. In: *Open Source Software*. [S.l.: s.n.], 2004. cap. 1, p. 1–17. ISBN 9781555583200.

MCINTOSH, Shane; KAMEI, Yasutaka; ADAMS, Bram; HASSAN, Ahmed E. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. New York, NY, USA: ACM, 2014. (MSR 2014), p. 192–201. ISBN 978-1-4503-2863-0. Disponível em: <http://doi.acm.org/10.1145/2597073.2597076>.

MUNAIAH, Nuthan; KROH, Steven; CABREY, Craig; NAGAPPAN, Meiyappan. Curating GitHub for engineered software projects. *Empirical Software Engineering*, 2017. ISSN 15737616.

MYERS, Glenford. *The art of software testing*. Hoboken, N.J: John Wiley & Sons, 2012. ISBN 978-1118031964.

PHAM, Raphael; SINGER, Leif; LISKIN, Olga; FILHO, Fernando Figueira; SCHNEIDER, Kurt. Creating a shared understanding of testing culture on a social coding site. In: *Proceedings - International Conference on Software Engineering*. [S.l.: s.n.], 2013. ISBN 9781467330763. ISSN 02705257.

PINTO, Gustavo; STEINMACHER, Igor; GEROSA, Marco Aurélio. More Common Than You Think: An In-depth Study of Casual Contributors. In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. [S.l.: s.n.], 2016. ISBN 978-1-5090-1855-0.

RAYMOND, Eric. The cathedral and the bazaar. *Knowledge, Technology & Policy*, Springer Nature, v. 12, n. 3, p. 23–49, sep 1999. Disponível em: <https://doi.org/10.1007/s12130-999-1026-0>.

SPADINI, Davide; ANICHE, Maurício; STOREY, Margaret-Anne; BRUNTINK, Magiel; BACCHELLI, Alberto. When testing meets code review. In: *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*. [S.l.: s.n.], 2018. ISBN 9781450356381. ISSN 02705257.

STEINMACHER, Igor; WIESE, Igor; CHAVES, Ana Paula; GEROSA, Marco Aurelio. Why do newcomers abandon open source software projects? In: *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2013 - Proceedings*. [S.l.: s.n.], 2013. ISBN 9781467362900.

ZAPPONI, Carlo. *GitHut - A Small Place to Discover Languages in GitHub*. 2014. Disponível em: <https://githut.info/>.