Um Modelo de Classificação de Documentação para Novatos em Projetos de Software Livre

Luiz Felipe Fronchetti Dias

Dissertação apresentada ao
Instituto de Matemática e Estatística
da Universidade de São Paulo
para obtenção do título de
Mestre em Ciências

Programa: Ciência da Computação
Orientador: Prof. Dr. Marco Aurélio Gerosa
Coorientador: Prof. Dr. Igor Steinmacher

Durante o desenvolvimento deste trabalho o autor
recebeu auxílio financeiro da FAPESP (18/02596-1)

São Paulo
01 de Janeiro de 2023

Um Modelo de Classificação de Documentação para Novatos em Projetos de Software Livre

Luiz Felipe Fronchetti Dias

Esta é a versão original da dissertação elaborada pelo candidato Luiz Felipe Fronchetti Dias, tal como submetida à Comissão Julgadora.

Resumo

Luiz Felipe Fronchetti Dias. **Um Modelo de Classificação de Documentação para Novatos em Projetos de Software Livre**. Dissertação (Mestrado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2023.

Contexto: Projetos de software livre representam uma grande parcela do mercado de desenvolvimento de software. Neste contexto, uma diversidade de projetos de software livre contam com contribuições voluntárias para se manterem ativos. Embora importantes para o ecossistema dos projetos, novos contribuidores ao tentarem ingressar voluntariamente em projetos de software livre, tendem a enfrentar um conjunto de barreiras que dificultam o processo de contribuição. Tais barreiras levam novatos a desistirem de contribuir com projetos de software livre. Entre as dificuldades já constatadas, existem barreiras associadas às documentações dos projetos que, muitas vezes, não atendem as necessidades que novos contribuidores têm durante o processo de contribuição.

Objetivo: Almejando contribuir com a avaliação de documentações de projetos de software livre, esta pesquisa se baseia na construção de um modelo de classificação, capaz de identificar trechos em arquivos de contribuição de projetos de software livre que sejam relevantes a novatos durante o processo de entrada em tais projetos. Acreditamos que tal ferramenta possa auxiliar novos contribuidores a compreender aspectos essenciais dos projetos aos que eles almejam contribuir, bem como permitir uma melhor avaliação dos projetos quanto à qualidade das documentações produzidas para novatos.

Método: Um modelo de classificação de trechos de documentação relevantes a novos contribuidores foi implementado a partir da análise qualitativa de arquivos de contribuição encontrados em repositórios de software livre. Utilizando como base algoritmos de aprendizado, este modelo recebe como entrada parágrafos de documentações de projetos, e identifica para cada um dos parágrafos, uma entre seis categorias de informação relevantes a novos contribuidores. O modelo de classificação, bem como os dados e resultados obtidos durante a execução deste método, foram analisados sob diferentes perspectivas a partir de duas etapas, uma de avaliação estatística, e outra através de um questionário aplicado à desenvolvedores de software com e sem experiência em software livre.

Resultados:

Palavras-chave: Software Livre e de Código Aberto. Novatos. Análise de Documentação.

Abstract

Luiz Felipe Fronchetti Dias. **Um Modelo de Classificação de Documentação para Novatos em Projetos de Software Livre**. Thesis (Masters). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2023.

Context: Open source projects constitute a significant part of the software development industry. In this context, for projects to remain active, voluntary contributions are expected. However, new contributors, when voluntarily trying to join open source projects, often face a set of barriers that compromises the contribution process. Such barriers tend to lead newcomers to give up contributing to these projects and, for this reason, are widely studied in the literature. Among the difficulties already noted in the literature, there are barriers associated with the documentation of open source projects that often do not meet the needs that new contributors have during the contribution process.

Objective: Aiming to contribute to the documentation of open source projects, we propose the development of a classification model, capable of identifying snippets in open source projects documentation that are relevant to newcomers. This tool, at the end of this study, will help newcomers to understand essential aspects of the projects they aim to contribute, mitigating potential difficulties faced during the contribution process.

Method: From an experiment to be conducted with newcomers to open source projects, and researchers in software engineering, a model of classification of categories relevant to new contributors will be built. Using machine learning concepts as a fundamental basis, this model will receive as input project documentation paragraphs, and should identify for each paragraph, a set of categories of information relevant to new contributors, which will be defined at a preliminary stage. The classification model, as well as the data and results obtained during the execution of this method, will be analyzed from different perspectives in a statistical evaluation stage, and through questionnaires, to be applied with newcomers and core contributors in the context of open source software.

Expected Results: We hope with this study to answer whether it is possible to identify relevant documentation snippets to new open source contributors using classification algorithms, and to present which features are most relevant for identifying these snippets, and the acceptance of predictions by contributors in this context.

Keywords: Free/Libre and Open Source Software. Newcomers. Documentation Analysis.

Lista de Abreviaturas

FLOSS	Free/Libre Open Source Sofware
NLP	Natural Language Processing
API	Application Programming Interface
YODA	Young NewcOmer Developer Assistant
CODES	Mining SourCe COde Descriptions from DevelopErs DiScussions
USP	Universidade de São Paulo
UFPA	Universidade Federal do Pará

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Projetos de software livre estão presentes em várias áreas do desenvolvimento de software, atendendo a uma parcela significava do mercado consumidor de programas de computador (Bonaccorsi e Rossi, 2003). Muitos projetos neste contexto, para se manterem ativos, contam com contribuições voluntárias de desenvolvedores de código (Crowston *et al.*, 2012). Voluntários, ao tentarem ingressar em projetos de software livre, costumam sofrer uma série de dificuldades, que comprometem a realização de contribuições (I. Steinmacher, Wiese *et al.*, 2014). Tais dificuldades levam, muitas das vezes, novatos a desistirem de contribuir com projetos de software livre, refletindo no enfraquecimento das comunidades.

Entre as dificuldades enfrentadas por novatos, se encontram barreiras associadas a documentação dos projetos de software livre que, muitas das vezes, se encontra obsoleta, incompleta, incoerente ou inconsistente (Briand, 2003; Aghajani *et al.*, 2019). Além do mais, evidências mostram que em muitos casos, as documentações de projetos de software livre, não atendem as necessidades que novos contribuidores têm (I. Steinmacher, Wiese *et al.*, 2014). Tais problemas de documentação, tendem a gerar confusão naqueles que desejam realizar uma primeira contribuição e por esta razão precisam ser solucionados.

Para mitigar problemas de documentações de software, uma série de ferramentas são propostas por cientistas e desenvolvedores. Zhong e Su (2013), por exemplo, sugerem um mecanismo de identificação de erros em documentações, utilizando uma combinação de técnicas de processamento de linguagem natural para encontrar trechos de documentação em códigos fonte que estejam incorretos ou obsoletos. Outro exemplo de trabalho é proposto por Haiduc *et al.* (2010), que estabelece um mecanismo de geração automática de documentação para classes e métodos em Java, visando aprimorar a compreensão de códigos fonte. Até mesmo fora do contexto acadêmico, uma série de projetos relacionados à documentação de software também são propostos, entre eles, o Javadoc[1], que analisa anotações e comentários de código e gera automaticamente páginas de documentação em formato HTML para os códigos analisados.

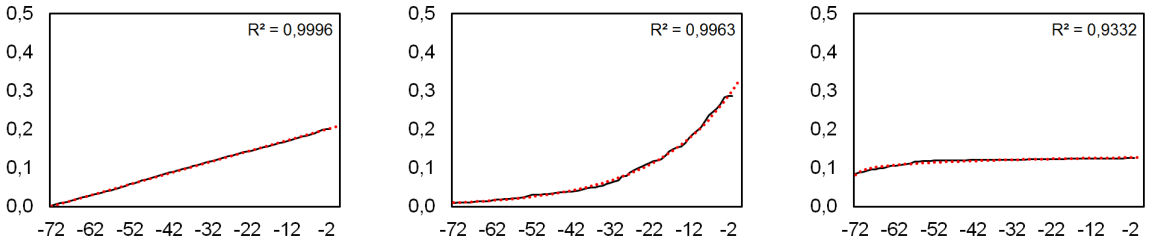
Apesar destes trabalhos contribuírem para documentação de software, nenhuma das ferramentas propostas foca em novatos em software livre. Por esta razão, implementa-

[1] docs.oracle.com/javase/10/javadoc

mos neste estudo a criação de um modelo de classificação que identifique trechos em documentações de projetos de software livre que sejam relevantes a novos contribuidores. Acreditamos que este modelo de classificação deverá não só auxiliar novatos a encontrarem trechos em documentações que sejam úteis durante o processo de contribuição em software livre, como também poderá contribuir com a avaliação da qualidade das documentações já existentes em projetos neste contexto.

1.1 Estudo preliminar

Este estudo dá sequência a uma pesquisa preliminar (Fronchetti *et al.*, 2019), na qual foram investigadas características que influenciam a entrada de novatos em projetos de software livre. Nesta pesquisa preliminar, 15 características de 450 projetos de software livre foram extraídas e analisadas, entre elas, a idade, o domínio de aplicação e o número de linguagens de programação utilizadas em cada um dos projetos selecionados. O algoritmo de clusterização K-Spectral Centroid (KSC) foi utilizado para investigar a taxa de crescimento de novatos nos projetos, e três diferentes padrões de entrada de novatos foram encontrados: logarítmico, exponencial e linear.



(a) *Linear* **(b)** *Exponencial* **(c)** *Logarítmico*

Figura 1.1: *Padrões de entrada de novatos encontrados em projetos de software livre. (Fronchetti et al., 2019).*

Com base nesses padrões de crescimento, um modelo de classificação foi construído a fim de compreender quais fatores poderiam explicar as taxas de entrada de novatos em cada um dos projetos. Foi constatado que fatores como a popularidade dos projetos, o tempo para revisão de submissões, a idade e as linguagens de programação utilizadas nos repositórios de código eram os fatores que melhor explicavam a entrada de novatos no projetos de software livre analisados.

Também foram avaliados fatores dicotômicos associados à documentação dos projetos, tais como a existência de arquivos README e CONTRIBUTING nos repositórios de código fonte, conhecidos por serem fornecedores primários de informação para novatos em software livre. Nenhuma das características relacionadas à documentação dos projetos apresentaram grande influência na atratividade de novatos. A existência de arquivos contendo o código de conduta dos projetos, por exemplo, foi o fator que apresentou a menor influência entre todos os analisados.

Essa baixa influência pode ser considerada uma ameaça à validade deste estudo preliminar, já que as características relacionadas às documentações foram avaliadas de maneira dicotômica, considerando apenas o fato de um determinado arquivo existir ou não. Por

este motivo, surge a necessidade deste novo estudo, cujo objetivo é identificar trechos de documentações que sejam relevantes aos novos contribuidores. Identificar tais trechos, não só pode abrir espaço para futuras ferramentas que auxiliarão novatos, como também pode contribuir para avaliar a qualidade das documentações que projetos de software livre disponibilizam.

1.2 Questões de pesquisa

A metodologia proposta para este estudo é fundamentada a partir de três questões pesquisa, focadas em compreender e avaliar o modelo de classificação construído:

QP1 É possível identificar trechos em documentações de projetos de software livre que sejam relevantes a novos contribuidores utilizando algoritmos de classificação?

QP2 Como diferentes características influenciam as predições feitas pelo classificador de trechos de documentação?

QP3 Qual a percepção dos desenvolvedores de software livre em relação à identificação de trechos de documentação relevantes a novos contribuidores?

1.3 Organização do trabalho

Este projeto de dissertação está organizado em três capítulos. O Capítulo 2 define o conceito de software livre, discutindo barreiras que novatos enfrentam neste contexto, problemas usualmente encontrados em documentação de software, e trabalhos relacionados que buscam solucionar problemas similares aos encontrados nesta pesquisa. O Capítulo 3 apresenta a metodologia proposta para este estudo, descrevendo um conjunto de cinco passos que irão guiar toda a pesquisa a ser executada. Por fim, o Capítulo **??** discute os resultados preliminares obtidos até o momento desta qualificação.

Capítulo 2

Referencial Teórico

Como o objetivo desta pesquisa concentra-se em investigar por meio de aprendizado de máquina a documentação de projetos de software livre, a revisão bibliográfica foi dividida em cinco seções: A Seção 2.1 traz uma breve explicação sobre o que é software livre e como desenvolvedores trabalham e se organizam neste contexto. Na Seção 2.2, é apresentado o conceito de barreiras enfrentadas por novatos em software livre. Na Seção 2.3, são apresentados problemas associados à documentação de projetos de software, dando ênfase naqueles constatados no contexto de software livre. Por fim, na Seção 2.4, são apresentados os trabalhos relacionados a este estudo.

2.1 Software Livre

Software livre (do inglês, *free software*) é o tipo de software que em sua essência é distribuído livremente, acompanhado por uma licença de software, e que conta com a disponibilização de seu código-fonte em algum meio acessível a usuários e desenvolvedores (Cristina Gacek e Budi Arief, 2004). Um programa de computador para ser considerado livre, deve fornecer a quem o utiliza quatro liberdades essenciais (Stallman, 2002):

- A liberdade de executar o programa, para qualquer propósito;
- A liberdade de estudar como o programa funciona, e poder adaptá-lo;
- A liberdade de redistribuir cópias do programa a outros usuários;
- A liberdade de aperfeiçoar o programa, e poder liberar os aperfeiçoamentos a comunidade.

Outro termo geralmente associado a software livre é o conceito de software de código aberto (do inglês, *open source software*), que apresenta certas diferenças[1] em sua definição se comparado a software livre. Um possível antônimo para software livre é o software proprietário, que restringe legalmente a terceiros o acesso ao código fonte do programa desenvolvido (Fuggetta, 2003).

[1] gnu.org/philosophy/open-source-misses-the-point.html

2.1.1 Representatividade

Embora o termo software livre ainda seja desconhecido por muitos usuários, projetos neste contexto têm mostrado grande influência no cenário de desenvolvimento e consumo de software (Höst e Oručević-Alagić, 2011). O servidor Apache é um exemplo de projeto de software livre popularmente conhecido entre desenvolvedores ao redor do mundo. Mantido por voluntários, o Apache hospedava em Julho de 2019[2], mais de 44% dos websites ativos na internet. O código fonte principal do sistema operacional Android é outro exemplo de projeto de software livre que tem apresentado grande adesão de usuários no mercado de software, podendo ser considerado um dos maiores sistemas operacionais para dispositivos móveis do mundo, representou em Setembro de 2019 uma parcela de mais de 76%[3] de adesão em dispositivos móveis.

Não apenas usuários e desenvolvedores têm demonstrado interesse em software livre, evidências mostram que diversas empresas também se encontram alinhadas a esta mesma filosofia de liberdade (Pinto, Igor Steinmacher *et al.*, 2018). A linguagem de programação Swift[4], desenvolvida pela Apple, é um exemplo de programa de computador que por muito tempo foi proprietário, e que hoje faz parte do contexto de software livre. Até mesmo a Microsoft, reconhecida por sua popularidade com projetos de código fechado como Windows e Office, lançou em Novembro de 2015 uma versão aberta do seu editor de texto, o Visual Studio Code[5]. Essas companhias parecem não só estarem interessadas em tornar disponíveis publicamente o código fonte de seus projetos, como também tem mostrado interesse em contribuir com o cenário de software livre, pagando desenvolvedores para trabalharem em projetos neste contexto (Pinto, Dias *et al.*, 2018).

2.1.2 Funcionamento

Geralmente, um novo projeto de software livre se inicia quando um programador possui uma ideia, problema ou porção de código a ser desenvolvido (Raymond, 1999). O programador submete uma proposta a comunidades colaborativas de software, anunciando seu projeto a outros desenvolvedores. Tais desenvolvedores, quando atraídos pelo projeto proposto, tendem a buscar por informações a respeito do projeto, e passam a participar do desenvolvimento dos códigos. Com o tempo e com a adesão de novos contribuidores, uma nova comunidade de software livre se forma em torno do projeto. Plataformas de codificação passam a ser utilizadas como meio de gerenciamento do código fonte, e canais de comunicação são estabelecidos entre os membros da comunidade.

Projetos neste contexto usualmente são mantidos a partir de contribuições de desenvolvedores internos e externos às comunidades de software livre. Para contribuir com um projeto, um desenvolvedor externo adquire uma cópia do código fonte através de uma plataforma de codificação, implementa modificações e submete as mudanças novamente ao repositório de código do projeto (Audris Mockus *et al.*, 2000). As modificações são revisadas por um grupo de mantenedores e, quando estão de acordo com os critérios de aceitação, são anexadas aos binários do programa. Desenvolvedores interessados em

[2] w3techs.com/technologies/overview/web_server/all
[3] https://gs.statcounter.com/os-market-share/mobile/worldwide
[4] developer.apple.com/swift/blog/?id=34
[5] github.com/microsoft/vscode/issues/60

participar de uma comunidade de software livre tendem a construir sua reputação por meio de contribuições ao código fonte dos programas desenvolvidos, de onde com o tempo, passam a ganhar espaço e notoriedade (Krogh *et al.*, 2003).

2.1.3 Organização

Em relação à organização dos projetos, Nakakoji *et al.* (2002) sintetizam a estrutura social de comunidades de software livre a partir de um modelo de camadas. Segundo os autores, nas camadas mais internas se encontram os principais desenvolvedores, aqueles que geralmente contribuem com a maior parte do código fonte e são responsáveis por revisar futuras contribuições. Ao meio da estrutura, se encontram membros do projeto que são responsáveis por contribuir com pequenas correções e aprimoramentos. E por fim, nas camadas mais externas, se encontram diferentes níveis de usuários, que não contribuem com o código fonte do projeto, mas que eventualmente relatam problemas e dão sugestões, além de desfrutar dos recursos oferecidos pelo programa. A representação do modelo de camadas é apresentada na Figura 2.1.

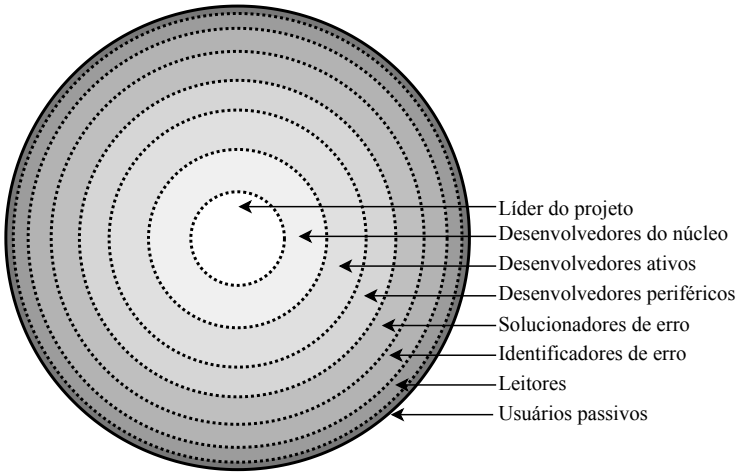


Figura 2.1: *Estrutura em camadas de uma comunidade de software livre (Nakakoji et al., 2002)3*

Além dos grupos apresentados pelo modelo de Nakakoji *et al.* (2002), existem ainda os novos contribuidores que, apesar de não necessariamente constituírem parte da hierarquia de um projeto, por serem considerados essenciais ao desenvolvimento dos projetos, são amplamente estudados pela literatura em software livre. Neste contexto, um novo contribuidor, novato ou recém-chegado, pode ser definido como um desenvolvedor interessado em ingressar em uma comunidade de software livre por meio de contribuições. Entre as razões pelas quais novatos são amplamente estudados pela literatura, se encontram as dificuldades que estes enfrentam ao tentarem ingressar em projetos de software livre, como mostra a seção a seguir.

2.2 Barreiras enfrentadas por novatos

Como as contribuições em software livre costumam ser voluntárias, o ato de atrair novos contribuidores para tais comunidades é considerado fator essencial para continuidade

de muitos projetos (Qureshi e Fang, 2011). Como em qualquer atividade coletiva, se membros ao saírem de uma comunidade não forem substituídos por novos contribuidores, a comunidade irá eventualmente deixar de existir, já que novatos podem ser considerados fonte de inovação, ideias e trabalho (Kraut *et al.*, 2012).

No entanto, atrair novos contribuidores para comunidades de software livre não é uma tarefa trivial (Zhou e A. Mockus, 2015). Em um estudo sobre empresas que compartilham comunidades neste contexto, Dahlander e Magnusson (2008) mostram que até mesmo para comunidades bem estabelecidas como o MySQL, diversos problemas são enfrentados ao atrair novos contribuidores. Entre os problemas que justificam a baixa adesão de novatos a projetos de software livre estão um conjunto de barreiras enfrentadas por novatos durante o processo de entrada e integração a tais projetos.

Krogh *et al.* (2003) descrevem as barreiras enfrentadas por novatos como um conjunto de dificuldades, empecilhos ou deficiências no processo de contribuição que atrapalham desenvolvedores a realizarem suas primeiras colaborações. De acordo com os autores, as barreiras estão intrinsecamente relacionadas ao nível de especialização de cada desenvolvedor. Como a complexidade de um projeto tende a crescer conforme seu desenvolvimento, ingressar em meio à sua execução tende a ser um processo lento e dificultoso a novos desenvolvedores que, por muitas vezes, desconhecem os processos e tecnologias utilizadas pela comunidade.

Hannebauer e Gruhn (2017) dividem barreiras enfrentadas por novatos em duas categorias, submissão e modificação. De acordo com os autores, novatos tendem a enfrentar dificuldades tanto durante a elaboração de uma nova contribuição, como durante o processo de submissão do código modificado ao projeto. Em uma perspectiva complementar, I. Steinmacher, Wiese *et al.* (2014) mostram que tais barreiras não somente estão associadas às motivações e características do desenvolvedor ao contribuir com um projeto, como também são desencadeadas por problemas da própria comunidade à qual ele almeja participar. Problemas como falta de documentação e a má recepção fornecida a novos contribuidores são exemplos de problemas que podem impactar a entrada dos novatos em comunidades de software livre.

Entre os resultados dos estudos que sintetizam as dificuldades enfrentadas por novatos em software livre se encontra o modelo de barreiras proposto por I. Steinmacher, Chaves *et al.* (2014). Dividido em seis categorias, o modelo apresenta cinquenta e oito barreiras enfrentadas por novos contribuidores. Entre as categorias, foram constatadas dificuldades relacionadas ao processo de recepção dos novatos, às características dos próprios desenvolvedores, à falta de orientação durante o processo de contribuição, aos problemas com documentação, aos problemas associados às diferenças culturais e aos obstáculos técnicos. A Figura 2.2 apresenta uma tradução livre do modelo de barreiras proposto pelos autores. Vale ressaltar, que tal modelo serviu de inspiração para elaboração de categorias relevantes a novatos em software livre utilizadas neste estudo, e apresentadas na Tabela 3.1.

O fato de tais comunidades não atraírem novos contribuidores o suficiente usualmente implica na falta de condições necessárias para se manter o desenvolvimento dos projetos ativo, levando, por consequência, muitas comunidades de software livre a inatividade. Por esta razão, é tido como proposta deste estudo a elaboração de uma metodologia que contribua com problemas que novatos enfrentam em relação às documentações de projetos

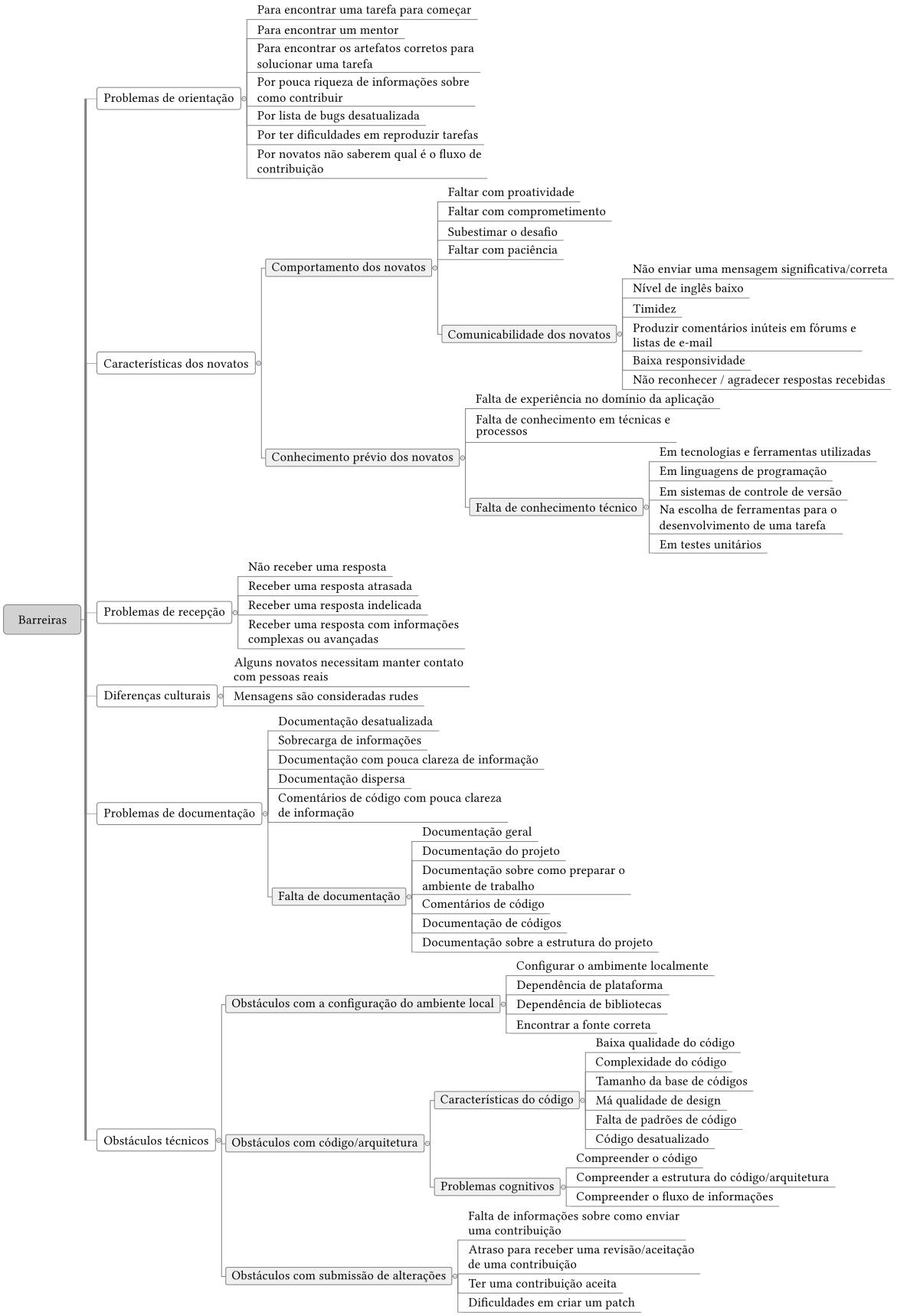


Figura 2.2: *Modelo de barreiras enfrentadas por novatos em projetos de software livre (I. Steinmacher, Chaves et al., 2014).*

de software livre, de modo com que algumas das barreiras apresentadas por I. Steinmacher, Chaves *et al.* (2014), e demais trabalhos relacionados, sejam reduzidas ou deixem de existir. Na seção a seguir, uma visão geral sobre problemas enfrentados em documentações de software é apresentada.

2.3 Problemas em documentações de software

A documentação de um programa de computador, quando correta, completa e consistente, tende a ser uma fonte crucial de informações para desenvolvedores (Kajko-Mattsson, 2005). O propósito de se documentar um programa é contribuir com a preservação dos processos desenvolvidos durante sua elaboração, de modo com que futuras modificações possam ser feitas (Aghajani *et al.*, 2019). Ainda que a elaboração de uma documentação possa ser custosa, a preservação do entendimento de um sistema pode apresentar diversos benefícios ao seu desenvolvimento, tendo reflexo, por exemplo, na produtividade dos desenvolvedores, na redução dos custos de manutenção e na extensão da vida útil do sistema desenvolvido (Zhi *et al.*, 2015; Souza *et al.*, 2005).

No entanto, apesar dos benefícios, evidências mostram que documentações de programas de computador costumam apresentar uma diversidade de problemas (Briand, 2003). Em muitos casos, as documentações se encontram incorretas, obsoletas, incompletas ou incoerentes, o que pode comprometer o entendimento do programa por parte dos desenvolvedores (Aghajani *et al.*, 2019; Forward e Lethbridge, 2002). Problemas de documentação não se restringem a uma única área de aplicação e podem estar presentes em diversos contextos do desenvolvimento de software.

Ao investigar problemas em documentações de API's, Uddin e Robillard (2015) dividem tais problemas em duas categorias, conteúdo e apresentação. Documentações obsoletas, incoerentes ou incompletas são exemplos de problemas associados à categoria conteúdo, que visa compreender problemas relacionados às informações contidas em cada documentação. A divisão desnecessária do texto em páginas ou seções e o excesso de informações sobre um único tópico ou elemento são exemplos de problemas associados à categoria apresentação, que abrange a estrutura e o modo como as documentações são expostas a desenvolvedores.

Além das categorias mencionadas, Robillard (2009) apresenta outras três classes de problemas associados a documentações de software: experiência, ambiente técnico e processo. Na classe experiência, o autor identifica que muitos dos obstáculos enfrentados com documentações de software são provenientes da falta de experiência prévia do desenvolvedor, que muitas vezes desconhece os conceitos fundamentais do sistema e que, por consequência, passa a enfrentar problemas com as documentações. Os diferentes ambientes de desenvolvimento nos quais um programador trabalha são apresentados como outro fator gerador de obstáculos em documentações de software, que muitas vezes costumam apresentar informações apenas para um único tipo de sistema. Por fim, o tempo e a maneira como os desenvolvedores se dedicam a compreender uma determinada documentação também são apresentados como fatores geradores de obstáculos, já que influenciam indiretamente no processo de entendimento da documentação por parte dos desenvolvedores.

No contexto de software livre, onde a documentação dos projetos é crucial para integração de novatos às comunidades, diversos problemas também são enfrentados (C. Gacek e B. Arief, 2004; Aberdour, 2007). Além de problemas associados às categorias mencionadas anteriormente, I. Steinmacher, Chaves *et al.* (2014) identificam em seu modelo de barreiras outros tipos de problemas com documentação, tais como a falta de clareza nas informações dispostas, a dispersão das informações sobre diferentes meios de comunicação e a sobrecarga de informações a respeito de um mesmo tópico.

2.4 Trabalhos Relacionados

Nesta seção, são apresentados trabalhos que buscam solucionar problemas similares aos atacados neste estudo, incluindo o apoio a entrada de novatos em software livre e a automatização da busca por trechos em documentação de software. Para cada tópico a seguir, são apresentados trabalhos encontrados na literatura e trabalhos provenientes de fora do contexto acadêmico.

2.4.1 Apoio a novatos em software livre

Na literatura, Malheiros *et al.* (2012) propõem uma ferramenta denominada *Mentor* que, através de algoritmos de recomendação, sugere a novos contribuidores arquivos do código fonte de projetos de software livre que podem contribuir com a realização de uma determinada tarefa. Tal ferramenta, facilita a busca de arquivos que estejam associados à, por exemplo, solução de um erro no programa desenvolvido. Outra ferramenta similar é proposta por Cubranic *et al.* (2005), que também utiliza de algoritmos de recomendação para sugerir a novos contribuidores arquivos do código fonte de projetos que possam contribuir na execução de uma tarefa. A diferença é que nesta ferramenta não são sugeridos apenas arquivos de código fonte, mas também trechos provenientes de outras fontes, como documentações dos projetos, canais de comunicação, relatórios de erro e planos para teste.

Buscando auxiliar novatos na escolha de uma primeira tarefa, Wang e Sarma (2011) propõem um mecanismo inteligente de busca por erros não resolvidos em projetos de software livre. De acordo com os autores, uma boa maneira de ingressar em um projeto de software livre se encontra na resolução de erros do programa desenvolvido. Encontrar erros que se adéquem às características técnicas e de interesse do novo contribuidor é um fator que deve ser levado em consideração, já que pode ser primordial na integração do novato a projetos neste contexto. O sistema proposto tem acesso à lista de erros de um projeto e possibilita que desenvolvedores encontrem erros que desejam solucionar por meio de buscas inteligentes, além de identificar erros similares àqueles que os novatos demonstrem interesse.

Engajados com a orientação de novatos em software livre, Canfora *et al.* (2012) descrevem uma abordagem de auxílio a novatos denominada YODA[6], que objetiva identificar e recomendar mentores para novos contribuidores em projetos de software livre. Por meio de algoritmos de recomendação e utilizando dados extraídos de listas de e-mail e sistemas

[6] spanichella.github.io/tools.html#yoda

de controle de versão, o YODA recomenda um mentor através de uma análise sistemática do histórico de contribuição e mentoria de cada desenvolvedor interno em um determinado projeto, que é associado pelo próprio sistema a um novato interessado em contribuir com a comunidade.

As ferramentas descritas acima tem relação com o modelo de barreiras proposto por I. Steinmacher, Chaves *et al.* (2014), que também propõem como contribuição de seu trabalho, uma ferramenta para auxiliar novatos em software livre. Conhecida como FLOSSCoach[7], tal ferramenta é uma plataforma de auxílio a novos contribuidores, que visa orientar novatos entre os principais caminhos que devem ser seguidos para realização de uma contribuição em projetos de software livre. Tais caminhos foram definidos a partir das principais dificuldades que novatos enfrentam neste contexto.

Além de trabalhos na literatura, uma diversidade de ferramentas também são propostas por desenvolvedores de fora do contexto acadêmico. O Up For Grabs[8], por exemplo, é um website que tem como objetivo auxiliar novatos a encontrarem tarefas disponíveis em projetos de software livre. Os novatos acessam o website que apresenta a eles, uma série de projetos com tarefas nos sistemas de caça tarefas (do inglês, *issue tracker*) específicas para novos contribuidores, como mostra a Figura 2.3. Outro exemplo de projeto é o First Timers Only[9], um website que estabelece diretrizes para novatos que desejam contribuir com projetos de software livre. Nesta página é possível encontrar uma diversidade de dicas e tutoriais relacionados aos primeiros passos de um novato em projetos neste contexto.d

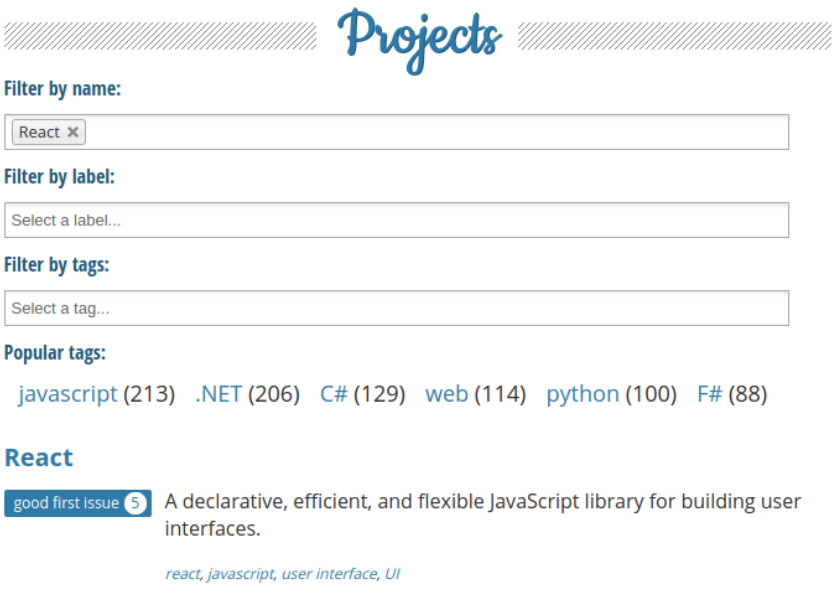


Figura 2.3: *Exemplo de projeto cadastrado no Up For Grabs. React é o nome do projeto, e good-first-issue é o rótulo atribuído as tarefas destinadas a novatos.*

[7] flosscoach.com
[8] up-for-grabs.net
[9] firsttimersonly.com

2.4.2 Suporte a documentação de software

Uma variedade de trabalhos relacionados também são propostos quanto a solução para problemas de documentação tais como os apresentados na Seção 2.3. Zhong e Su (2013) propõem um mecanismo de identificação de erros em documentações de API's. Utilizando uma combinação de técnicas de processamento de linguagem natural e de análise de códigos fonte, os autores apresentam uma ferramenta para identificar trechos de documentação incoerentes e obsoletos. A ferramenta proposta, denominada DocRef, também identifica erros gramaticais. Tal ferramenta foi submetida a análise de cinco API's popularmente conhecidas e, de acordo com os autores, já reportou a mantenedores das cinco bibliotecas mais de mil erros de documentação.

Focados no processo de manutenção de programas escritos em Java, Haiduc *et al.* (2010) propõem uma ferramenta para gerar sumários de classes e métodos em códigos fonte. Feita através de técnicas de sumarização de texto, a ferramenta visa facilitar o entendimento do código fonte de projetos Java, gerando um conjunto de palavras que descrevam corretamente as características principais de cada trecho de código. De acordo com os autores, quando desenvolvedores necessitam dar manutenção a uma determinada funcionalidade de um programa, dicas textuais podem auxiliar estes a compreenderem quais parte do código eles devem investigar, sem necessariamente gastarem um longo tempo compreendendo cada parte do projeto.

Em um estudo similar, Li *et al.* (2016) desenvolvem uma abordagem que gera automaticamente documentação para testes de unidade. Considerando a importância deste tipo de teste para manutenção e prevenção de erros em programas de computador, os autores desenvolveram uma ferramenta que analisa, a partir de um conjunto de técnicas, testes de unidade em repositórios de código fonte e preenche modelos pré-definidos de documentação com informações relevantes sobre cada teste de unidade. Tanto neste como no estudo de Haiduc *et al.* (2010), questionários foram aplicados a desenvolvedores, a fim de avaliar a qualidade das documentações produzidas. Ambas as ferramentas tiveram grande aceitação por parte dos desenvolvedores, que compreendem o impacto da documentação de códigos na manutenção de projetos de software.

No contexto de software livre, Panichella (2015) apresenta dois sistemas de recomendação que visam dar suporte a novatos em software livre. O primeiro, já mencionado anteriormente, é a ferramenta YODA, que recomenda mentores a novatos por meio da análise de dados dos projetos, como listas de e-mail e históricos de contribuição. E o segundo, nomeado CODES[10], é um sistema de recomendação que sugere comentários para códigos Java, utilizando como fonte de dados, discussões entre desenvolvedores em meios de comunicação, tal como listas de e-mail e mecanismos de caça tarefa. Segundo os autores, sugerir mentores não é suficiente para dar suporte a novatos em software livre, auxiliá-los durante a compreensão e re-documentação dos projetos também são passos importantes na integração dos novatos a comunidades de software livre.

Fora do contexto acadêmico uma diversidade de projetos relacionados a este estudo dá suporte a documentações de software. Uma categoria comum são os geradores de

[10] spanichella.github.io/tools.html#codes

documentação. O Javadoc[11], por exemplo, analisa anotações e comentários em códigos Java, e gera automaticamente páginas HTML contendo informações sobre cada classe, método, interface e construtor desenvolvido. Como o Javadoc é específico para linguagem Java, é comum que outras linguagens apresentem outras alternativas, como o Pydoc[12], para linguagem Python, e o YARD[13], para Ruby. É valido mencionar que, além dos geradores de documentação, outra categoria comum são as plataformas de hospedagem, tais como a Readthedocs[14] e a Stoplight[15], que fornecem modelos de documentação para que desenvolvedores organizem seus projetos online. A Figura 2.4 apresenta um exemplo de documentação gerada automaticamente pela ferramenta Javadoc.

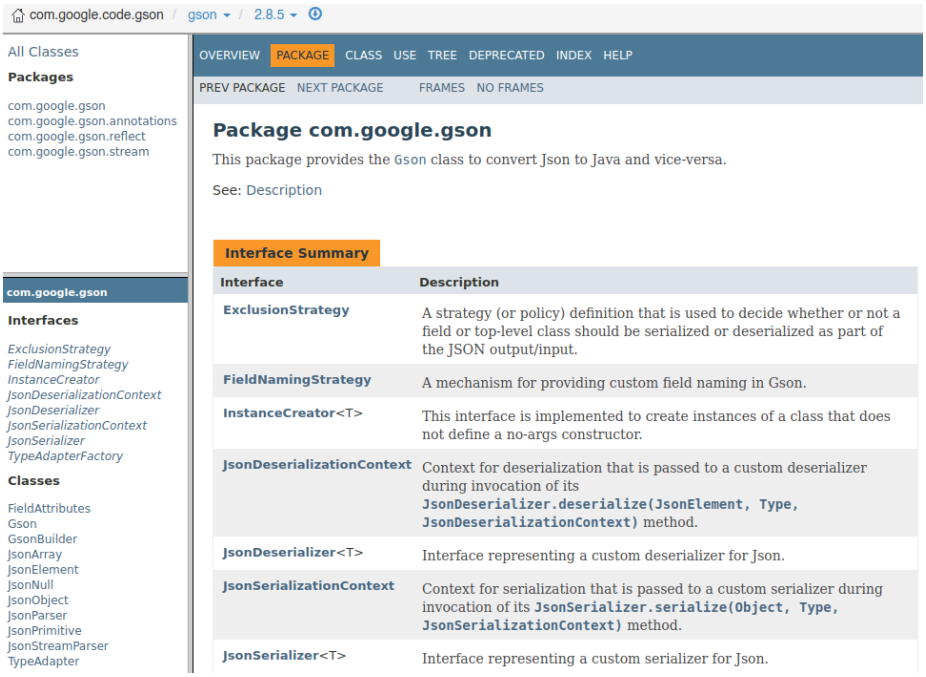


Figura 2.4: *Exemplo de documentação gerada pela ferramenta Javadoc. Gson é o nome da classe contendo anotações e comentários de código.*

[11] docs.oracle.com/javase/10/javadoc
[12] docs.python.org/3/library/pydoc
[13] yardoc.org
[14] readthedocs.org
[15] stoplight.io

Capítulo 3

Metodologia

Levando em consideração as barreiras enfrentadas por novatos em software livre, em especial aquelas relacionadas à documentação dos projetos, propomos por meio desta pesquisa um modelo de classificação que contribua com a identificação de informações em documentações relevantes a novos contribuidores. A metodologia proposta dá continuidade a um trabalho preliminar (Fronchetti *et al.*, 2019), na qual foram investigadas características que influenciam a atratividade de novatos em projetos de software livre.

A elaboração deste método é baseada em dois estudos relacionados, um deles proposto por I. Steinmacher, Chaves *et al.* (2014), que evidenciaram a necessidade de uma ferramenta que auxilie novatos na compreensão de documentações de projetos em de software livre e outro proposto por Panichella *et al.* (2015), que desenvolveram um método de identificação de seções em documentações de software, similar ao apresentado para este estudo.

3.1 Método

Para que seja possível responder as questões de pesquisa apresentadas na Seção 1.2, o método proposto é dividido em um conjunto de cinco etapas. Primeiro, são definidas categorias de informação a serem identificadas em arquivos de contribuição de projetos de software livre. Em seguida, documentações de um conjunto de projetos são extraídas como amostra, e uma análise qualitativa é executada para identificar trechos das documentações que sejam relevantes a novos contribuidores. Por fim, o modelo de classificação é desenvolvido e avaliado. A Figura 3.1 apresenta a sequência de atividades executadas no método desta pesquisa, e as seções a seguir descrevem detalhadamente cada uma das etapas executadas.

3.1.1 Definição das categorias

Um conjunto de categorias de informação foram elaboradas para facilitar a identificação de trechos relevantes a novos contribuidores nas documentações de projetos de software livre. Estas categorias foram estabelecidas com base nas barreiras apresentadas pelo modelo de I. Steinmacher, Conte *et al.* (2016) e em demais trabalhos relacionados,

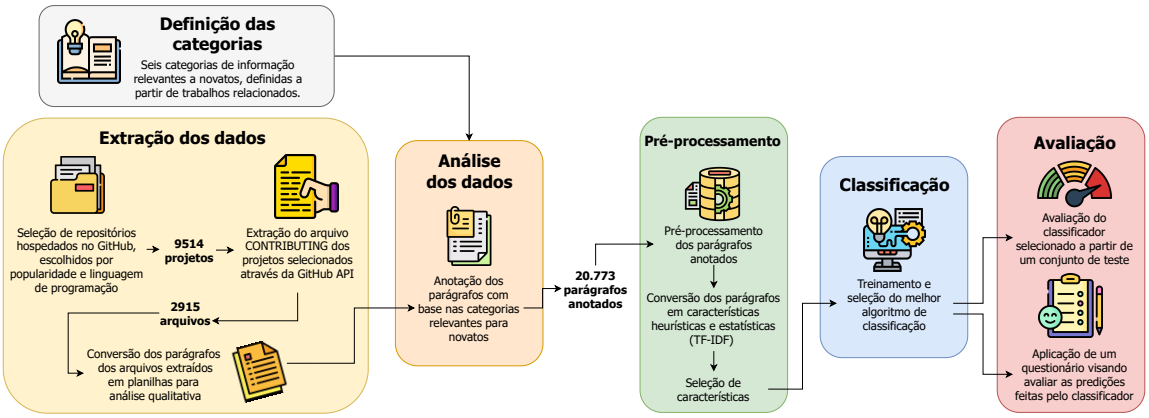


Figura 3.1: *Método de pesquisa.*

que também identificaram dificuldades que novatos enfrentam neste contexto (Krogh *et al.*, 2003; Hannebauer e Gruhn, 2017). Os trabalhos relacionados foram discutidos pelos pesquisadores em reuniões, visando encontrar categorias que estivessem de acordo com as dificuldades que novatos enfrentam. Na Tabela 3.1, apresentamos o conjunto contendo seis categorias que identificamos nas documentações de projetos de software livre. As principais dificuldades enfatizadas nas categorias estabelecidas foram aquelas passíveis de solução por meio da documentação dos projetos. Cada categoria representa a descrição de uma barreira retirada do modelo apresentado por I. Steinmacher, Conte *et al.* (2016) e das informações disponíveis na plataforma de suporte a novatos FLOSSCOACH (Igor Steinmacher *et al.*, 2018), também apresentada pelos autores neste estudo. Estas categorias devem contribuir com o processo de contribuição dos novatos, já que abrangem problemas importantes, como a compreensão do projeto, contato com a comunidade e submissão de mudanças ao código fonte.

3.1.2 Extração dos dados

Seleção dos repositórios

Considerando que a premissa principal deste estudo é conseguir identificar trechos em documentações de projetos de software livre, um conjunto de projetos hospedados na plataforma de codificação GitHub foi selecionado como amostra. Visando maximizar a pluralidade dos projetos escolhidos, e buscando escapar de repositórios vazios ou sem atividade na plataforma, a seleção de projetos foi feita a partir da popularidade e linguagem de programação de cada repositório. Para compor nossa amostra, os projetos mais populares com predominância nas seguintes linguagens de programação foram escolhidos: JavaScript, Python, Java, PHP, C#, C++, TypeScript, Shell, C, e Ruby. A escolha de tais linguagens de programação se baseia em dados do evento GitHub Octoverse (GitHub, 2020), que lista anualmente as linguagens de programação masi populares na plataforma. Já a filtragem por popularidade se baseou no número de estrelas de cada repositório no GitHub que, de acordo com Borges *et al.* (2016), pode servir como métrica para definir quão popular os projetos são. Um total de 9.514 repositórios foram selecionados para compor nossa amostra por meio da GitHub API, um serviço que fornece dados sobre repositórios de código hospedados nesta plataforma (GitHub, 2022).

Categoria	Descrição
Compreender o fluxo de contribuição (CF)	Não é incomum que os recém-chegados se sintam perdidos ou desmotivados quando não está claro como contribuir com um projeto de software livre. Para esta categoria, identificamos trechos de documentação que descrevem qual é o fluxo de contribuição de um projeto. O fluxo de contribuição pode ser definido como um conjunto de etapas que um recém-chegado precisa seguir para desenvolver uma contribuição aceitável para o projeto.
Escolher uma tarefa (ET)	Muitos desenvolvedores estão interessados em contribuir com projetos de software livre, mas a maioria deles não sabe com qual tarefa começar. Nesta categoria, identificamos frases descrevendo como os recém-chegados podem encontrar uma tarefa para contribuir com o projeto.
Contactar a comunidade (CC)	Além de um mentor, também é importante que os recém-chegados entrem em contato com a própria comunidade do projeto. Por esse motivo, identificamos nesta categoria qualquer informação que detalhasse como um recém-chegado pode entrar em contato com os membros da comunidade, incluindo links para canais de comunicação, tutoriais sobre como enviar uma mensagem, etiquetas de comunicação, entre outros.
Construir o ambiente de trabalho (CA)	Recém-chegados relataram em estudos anteriores que não encontraram explicações sobre como poderiam construir seu próprio ambiente de trabalho (construir, compilar, executar, gerenciar dependências, etc.) antes de contribuir. Para esta categoria, identificamos frases que explicassem como um novato poderia construir seu ambiente local de trabalho.
Lidar com código (LC)	Muitos projetos têm seus próprios padrões de código, arquiteturas e práticas de software. Nesta categoria, identificamos frases na documentação dos projetos descrevendo como o código deveria ser escrito, organizado e documentado pelos novos contribuidores.
Submeter mudanças (SM)	O último passo no processo de contribuição é a submissão de mudanças ao repositório do projeto. Nesta categoria, identificamos informações sobre como a submissão de alteração (em inglês, patch) deveria ser feita pelos novatos.

Tabela 3.1: *Categorias estabelecidas para identificação nas documentações dos projetos.*

Extração dos arquivos de contribuição

De cada um dos projetos selecionados, foi extraído o arquivo CONTRIBUTING do repositório de código fonte. A razão para escolha deste arquivo e não de outras fontes de documentação, se concentra no fato deste apresentar informações relevantes a novos contribuidores, ser de fácil extração e ser usualmente escrito a partir de uma mesma sintaxe, o que facilitaria o processo de identificação das categorias. Visto que os projetos selecionados estavam hospedados na plataforma de codificação GitHub, a documentação dos projetos também foram extraídas por meio da GitHub API. Para garantir que os projetos a serem estudados continham um arquivo CONTRIBUTING válido para análise, foram definidos um conjunto de filtros de seleção. Projetos que não continham o arquivo CONTRIBUTING, não continham o arquivo escrito em inglês, não continham o arquivo em formato Markdown, ou que continham o arquivo com tamanho menor que 0.5kB, foram removidos da amostra.

Ao fim da filtragem, o arquivo CONTRIBUTING de 2.915 projetos de software livre foram escolhidos para análise qualitativa. As linguagens de programação com maior número de projetos válidos foram TypeScript (n = 469), JavaScript (n = 399) e Ruby (n = 328). O total de 6599 projetos foram removidos após a filtragem, com uma média de 660 repositórios removidos por linguagem, sendo Java (n = 221), C (n = 172) e Shell (n = 172) as linguagens com menor número de projetos inclusos na amostra final, e a ausência do arquivo CONTRIBUTING o principal motivo de exclusão (n = 6055). Os motivos pelos quais certos projetos foram removidos são apresentados na Tabela 3.2.

Removido porque o arquivo CONTRIBUTING:	**JavaScript** (n=824)	**Python** (n=929)	**Java** (n=942)	**PHP** (n=941)	**C#** (n=990)	**C++** (n=942)	**TypeScript** (n=990)	**Shell** (n=990)	**C** (n=947)	**Ruby** (n=1019)
Estava faltando	381 (46%)	527 (57%)	692 (73%)	593 (63%)	651 (66%)	604 (64%)	474 (48%)	785 (79%)	702 (74%)	646 (63%)
Não estava em inglês	3 (> 1%)	12 (1%)	4 (> 1%)	4 (> 1%)	3 (> 1%)	2 (> 1%)	3 (> 1%)	4 (> 1%)	2 (> 1%)	1 (> 1%)
Não estava em Markdown	2 (> 1%)	91 (10%)	5 (1%)	4 (> 1%)	1 (> 1%)	14 (1%)	4 (> 1%)	6 (1%)	26 (3%)	11 (1%)
Tinha tamanho menor que 0.5kB	41 (5%)	37 (4%)	22 (2%)	49 (5%)	54 (5%)	35 (4%)	42 (4%)	23 (2%)	28 (3%)	33 (3%)
Total de projetos removidos	**425 (52%)**	**661 (71%)**	**721 (76%)**	**647 (68%)**	**709 (71%)**	**651 (69%)**	**521 (52%)**	**818 (82%)**	**755 (79%)**	**691 (67%)**

Tabela 3.2: *Número de projetos removidos por linguagem e as razões para exclusão. O valor "n"representa o total de projetos extraídos para determinada linguagem. É válido mencionar que certos arquivos podem ter sido removidos por mais de uma razão.*

Conversão dos arquivos em planilhas para análise

Com apenas arquivos CONTRIBUTING válidos na amostra, os arquivos Markdown extraídos dos repositórios de código foram convertidos para planilhas em Excel. As planilhas, cada qual representando um projeto, foram organizadas em sete colunas. Na primeira coluna, os parágrafos do arquivo CONTRIBUTING foram divididos entre as cédulas da coluna. A divisão dos parágrafos foi implementada com base na definição fornecida pelo próprio GitHub (GitHub, s.d.), que define parágrafo como "*uma sequência de linhas não brancas que não podem ser interpretadas como outras estruturas de bloco*". As colunas seguintes foram reservadas para identificação das categorias de informação listadas na Tabela 2.3, a serem marcadas durante a análise qualitativa.

3.1.3 Análise dos dados

Com os arquivos em formato de planilha, foi realizada a análise qualitativa dos parágrafos para os projetos considerados válidos. Dois pesquisadores ficaram responsáveis por analisar os dados, um sendo aluno de mestrado em Ciência da Computação, e outro doutor em Ciência da Computação. Para assegurar que ambos obtivessem um padrão similar na identificação das categorias de informação, um conjunto de trinta projetos foram escolhidos aleatoriamente para análise e discussão. A análise dos trinta projetos foi dividida em três etapas consecutivas, e as planilhas analisadas da seguinte forma: Com a planilha de um projeto aberta, o pesquisador lia os parágrafos na primeira coluna e, quando julgasse tal informação pertencente a uma categoria de informação, adicionava uma marcação a coluna da respectiva categoria na mesma linha do parágrafo. Em nosso método, ficou decidido que cada parágrafo poderia receber apenas uma única categoria, já que a quantidade de informação oferecida por parágrafo, de modo geral, seria suficiente apenas para uma e não múltiplas categorias.

Ao final de cada etapa, os pesquisadores discutiam as planilhas analisadas e definiam acordos a serem seguidos durante a análise qualitativa. Ao fim da primeira etapa, os pesquisadores atingiram uma concordância de 47.81% a partir de 10 planilhas analisadas. Para as duas últimas etapas, as concordâncias chegaram a 74.78%, o que foi considerado pelos pesquisadores como um valor suficiente de concordância, dado a complexidade do problema em questão. Ao final da terceira e última etapa, ambos os pesquisadores passaram então a analisar um novo conjunto de 500 planilhas retiradas da amostra de projetos válidos. Nenhum destes projetos havia sido utilizado na análise preliminar. A anotação ocorreu da mesma forma, com no máximo uma categoria anotada por parágrafo. Ao fim da análise, os pesquisadores chegaram a um total de 20.733 parágrafos analisados.

3.1.4 Pré-processamento

Preparação dos parágrafos

A preparação dos dados para classificação, definida pelas etapas de pré-processamento e extração de características na Figura 3.2, consistiu em preparar e dividir o conteúdo das planilhas analisadas entre características e rótulos para implementação do classificador. Neste estudo, consideramos como características os parágrafos das planilhas analisadas, e rótulos a categoria identificada pelos pesquisadores para cada um dos parágrafos. Antes de transformarmos os parágrafos em características para classificação, uma etapa de pré-processamento destes dados foi executada. Nesta etapa, informações não relevantes à classificação foram removidas a partir da combinação de três métodos comuns ao processamento de linguagem natural: lemmatização (Thanaki, 2017), remoção de pontuações (Lantz, 2013), e remoção de palavras irrelevantes (Em inglês, *stopwords*).

No processo de lemmatização, variações de uma mesma palavra foram agrupadas em uma única forma a fim de facilitar o entendimento do conteúdo dos parágrafos por parte do classificador (e.g., em inglês, as palavras "*studies*" e "*studying*" foram reduzidas à sua forma raiz "*study*"). A remoção de pontuações foi feita através de expressões regulares e métodos de substituição de palavras, e a remoção de palavras irrelevantes feita a partir de um conjunto de palavras pré-definido, removidas por meio de código. Neste estudo, o

processo de lemmatização e removação de palavras irrelevantes foi feito através da biblioteca NLTK[1], amplamente utilizada em estudos envolvendo processamento de linguagem natural (Bonaccorso, 2017b; Loper e Bird, 2002; Bird, 2006).

Conversão dos parágrafos em características

Para melhor descrever o conteúdo dos projetos analisados, os parágrafos foram convertidos em características estatísticas e heurísticas. Foram considerados como valores estatísticos, a transformação dos parágrafos em características TF-IDF[2] obtidas com o uso de bibliotecas de aprendizado de máquina. Os valores heurísticos foram obtidos por meio da análise manual dos parágrafos e categorias marcadas. Quando um padrão linguístico que associava parágrafos a uma categoria era identificado pelos pesquisadores, um novo valor heurístico era então gerado como característica para classificação.

A ideia de valores heurísticos emergiu do estudo realizado por Prana *et al.* (2019) que identificou, por exemplo, que a categoria "cabeçalho da documentação" proposta em seu estudo, poderia ser identificada pela ocorrência do título do repositório de código na documentação analisada. Além da divisão entre rótulos e características para classificação, os valores convertidos também foram divididos em conjuntos de treino e teste, a serem utilizados nas etapas seguintes.

Seleção de características

Visando aprimorar o processo de classificação dos parágrafos, uma etapa de seleção de características foi executada. Nesta etapa, a função SelectPercentile[3] foi utilizada para selecionar apenas as melhores características para treinamento. Tal função atribui pontuações para o conjunto de características disponível para treinamento através de testes estatísticos, e é capaz de selecionar apenas as melhores características a partir de um percentil definido como entrada pelo usuário. Nesta pesquisa, 15% das características geradas na etapa anterior foram mantidas, sendo estas as com melhores pontuações obtidas através da função SelectPercentile. O teste estatístico escolhido para esta seleção foi o Qui-quadrado, comum a problemas de classificação (Bonaccorso, 2017d; Brownlee, 2019).

3.1.5 Classificação

Para que fosse possível automatizar o processo de identificação das categorias em documentações de projetos de software livre, um modelo de classificação foi implementado. A elaboração deste modelo aconteceu em três fases: Primeiro, cinco algoritmos de aprendizagem supervisionada foram utilizados para treinar diferentes modelos de classificação utilizando um subconjunto de características disponíveis para treinamento. Em seguida, um processo de avaliação da performance destes classificadores foi executado, de modo a encontrar o modelo que melhor identificasse as categorias em documentações de projetos

[1] https://www.nltk.org/

[2] TF-IDF (do inglês, *Term Frequency - Inverse Document Frequency*) é uma técnica estatística de extração de características textuais, que tem como principal objetivo identificar palavras relevantes em um texto não ou semi estruturado (Kido *et al.*, 2014)

[3] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectPercentile.html

de software livre. Tal modelo foi então treinado novamente com todas as características disponíveis para classificação, e avaliado através de uma análise quantitativa e um questionário com programadores. Uma ilustração do processo de desenvolvimento do modelo de classificação, bem como os passos da etapa de pré-processamento também utilizadas por este modelo, é apresentada na Figura 3.2.

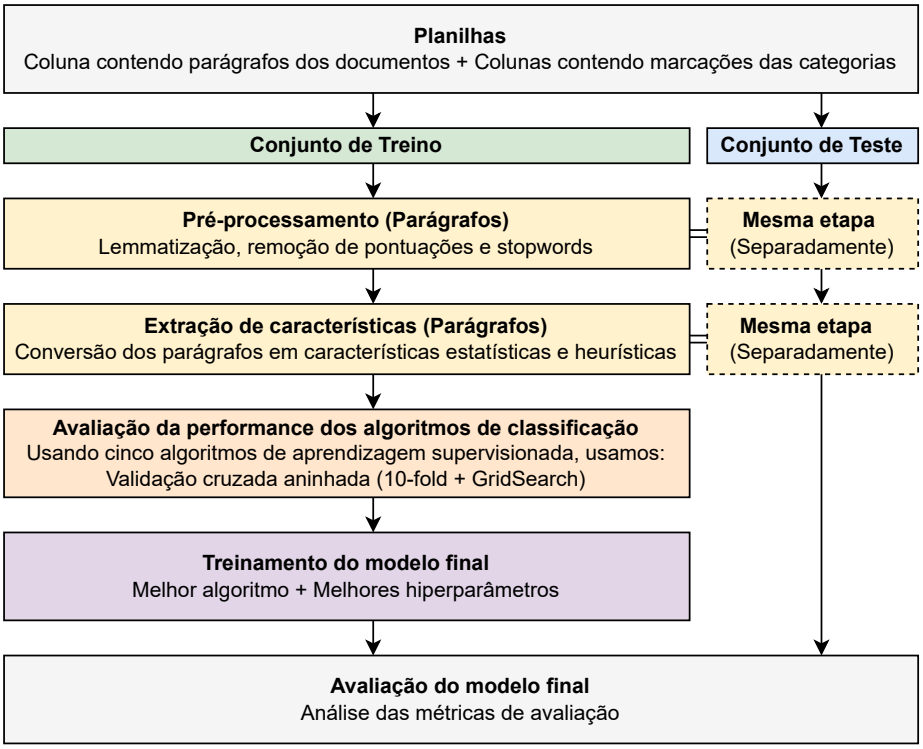


Figura 3.2: *Construção do modelo de classificação.*

Algoritmos de classificação

Cinco algoritmos de aprendizagem supervisionada foram treinados a fim de se obter um modelo de classificação que identificasse trechos relevantes a novatos em documentações de projetos em software livre. Como foi estabelecido que seis categorias de informação poderiam ser identificadas nas documentações dos projetos, algoritmos com suporte a classificação multi-classe foram escolhidos para treinamento, e duas estratégias de classificação foram utilizadas, One-vs-Rest (OvR) e One-vs-One (OvO) (Bonaccorso, 2017c).

Na estratégia One-vs-Rest (também conhecida como One-vs-All), classificadores binários foram treinados para identificar cada classe de informação. Cada classificador ficou responsável por distinguir uma classe das demais existentes, e uma função de decisão ficou responsável por atribuir uma classe a um novo parágrafo de entrada com base nos resultados dos classificadores binários. Na estratégia One-vs-One, classificadores binários foram utilizados para treinar as seis classes de informação em pares, e uma função de decisão também foi utilizada na categorização de uma entrada desconhecida.

Entre os algoritmos utilizados para treinamento, se encontravam: Random Forest (RF), Linear Support Vector Machine (SVM), Multinomial Naive Bayes (NB), K-Neighbors (kNN),

Logistic Regression (LR). A escolha dos algoritmos de classificação foi feita com base no fato destes serem amplamente utilizados em aprendizagem de máquina (Bonaccorso, 2017a), e por também terem sido utilizados em trabalhos relacionados, em especial, nos estudos desenvolvidos por Prana *et al.* (2019) e Panichella *et al.* (2015).

Para comparar a performance dos algoritmos selecionados com estratégias mais simples de predição, dois algoritmos de classificação aleatórios também foram utilizados nesta etapa. Um deles retornando sempre a classe mais frequente para todas as instâncias no conjunto de treinamento, e o outro atribuindo classes de maneira aleatória.

Escolha do melhor modelo de classificação

Na etapa de escolha do melhor algoritmo, diferentes modelos de classificação foram construídos a partir dos cinco algoritmos de classificação escolhidos anteriormente. O objetivo desta etapa era encontrar o modelo que melhor identificasse as categorias de documentação desejadas. Para melhor avaliar a capacidade de predição de cada um dos algoritmos, duas estratégias foram combinadas, a técnica de validação cruzada k-fold e a métrica de avaliação *f-measure*. A técnica de validação cruzada k-fold foi utilizada para gerar diferentes cenários de classificação a partir dos dados obtidos na etapa de geração das características. Os algoritmos de classificação selecionados na etapa de treinamento foram treinados separadamente no processo de validação cruzada.

As métricas de avaliação *f-measure*, *precision* e *recall* foram utilizadas em conjunto com a validação cruzada, sendo as responsáveis por definir qual dos algoritmos de classificação selecionados obtiveram a melhor performance nos diferentes cenários gerados pela validação. Para um mesmo algoritmo de classificação, diferentes configurações de parâmetros também foram testadas. Ao final, o algoritmo de classificação que apresentou o melhor valor de *f-measure*, foi responsável por gerar o modelo de classificação final deste estudo, submetido a etapa de avaliação (Seção 3.1.6) como parte resultante desta pesquisa.

As duas atividades a serem executadas na etapa de validação são bem simples. Primeiro, a estratégia de validação cruzada k-fold será utilizada para dividir os dados das planilhas, então convertidos em características e rótulos, em subconjuntos de dados. Nesta estratégia, os dados são particionados em k conjuntos de dados de tamanho similar. O primeiro conjunto de dados é utilizado para teste, e os demais $k - 1$ são utilizados para treinar modelos a partir de um mesmo algoritmo de classificação. Esta primeira etapa é executada para avaliar a capacidade de generalização[4] dos algoritmos de classificação selecionados na etapa de treinamento em relação ao problema definido nesta pesquisa. Após o processo de divisão entre k conjuntos de dados, os modelos treinados com $k - 1$ subconjuntos de treino serão avaliados pela métrica de avaliação *f-measure* utilizando o conjunto de teste remanescente. A média aritmética do valor de *f-measure* é obtida para os $k - 1$ modelos, e o algoritmo de classificação que obtiver o maior valor entre as médias será aquele a ser selecionado para gerar o modelo de classificação final.

Descrição da métrica de avaliação

A métrica *f-measure* para problemas de classificação multi-rótulo é definida da seguinte

[4]Generalização refere-se à capacidade de realizar predições corretas em dados novos, anteriormente não vistos, em oposição aos dados usados para treinar o modelo de classificação (Google Glossary, 2019)

forma (Panichella *et al.*, 2015):

$$F1 = \frac{\sum_{l \in L} w_l \times F1_l}{|L|}$$
$$F1_l = \frac{2 \times Precision_l \times Recall_l}{Precision_l + Recall_l} \tag{3.1}$$

onde w_l é a proporção do rótulo l em todos os dados previstos, $F1_l$ é o valor de *f-measure* para o rótulo l, L é o conjunto de rótulos, $Precision_l$ é a precisão para o rótulo l, e $Recall_l$ é o sensibilidade para o rótulo l. Ao computar precisão e sensibilidade para um determinado rótulo l, uma instância é considerada positiva se ela contém o rótulo l, caso contrário, é considerada negativa. Desta forma, temos que precisão é a proporção de instâncias positivas preditas que são realmente positivas, enquanto que sensibilidade é a proporção de instâncias positivas que foram preditas como positivas. Vale ressaltar que o valor de *f-measure* é calculado para os $k - 1$ modelos gerados na etapa validação cruzada separadamente, e uma média aritmética destes valores é apresentada como resultado para o algoritmo de classificação a ser testado neste processo.

3.1.6 Avaliação do modelo final

Assim que o modelo de classificação final for obtido após a validação, duas análises serão entregues como parte resultante deste trabalho. Entre as atividades, serão entregues uma análise da importância das características do modelo de classificação para a predição das categorias, e uma avaliação das predições feitas pelo modelo, a serem avaliadas por novos e experientes contribuidores vindos de projetos de software livre. A descrição de cada uma das atividades é definida abaixo.

Avaliação das características

Assim que um modelo de classificação for selecionado após a etapa de validação, uma investigação das características deste modelo será executada. A ideia é compreender quais características são importantes para identificação de cada uma das categorias relevantes a novatos em software livre. Tanto as características estatísticas como as heurísticas serão analisadas. Para que essa análise seja possível, os pesos de cada uma das características serão extraídos do modelo de classificação elegido, através da própria biblioteca de aprendizado de máquina a ser utilizada durante a construção do modelo. Quanto maior o peso de uma característica, consequentemente maior será sua importância para identificação de uma ou mais categorias. A relevância das principais características será apresentada por meio de gráficos de dispersão.

Além disso, uma avaliação da diferença de importância entre características estatísticas e heurísticas também será efetuada. Para isso, dois novos modelos de classificação similares a aquele selecionado na validação serão criados, um contendo apenas o conjunto de valores estatísticos, e outro apenas o conjunto de valores heurísticos. O valor de *f-measure* será calculado para ambos os modelos. A ideia é compreender qual conjunto de valores tem maior impacto na identificação das categorias relevantes a novos contribuidores. O conjunto

de características do modelo que obtiver o melhor valor *f-measure* será definido como aquele com o maior relevância para identificação das categorias.

Avaliação das predições

A última etapa desta pesquisa consistirá em avaliar se as predições feitas pelo modelo de classificação selecionado durante a validação fazem sentido para contribuidores no contexto de software livre. Para isso, novos e experientes contribuidores serão convidados, através de um questionário, a avaliarem trechos de documentação identificados pelo modelo de classificação. Documentações de um novo conjunto de projetos serão utilizadas durante esta etapa. Os projetos a serem selecionados serão extraídos da mesma lista proposta por Avelino *et al.* (2015), só que desta vez, com *fator ônibus* maior que três, e o modelo de classificação será aplicado para identificar as mesmas categorias nas novas documentações a serem coletadas.

Índice	Pergunta
1	Qual é sua profissão atualmente?
2	Você considera o desenvolvimento de projetos de software livre parte de sua rotina profissional?
3	Para quantos projetos de software livre você já contribuiu?
4	Se você já realizou alguma contribuição para projetos de software livre, responda: Há quantos anos você tem contribuído com projetos neste contexto?
5	Você já desistiu alguma vez de contribuir com um projeto de software livre? Se sim, quais foram as razões que te levaram a tal desistência?
6	Como você avalia o processo de entrada em um projeto de software livre na perspectiva de alguém que almeja contribuir pela primeira vez?
7	Que tipo de informação você consideraria relevante para um novato ao ingressar em um projeto de software livre?
8	Você concorda que as categorias definidas a seguir são relevantes para novos contribuidores durante o processo de entrada em software livre?
9	Considerando as categorias definidas neste estudo, responda: Você concorda que os trechos de documentação a seguir condizem com as categorias identificadas?
10	Você acredita que a identificação de trechos relevantes a novos contribuidores podem contribuir com a entrada de novatos em projetos de software livre? Justifique sua resposta.

Tabela 3.3: *Perguntas a serem aplicadas no questionário*

A Tabela 3.3 apresenta as questões a serem respondidas pelos participantes durante a execução do questionário. A seleção de novos contribuidores deverá acontecer da mesma forma que o estudo empírico proposto na seção anterior, solicitando através de instituições de ensino superior, que alunos das disciplinas de desenvolvimento de software livre respondam as perguntas definidas. Quanto a contribuidores com maior experiência em software livre, serão convidados aqueles que mantêm projetos inclusos na lista de analisados pelo modelo de classificação, com *fator ônibus* maior que três. O convite dos novatos acontecerá por intermédio das próprias instituições de ensino e os de contribuidores experientes, por meio das listas de e-mail dos projetos que estes mantêm.

Além de indagar os participantes sobre a qualidade das identificações feitas pelo modelo de classificação, algumas questões tentarão também levantar informações relacionadas a profissão e histórico de contribuição em software livre. Ao final desta etapa, espera-se construir um apanhado geral das respostas dadas pelos participantes, de modo a compreender se, de fato, o modelo de classificação foi capaz de identificar as categorias corretamente, levando em consideração a perspectiva de novos e experientes contribuidores em software livre.

Capítulo 4

Resultados

Capítulo 5

Plano de Trabalho

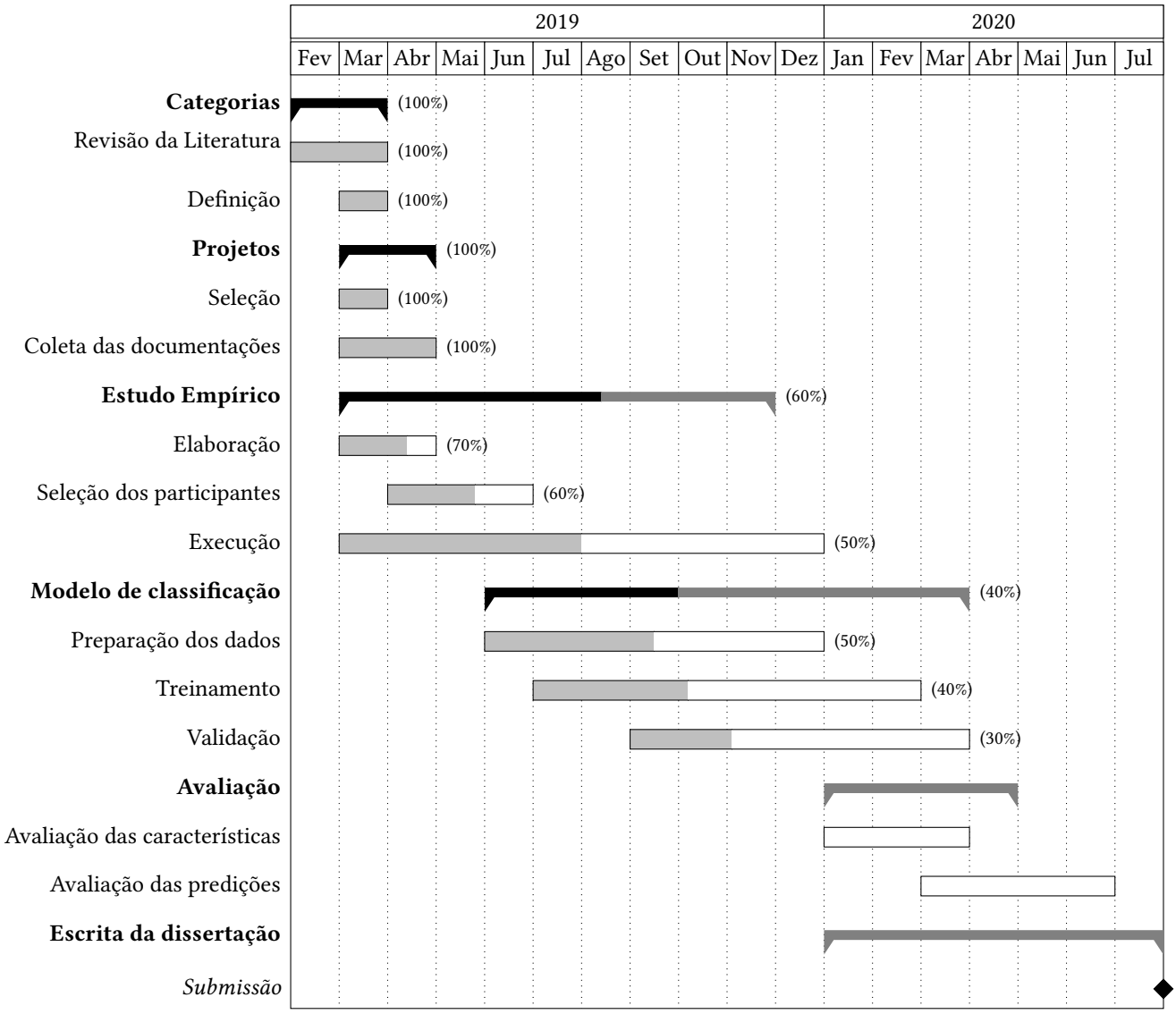


Figura 5.1: *Cronograma proposto.*

Referências

[Aberdour 2007] M. Aberdour. "Achieving quality in open-source software". Em: *IEEE Software* 24.1 (jan. de 2007), pgs. 58–64. doi: 10.1109/MS.2007.2 (citado na pg. 11).

[Aghajani *et al.* 2019] Emad Aghajani *et al.* "Software documentation issues unveiled". Em: *Proceedings of the 41st International Conference on Software Engineering*. ICSE '19. Montreal, Quebec, Canada: IEEE Press, 2019, pgs. 1199–1210. doi: 10.1109/ICSE.2019.00122. url: https://doi.org/10.1109/ICSE.2019.00122 (citado nas pgs. 1, 10).

[Avelino *et al.* 2015] Guilherme Avelino, Marco Tulio Valente e Andre Hora. *What is the Truck Factor of popular GitHub applications? A first assessment*. Rel. técn. PeerJ PrePrints, 2015 (citado na pg. 24).

[Borges *et al.* 2016] Hudson Borges, Andre Hora e Marco Tulio Valente. "Understanding the factors that impact the popularity of github repositories". Em: *2016 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE. 2016, pgs. 334–344 (citado na pg. 16).

[Bird 2006] Steven Bird. "Nltk: the natural language toolkit". Em: *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*. 2006, pgs. 69–72 (citado na pg. 20).

[Bonaccorso 2017a] Giuseppe Bonaccorso. Packt Publishing, 2017. isbn: 978-1-78588-962-2. url: https://app.knovel.com/hotlink/toc/id:kpMLA00001/machine-learning-algorithms/machine-learning-algorithms (citado na pg. 22).

[Bonaccorso 2017b] Giuseppe Bonaccorso. *12.2.2 Stopword Removal*. Packt Publishing, 2017. isbn: 978-1-78588-962-2. url: https://app.knovel.com/hotlink/khtml/id:kt011DN9A6/machine-learning-algorithms/stopword-removal (citado na pg. 20).

[Bonaccorso 2017c] Giuseppe Bonaccorso. *2.1.1.1 one-vs-all*. 2017. url: https://app.knovel.com/hotlink/khtml/id:kt011DN5UB/machine-learning-algorithms/one-vs-all (citado na pg. 21).

[Bonaccorso 2017d] Giuseppe Bonaccorso. *3.6 feature selection and filtering*. 2017. URL: https://app.knovel.com/hotlink/khtml/id:kt011DN6C1/machine-learning-algorithms/feature-selection-filtering (citado na pg. 20).

[Bonaccorsi e Rossi 2003] Andrea Bonaccorsi e Cristina Rossi. "Why open source software can succeed". Em: *Research policy* 32.7 (2003), pgs. 1243–1258 (citado na pg. 1).

[Briand 2003] L. C. Briand. "Software documentation: how much is enough?" Em: *Seventh European Conference onSoftware Maintenance and Reengineering, 2003. Proceedings.* Mar. de 2003, pgs. 13–15. DOI: 10.1109/CSMR.2003.1192406 (citado nas pgs. 1, 10).

[Brownlee 2019] Jason Brownlee. "How to choose a feature selection method for machine learning". Em: *Machine Learning Mastery* 10 (2019) (citado na pg. 20).

[Canfora *et al.* 2012] Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto e Sebastiano Panichella. "Who is going to mentor newcomers in open source projects?" Em: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM. 2012, pg. 44 (citado na pg. 11).

[Crowston *et al.* 2012] Kevin Crowston, Kangning Wei, James Howison e Andrea Wiggins. "Free/libre open-source software development: what we know and what we do not know". Em: *ACM Computing Surveys (CSUR)* 44.2 (2012), pg. 7 (citado na pg. 1).

[Cubranic *et al.* 2005] D. Cubranic, G. C. Murphy, J. Singer e K. S. Booth. "Hipikat: a project memory for software development". Em: *IEEE Transactions on Software Engineering* 31.6 (jun. de 2005), pgs. 446–465. DOI: 10.1109/TSE.2005.71 (citado na pg. 11).

[Dahlander e Magnusson 2008] Linus Dahlander e Mats Magnusson. "How do firms make use of open source communities?" Em: *Long Range Planning* 41.6 (2008), pgs. 629–649. ISSN: 0024-6301. DOI: https://doi.org/10.1016/j.lrp.2008.09.003. URL: http://www.sciencedirect.com/science/article/pii/S0024630108000836 (citado na pg. 8).

[Forward e Lethbridge 2002] Andrew Forward e Timothy C. Lethbridge. "The relevance of software documentation, tools and technologies: a survey". Em: *Proceedings of the 2002 ACM Symposium on Document Engineering*. DocEng '02. McLean, Virginia, USA: ACM, 2002, pgs. 26–33. ISBN: 1-58113-594-7. DOI: 10.1145/585058.585065. URL: http://doi.acm.org/10.1145/585058.585065 (citado na pg. 10).

[Fronchetti *et al.* 2019] Felipe Fronchetti, Igor Wiese, Gustavo Pinto e Igor Steinmacher. "What attracts newcomers to onboard on oss projects? tl; dr: popularity". Em: *IFIP International Conference on Open Source Systems*. Springer. 2019, pgs. 91–103 (citado nas pgs. 2, 15).

[Fuggetta 2003] Alfonso Fuggetta. “Open source software––an evaluation”. Em: *Journal of Systems and Software* 66.1 (2003), pgs. 77–90. ISSN: 0164-1212. DOI: https://doi.org/10.1016/S0164-1212(02)00065-1. URL: http://www.sciencedirect.com/science/article/pii/S0164121202000651 (citado na pg. 5).

[C. Gacek e B. Arief 2004] C. Gacek e B. Arief. “The many meanings of open source”. Em: *IEEE Software* 21.1 (jan. de 2004), pgs. 34–40. DOI: 10.1109/MS.2004.1259206 (citado na pg. 11).

[Cristina Gacek e Budi Arief 2004] Cristina Gacek e Budi Arief. “The many meanings of open source”. Em: *IEEE Softw.* 21.1 (jan. de 2004), pgs. 34–40. ISSN: 0740-7459. DOI: 10.1109/MS.2004.1259206. URL: http://dx.doi.org/10.1109/MS.2004.1259206 (citado na pg. 5).

[GitHub s.d.] GitHub. *GitHub Flavored Markdown Specs: Paragraphs*. [Accessed on Aug-29-2022]. URL: https://github.github.com/gfm/#paragraphs (citado na pg. 18).

[GitHub 2020] GitHub. *GitHub Octoverse*. [Accessed on Jul-1-2020]. 2020. URL: https://octoverse.github.com/ (citado na pg. 16).

[GitHub 2022] GitHub. *GitHub API*. [Accessed on Out-03-2022]. 2022. URL: https://docs.github.com/en/rest (citado na pg. 16).

[Google Glossary 2019] Google Glossary. *Definition of Generalization*. https://developers.google.com/machine-learning/glossary#generalization, Last accessed on 20-08-2019. 2019 (citado na pg. 22).

[Haiduc *et al.* 2010] S. Haiduc, J. Aponte, L. Moreno e A. Marcus. “On the use of automated text summarization techniques for summarizing source code”. Em: *2010 17th Working Conference on Reverse Engineering*. Out. de 2010, pgs. 35–44. DOI: 10.1109/WCRE.2010.13 (citado nas pgs. 1, 13).

[Hannebauer e Gruhn 2017] Christoph Hannebauer e Volker Gruhn. “On the relationship between newcomer motivations and contribution barriers in open source projects”. Em: *Proceedings of the 13th International Symposium on Open Collaboration*. OpenSym ’17. Galway, Ireland: ACM, 2017, 2:1–2:10. ISBN: 978-1-4503-5187-4. DOI: 10.1145/3125433.3125446. URL: http://doi.acm.org/10.1145/3125433.3125446 (citado nas pgs. 8, 16).

[Höst e Oručević-Alagić 2011] Martin Höst e Alma Oručević-Alagić. “A systematic review of research on open source software in commercial software product development”. Em: *Information and Software Technology* 53.6 (2011), pgs. 616–624 (citado na pg. 6).

[Kajko-Mattsson 2005] Mira Kajko-Mattsson. “A survey of documentation practice within corrective maintenance”. Em: *Empirical Software Engineering* 10.1 (jan. de 2005), pgs. 31–55. ISSN: 1573-7616. DOI: 10.1023/B:LIDA.0000048322.42751.ca. URL: https://doi.org/10.1023/B:LIDA.0000048322.42751.ca (citado na pg. 10).

[Kido *et al.* 2014] Guilherme Sakaji Kido, Sylvio Barbon Junior e Stella Naomi Moriguchi. *Comparação entre TF-IDF e LSI para pesagem de termos em micro-blog*. 2014 (citado na pg. 20).

[Kraut *et al.* 2012] Robert E Kraut, Moira Burke, John Riedl e Paul Resnick. *The challenges of dealing with newcomers*. 2012 (citado na pg. 8).

[Krogh *et al.* 2003] Georg von Krogh, Sebastian Spaeth e Karim R Lakhani. "Community, joining, and specialization in open source software innovation: a case study". Em: *Research Policy* 32.7 (2003). Open Source Software Development, pgs. 1217–1241. ISSN: 0048-7333. DOI: https://doi.org/10.1016/S0048-7333(03)00050-7. URL: http://www.sciencedirect.com/science/article/pii/S0048733303000507 (citado nas pgs. 7, 8, 16).

[Lantz 2013] Brett Lantz. *4.2.3 Data Preparation - Processing Text Data for Analysis*. Packt Publishing, 2013. ISBN: 978-1-78216-214-8. URL: https://app.knovel.com/hotlink/khtml/id:kt00U5RBR6/machine-learning-with/data-preparation-processing (citado na pg. 19).

[Loper e Bird 2002] Edward Loper e Steven Bird. "Nltk: the natural language toolkit". Em: *arXiv preprint cs/0205028* (2002) (citado na pg. 20).

[Li *et al.* 2016] B. Li, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk e N. A. Kraft. "Automatically documenting unit test cases". Em: *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. Abr. de 2016, pgs. 341–352. DOI: 10.1109/ICST.2016.30 (citado na pg. 13).

[Malheiros *et al.* 2012] Y. Malheiros, A. Moraes, C. Trindade e S. Meira. "A source code recommender system to support newcomers". Em: *2012 IEEE 36th Annual Computer Software and Applications Conference*. 2012, pgs. 19–24. DOI: 10.1109/COMPSAC.2012.11 (citado na pg. 11).

[Audris Mockus *et al.* 2000] Audris Mockus, Roy T. Fielding e James Herbsleb. "A case study of open source software development: the apache server". Em: *Proceedings of the 22Nd International Conference on Software Engineering*. ICSE '00. Limerick, Ireland: ACM, 2000, pgs. 263–272. ISBN: 1-58113-206-9. DOI: 10.1145/337180.337209. URL: http://doi.acm.org/10.1145/337180.337209 (citado na pg. 6).

[Nakakoji *et al.* 2002] Kumiyo Nakakoji, Yasuhiro Yamamoto, Yoshiyuki Nishinaka, Kouichi Kishida e Yunwen Ye. "Evolution patterns of open-source software systems and communities". Em: *Proceedings of the International Workshop on Principles of Software Evolution*. IWPSE '02. Orlando, Florida: ACM, 2002, pgs. 76–85. ISBN: 1-58113-545-9. DOI: 10.1145/512035.512055. URL: http://doi.acm.org/10.1145/512035.512055 (citado na pg. 7).

[Panichella *et al.* 2015] S. Panichella *et al.* "How can i improve my app? classifying user reviews for software maintenance and evolution". Em: *2015 IEEE International*

Conference on Software Maintenance and Evolution (ICSME). Set. de 2015, pgs. 281–290. DOI: 10.1109/ICSM.2015.7332474 (citado nas pgs. 15, 22, 23).

[Panichella 2015] S. Panichella. "Supporting newcomers in software development projects". Em: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Set. de 2015, pgs. 586–589. DOI: 10.1109/ICSM.2015.7332519 (citado na pg. 13).

[Pinto, Dias *et al.* 2018] Gustavo Pinto, Luiz Felipe Dias e Igor Steinmacher. "Who gets a patch accepted first?: comparing the contributions of employees and volunteers". Em: *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM. 2018, pgs. 110–113 (citado na pg. 6).

[Pinto, Igor Steinmacher *et al.* 2018] Gustavo Pinto, Igor Steinmacher, Luiz Felipe Dias e Marco Gerosa. "On the challenges of open-sourcing proprietary software projects". Em: *Empirical Software Engineering* 23.6 (dez. de 2018), pgs. 3221–3247. ISSN: 1573-7616. DOI: 10.1007/s10664-018-9609-6. URL: https://doi.org/10.1007/s10664-018-9609-6 (citado na pg. 6).

[Prana *et al.* 2019] Gede Artha Azriadi Prana, Christoph Treude, Ferdian Thung, Thushari Atapattu e David Lo. "Categorizing the content of github readme files". Em: *Empirical Software Engineering* 24.3 (jun. de 2019), pgs. 1296–1327. ISSN: 1573-7616. DOI: 10.1007/s10664-018-9660-3. URL: https://doi.org/10.1007/s10664-018-9660-3 (citado nas pgs. 20, 22).

[Qureshi e Fang 2011] Israr Qureshi e Yulin Fang. "Socialization in open source software projects: a growth mixture modeling approach". Em: *Organizational Research Methods* 14.1 (2011), pgs. 208–238. DOI: 10.1177/1094428110375002. eprint: https://doi.org/10.1177/1094428110375002. URL: https://doi.org/10.1177/1094428110375002 (citado na pg. 8).

[Raymond 1999] Eric Raymond. "The cathedral and the bazaar". Em: *Knowledge, Technology & Policy* 12.3 (1999), pgs. 23–49. ISSN: 1874-6314. DOI: 10.1007/s12130-999-1026-0. URL: https://doi.org/10.1007/s12130-999-1026-0 (citado na pg. 6).

[Robillard 2009] M. P. Robillard. "What makes apis hard to learn? answers from developers". Em: *IEEE Software* 26.6 (nov. de 2009), pgs. 27–34. DOI: 10.1109/MS.2009.193 (citado na pg. 10).

[Souza *et al.* 2005] Sergio Cozzetti B. de Souza, Nicolas Anquetil e Káthia M. de Oliveira. "A study of the documentation essential to software maintenance". Em: *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting &Amp; Designing for Pervasive Information*. SIGDOC '05. Coventry, United Kingdom: ACM, 2005, pgs. 68–75. ISBN: 1-59593-175-9. DOI: 10.1145/1085313.1085331. URL: http://doi.acm.org/10.1145/1085313.1085331 (citado na pg. 10).

[Stallman 2002] Richard Stallman. "What is free software". Em: *Free Society: Selected Essays of* 23 (2002). URL: https://www.gnu.org/philosophy/free-sw.html (citado na pg. 5).

[I. Steinmacher, Chaves *et al.* 2014] I. Steinmacher, A. P. Chaves, T. U. Conte e M. A. Gerosa. "Preliminary empirical identification of barriers faced by newcomers to open source software projects". Em: *2014 Brazilian Symposium on Software Engineering*. 2014, pgs. 51–60. DOI: 10.1109/SBES.2014.9 (citado nas pgs. 8–12, 15).

[I. Steinmacher, Wiese *et al.* 2014] I. Steinmacher, Igor Scaliante Wiese, Tayana Conte, Marco Aurélio Gerosa e David Redmiles. "The hard life of open source software project newcomers". Em: *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE 2014. Hyderabad, India: ACM, 2014, pgs. 72–78. ISBN: 978-1-4503-2860-9. DOI: 10.1145/2593702.2593704. URL: http://doi.acm.org/10.1145/2593702.2593704 (citado nas pgs. 1, 8).

[I. Steinmacher, Conte *et al.* 2016] I. Steinmacher, T. U. Conte, C. Treude e M. A. Gerosa. "Overcoming open source project entry barriers with a portal for newcomers". Em: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. Mai. de 2016, pgs. 273–284. DOI: 10.1145/2884781.2884806 (citado nas pgs. 15, 16).

[Igor Steinmacher *et al.* 2018] Igor Steinmacher, Christoph Treude e Marco Aurelio Gerosa. "Let me in: guidelines for the successful onboarding of newcomers to open source projects". Em: *IEEE Software* 36.4 (2018), pgs. 41–49 (citado na pg. 16).

[Thanaki 2017] Jalaj Thanaki. *4.3 Basic Preprocessing*. Packt Publishing, 2017. ISBN: 978-1-78712-142-3. URL: https://app.knovel.com/hotlink/khtml/id:kt011DM3U2/python-natural-language/basic-preprocessing (citado na pg. 19).

[Uddin e Robillard 2015] G. Uddin e M. P. Robillard. "How api documentation fails". Em: *IEEE Software* 32.4 (jul. de 2015), pgs. 68–75. DOI: 10.1109/MS.2014.80 (citado na pg. 10).

[Wang e Sarma 2011] Jianguo Wang e Anita Sarma. "Which bug should i fix: helping new developers onboard a new project". Em: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. CHASE '11. Waikiki, Honolulu, HI, USA: ACM, 2011, pgs. 76–79. ISBN: 978-1-4503-0576-1. DOI: 10.1145/1984642.1984661. URL: http://doi.acm.org/10.1145/1984642.1984661 (citado na pg. 11).

[Zhi *et al.* 2015] Junji Zhi *et al.* "Cost, benefits and quality of software development documentation: a systematic mapping". Em: *Journal of Systems and Software* 99 (2015), pgs. 175–198. ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2014.09.042. URL: http://www.sciencedirect.com/science/article/pii/S0164121214002131 (citado na pg. 10).

[Zhou e A. Mockus 2015] M. Zhou e A. Mockus. “Who will stay in the floss community? modeling participant’s initial behavior”. Em: *IEEE Transactions on Software Engineering* 41.1 (2015), pgs. 82–99. ISSN: 0098-5589. DOI: 10.1109/TSE.2014.2349496. URL: https://ieeexplore.ieee.org/document/6880395 (citado na pg. 8).

[Zhong e Su 2013] Hao Zhong e Zhendong Su. “Detecting api documentation errors”. Em: *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications*. OOPSLA ’13. Indianapolis, Indiana, USA: ACM, 2013, pgs. 803–816. ISBN: 978-1-4503-2374-1. DOI: 10.1145/2509136.2509523. URL: http://doi.acm.org/10.1145/2509136.2509523 (citado nas pgs. 1, 13).