Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Programa de Pós-graduação em Ciência da Computação

Vinicius Junqueira Schettino

Uma ferramenta para recomendação de revisores de código para apoiar a colaboração em Desenvolvimento Distribuído de Software

Juiz de Fora

2018

Vinicius Junqueira Schettino

Uma ferramenta para recomendação de revisores de código para apoiar a colaboração em Desenvolvimento Distribuído de Software

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Marco Antônio Pereira Araújo

Juiz de Fora

2018

Ficha catalográfica elaborada através do Modelo Latex do CDC da UFJF
com os dados fornecidos pelo(a) autor(a)

Junqueira Schettino, Vinicius.
Uma ferramenta para recomendação de revisores de código para apoiar a colaboração em Desenvolvimento Distribuído de Software / Vinicius Junqueira Schettino. – 2018.
71 f.

Orientador: Marco Antônio Pereira Araújo
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-graduação em Ciência da Computação, 2018.

1. Palavra-chave. 2. Palavra-chave. 3. Palavra-chave. I. Pereira Araújo, Marco Antônio, orient. II. Título.

Vinicius Junqueira Schettino

Uma ferramenta para recomendação de revisores de código para apoiar a colaboração em Desenvolvimento Distribuído de Software

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Juiz de Fora, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

AGRADECIMENTOS

“Texto em que o autor apresenta uma citação, seguida de autoria, relacionada com a matéria tratada no corpo do trabalho”
(ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 2011, p. 2)
A epígrafe elaborada conforme NBR 10520 (Epígrafe - Opcional)

RESUMO

De acordo com a Associação Brasileira de Normas Técnicas - 6028 (2003, p. 2) “o resumo deve ressaltar o objetivo, método e as conclusões do documento (...) Deve ser composto de uma sequência de frases concisas, afirmativas e não de enumeração de tópicos. Recomenda-se o uso de parágrafo único.” O resumo deve ter de 150 a 500 palavras.

Palavras-chave: Palavra-chave. Palavra-chave. Palavra-chave.

ABSTRACT

...

Key-words: ...

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

UFJF	Universidade Federal de Juiz de Fora
DDS	Desenvolvimento Distribuído de Software

SUMÁRIO

1 INTRODUÇÃO

O *code review* é considerado uma das principais técnicas para diminuição de defeitos de software [6]. Nela, o autor de uma alteração na base de código de um projeto submete tal conteúdo ao crivo de um conjunto de pares técnicos, que irão revisar sua estrutura com base em um lista de regras e convenções previamente definida. Diferentes aspectos relacionados ao autor, ao revisor e ao processo de revisão em si estão diretamente relacionados à eficiência da prática. Autores relatam a diminuição da incidência de *anti-patterns* [7] de acordo com o nível de participação dos envolvidos e cobertura do código revisado [8, 9, 10]. Reputação [11, 12] e experiência [13] do revisor também parecem impactar nos efeitos do *code review*

Intrinsecamente colaborativa, a atividade de *code review* é exercida com suporte de ferramentas computacionais específicas [14], principalmente no desenvolvimento distribuído. Dentro de workflows de trabalho descentralizados [15], a prática funciona como um *gateway* de qualidade que busca garantir que apenas alterações aderentes aos padrões de qualidade do projeto serão incorporados à codebase principal. Esta etapa do desenvolvimento se torna uma oportunidade para disseminação de conhecimento, embate de ideias e discussão de melhores práticas entre profissionais de experiência e visões diferentes. Para tanto, percebe-se a necessidade de suporte computacional para essas atividades colaborativas.

Tais aspectos configuram o Desenvolvimento Distribuído de Software (DDS), onde equipes de desenvolvimento se encontram espalhadas por organizações e espaços geográficos distintos. Este novo ramo da Engenharia de Software vem modificando a relação entre empresas e sistemas, principalmente em relação às estratégias de negócios [16]. As próprias relações de negócios fomentam a distribuição das equipes, procurando diminuição dos custos e a incorporação de mão de obra qualificada que pode estar em qualquer lugar do planeta.

Neste contexto, porém, os os desafios à colaboração co-localizada são potencializados e as soluções tradicionais não são suficientes para fomentar esta aspecto das atividades distribuídas [17]. Casey [18] mostra que, com a distribuição geográfica dos times, diversos outros desafios, antes considerados colaterais ou resolvidos, emergem de forma a ameaçar a colaboração entre os membros da equipe: barreiras culturais, temporais e geográficas; reengenharia dos processos de desenvolvimento; resistência em compartilhar informações e conhecimento com os pares distribuídos; entre outros desafios.

Estes desafios do Desenvolvimento Distribuído de Software afetam o *code review* de duas formas distintas. Primeiro, o processo de revisão pode se tornar lento e ineficiente quando a colaboração é afetada, devido aos baixos níveis de participação e cobertura. O mesmo vale para a disseminação do conhecimento, que fica prejudicada. Outro desafio que se consolida é a escolha do revisor adequado para aquele *patch*. Com um vasto número de

opções e pouca informação disponível sobre seus aspectos técnicos e gerenciais (e.g. tempo disponível) já que não há contato co-localizado entre eles, a natureza distribuída deste tipo de desenvolvimento dificulta o processo de escolha do revisor, impactando negativamente a eficiência do processo.

Uma possível solução, visando amparar a colaboração e evitando o *overhead* da escolha do revisor, seria manter grupos bem testados e experientes exercendo as atividades de revisão. Ou ainda, fixar, dentro de cada equipe de desenvolvimento, quem são os responsáveis por revisão e pela submissão dos *patches*, evitando a diversificação das relações de trabalho.

Contudo, estudos recentes demonstram que a fixação de grupos e responsabilidades pode não ser benéfica para o processo de desenvolvimento. Scott Page [19] argumenta que a diversidade de experiências, visões e especilidades fazem com que grupos sejam mais eficientes. Já Prikladnicki et al. [20] apontam índicios de que a formação de grupos temporários em detrimento ou em conjunto com permanentes é um fator de eficiência em projetos de software:

> "Although old colleagues bring knowledge of the development process and prior norms from previous teams, new members bring fresh ideas that could promote project performance and creativity. Old colleagues might not do so and might not give new members a chance to implement their ideas."

Essa visão aponta que a formação dinâmica dos grupos de trabalho em desenvolvimento de software potencializa a disseminação do conhecimento, um dos objetivos primários do *code review* [14].

Existem alguns trabalhos congêneres que demonstram métodos de recomendação de revisores [21, 22, 23]. Esses trabalhos foram estudados e levados em consideração para escrita do presente texto. Também foram revisadas pesquisas que apontam caracterísitcas de revisões, revisores e autores que possivelmente potencializam a colaboração [7, 24, 11]. Tais aspectos são apresentados e discutidos no capítulo 3.

As principais lacunas deixadas pelos trabalhos anteriores estão relacionadas aos objetivos e à avaliação dos métodos propostos, principalmente em DDS. Primeiramente, não há relato de método de recomendação de revisores de código com o objetivo específico de potencializar a colaboração. Por isso, métodos já propostos não utilizam métricas nem variáveis de entrada relacionadas aos aspectos de cooperação, coordenação e comunicação, como por exemplo a abordagem 3C em DDS [25].

Outro ponto observado diz respeito à avaliação dos modelos de avaliação. Os trabalhos encontrados se limitam a comparar seus resultados com métricas relacionadas à proximidade dos mesmos com a indicação manual do revisor. Ou seja, a eficência é tida de acordo com a interseção entre o recomendado automaticamente e por decisão de um

especialista, geralmente um desenvolvedor. Este modelo assume que o responsável pela indicação manual tem os subsídios naturais para fazer uma boa escolha. Em DDS isso pode não ser verdade, uma vez que fatores como diferenças culturais, de horário, geográficas e de maturidade podem diminuir a compreensão do indicador e propiciar a escolha inadequada do revisor. Por isso, no contexto apresentado, outras formas de avaliação podem ser mais apropriadas. Tais discussão são extendidas no capítulo 4.

Expostos os desafios que o Desenvolvimento Distribuído de Software impõe sobre a escolha do revisor de código, a importância da indicação do revisor adequado do ponto de vista de colaboração e a motivação da formação de grupos heterogêneos e dinâmicos, sumariza-se o intuito do presente texto. De acordo com a abordagem QGM (Goal/Question/Metric) proposta por Basili et al. [26], postula-se o objetivo do trabalho como: **Implementar** um método de recomendação de revisores **com o objetivo de** potencializar a colaboração **em relação aos aspectos** de coordenação **do ponto de vista** de revisores e autores **no contexto de** desenvolvimento distribuído de software.

A principal hipótese que norteia o andamento desta proposta, e que será revisitada e discutida nos capítulos derradeiros é:

- O método de recomendação apresentado pode potencializar a colaboração entre revisores e autores.

O uso de ferramentas computacionais para o processo de revisão de código se tornou prática comum nos últimos anos [14]. O GitHub é uma plataforma rica em repositórios de projetos de software. Muitos são de código aberto, disponíveis para mineração. Discussões sobre o *workflow* de trabalho na ferramenta em contraponto à métodos tradicionais de revisão podem ser vistas na seção 2.1. São 24 milhões de usuários, 67 milhões de projetos e 47 milhões de revisões[1], também chamadas de *pull requests* no modelo de desenvolvimento "*pull based*" [1]. Essa abordagem é explorada na seção 2.1.3.

Esta característica permitiu a extração e análise automatizadas das informações sobre as revisões em projetos de código aberto, através de APIs disponibilizadas para este fim. Foram extraídas métricas apontadas como relevantes para nossos objetivos pela literatura relacionada. A arquitetura que embasa a extração e análise destes dados com objetivo de recomendação é explicada no capítulo 5.

A avaliação da eficiência do método proposto apresenta particularidades em relação à trabalhos relacionados, devido ao enfoque em colaboração no contexto de DDS. O método de avaliação é devidamente discutido e aplicado no capítulo 6, incluindo a apresentação dos experimentos e a revisitação da hipótese levantada neste capítulo. Por fim, o capítulo 7

[1] https://octoverse.github.com/

é dedicado ao fechamento do trabalho, inclusindo a sugestação de trabalhos futuros e a discussão de ameaças a validade e generalização dos resultados apresentados.

2 PRESSUPOSTOS TEÓRICOS

2.1 *Code review*

O *code review* é uma prática consolidada e difundida em diversas organizações, contemplando diferentes portes e segmentos de mercado. A técnica constitui da análise técnica de uma mudança a ser submetida à base principal de código (repositório-mestre) por parte de um revisor técnico, tendo como base uma lista de diretrizes e padrões a serem observados. As nuances do processo variam em cada contexto levando em consideração, por exemplo, tolerância a defeitos, modelo de desenvolvimento e os objetivos almejados.

2.1.1 Relevância

O *code review* está associada diretamente à detecção precoce de defeitos em produtos de software [27, 7], sendo reconhecida como uma das principais técnicas com este fim [6]. Mais especificamente, é relatada maior eficiência quanto aos defeitos não-funcionais, enquanto os defeitos funcionais são menos afetados no processo [28]. Outros autores reportam a diminuição de defeitos através de estudos de caso [29, 10, 9].

2.1.2 Histórico

A atividade de revisão remonta da década de 80 [30], e desde então vem evoluindo para suportar interações mais rápidas e constantes, com uso de ferramentas computacionais e práticas ágeis. O Modern Code Review (MCR) surge em sinergia com os modelos ágeis e distribuídos de desenvolvimento, valorizando mais a comunicação e troca de experiências entre autor e revisor [14].

2.1.3 Pull Based Method

O conceito de *branches* é a base para sistemas de controle de versão descentralizados, como o Git[1] e o Mercurial[2]. Com as *branches* é possível desenvolver paralelamente, subm-tendo e mesclando as alterações no código em momentos oportunos. Esta característica é interessante para o DDS, uma vez que o isolamento e a atomicidade do trabalho de cada um até o momento de submissão é fundamental para a coordenação dos esforços [31].

Estas tecnologias permitiram o surgimento de um paradigma de desenvolvimento baseado em pulls, ou *pull-based method* [1]. O processo de revisão de código evolui neste novo paradigma, servindo como um *gateway* de qualidade que busca garantir que apenas alterações aderentes aos padrões de qualidade do projeto serão incorporados à codebase principal [32]. A figura 1 ilustra tal modelo de trabalho instanciado no GitHub[3],

[1] https://git-scm.com/
[2] https://www.mercurial-scm.org/
[3] https://github.com

principal expoente que oferece este paradigma. Nele é representado um modelo comum em desenvolvimento OpenSource [33], onde há um *core team* responsável por revisar os *pulls* de seus colegas e da comunidade no geral. Neste modelo, a mudança chega à codebase principal somente se houver o aval de um membro do *core team*.

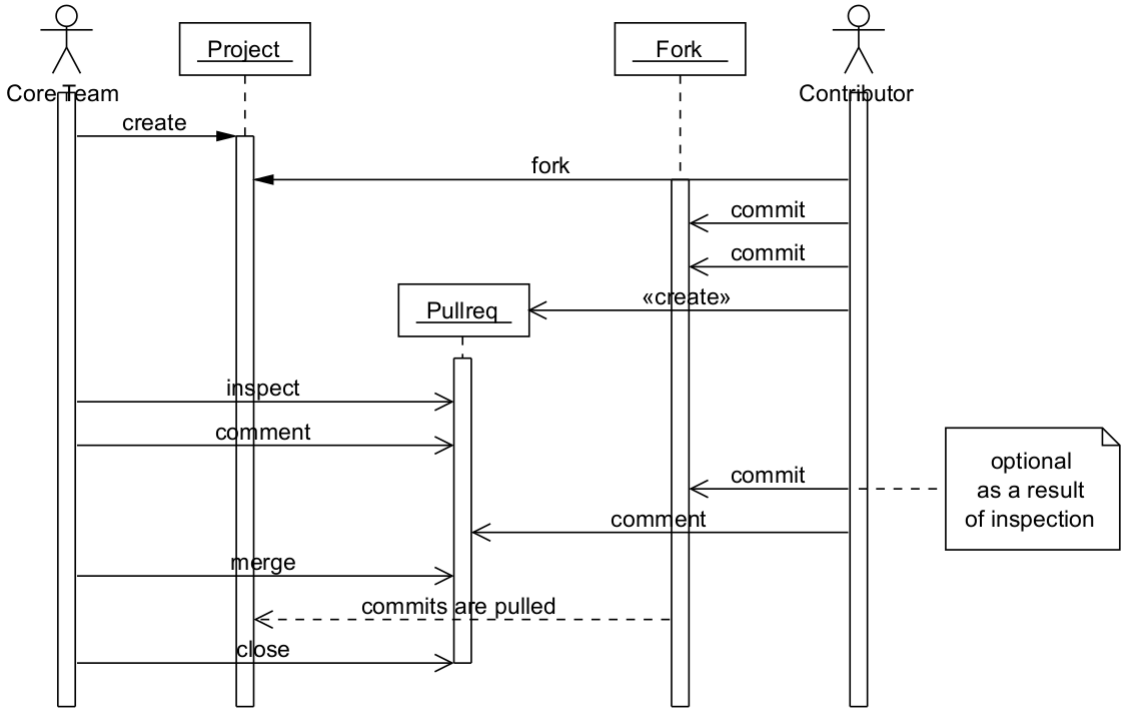

Figura 1 – processo do *"pull request"* [1]

Aquele que deseja contribuir cria para si uma cópia do projeto através de um *fork*. Esta ação cria em seu diretório de trabalho um projeto idêntico ao original, mas ao qual ele tem acesso total de submissão e modificação. Nessa cópia, ele executa as modificações desejadas, geralmente em uma *branch* dedicada para tal [15]. Ao terminar, ele solicita a integração da branch do *fork* de volta ao projeto original. Essa solitação é chamada de *pull-request*, que será analisada por um desenvolvedor com as devidas permissões. Durante esta revisão, o autor pode gerar novas modificações, geralmente atreladas aos pedidos do revisor. Ao final, a mudança é rejeitada (*closed*) ou aceita *merged*.

Os membros do core também têm suas *branches* revisadas por um processo análogo [33, 12]. A principal diferença é que não há necessidade do fork, já que eles tem as permissões necessárias para criar uma nova *branch* no projeto-alvo.

2.2 Desenvolvimento Distribuído de Software

O Desenvolvimento Distribuído de Software é uma abordagem em crescente utilização no cenário atual [34]. Este *workflow* descentralizado é caracterizado por membros das equipes de trabalho localizadas em lugares distintos espalhados pelo globo. Modalidades

de *home office* e *freelancer* são comuns nestes contextos, mas basta que um membro de uma equipe esteja geograficamente disperso para caracterizar este fenômeno [35]. Esta mudança no paradigma tradicional de desenvolviemento colocalizado é patrocinada pelas organizações com o objetivo de reduzir custos, ter acesso a mão de obra mais qualificada e especialmente se manter competitivos num mercado cada vez mais concorrido [36].

Assim, os aspectos técnicos do processo produtivo devem se alinhar aos objetivos globais das organizações para dar suporte ao desenvolvimento distribuído e encarar os diversos desafios que permeiam esta mudança. Entre eles, é possível destacar o embate de aspectos socio-culturais, geográficos e de fuso horários. Em suma, estas caracterísitcas inerentes ao desenvolvimento distribuído ameçam a produtividade ao dificultar os processos que envolvem coodernação, comunicação e cooperação [37].

3 REFERENCIAL TEÓRICO

Levando em consideração a complexidade e a importância da recomendação de revisores de código, associados com a grande variedade de métodos já propostos para tal fim, foram conduzidas uma revisão e um mapeamento da literatura relacionada «citar aqui». No presente trabalho os principais aspectos e resultados deste estudo serão apresentados, com ênfase naqueles particularmente relevantes aqui. De acordo com parâmetros e diretrizes estabelecidos, [38] o objetivo é apresentar o estado da arte das ferramentas, métodos e potenciais funcionalidades que suportem a escolha de revisores de código. Conhecendo o comportamento histórico do campo, podemos nutrir o desenvolvimento de novas soluções embasadas nas contribuições mais recentes com foco nas lacunas existentes e capazes de contribuir com a evolução das pesquisas relacionadas.

De acordo com estes objetivos, buscamos apresentar em quais contextos a técnica de recomendação automatizada é mais relevante, e quais são as métricas que os pesquisadores utilizam para avaliar e comparar as abordagens propostas. Estas análises podem auxiliar a criar uma estrutura sólida para que novos métodos possam ter sua eficiência avaliada de maneira direta e reprodutível.

Existem tentativas anteriores de reunir e avaliar diferentes métodos de recomendação de revisores, apesar de nenhuma delas envolver um processo sistemático de revisão da literatura relevante. Por isso, não há uma clara classificação dos modelos e os aspectos de reprodutibilidade e replicação dos estudos foram negativamente afetados. Yang et al. [39] apresentam diversos estudos e aplicam um método conhecido numa grande base de dados, tirando conclusões sobre como a eficência dos revisores ativos é muito maior do que aqueles que não participam do processo de forma contundente. Os autores também apresentam uma análise detalhada da base de dados utilizada, possibilitando entender melhor o contexto e a importância do comportamento dos envolvidos nos resultados do processo.

Hannebauer et al. [40] reconhecem que encontrar revisores adequados é um desafio e focam em comparar empiricamente oito diferentes abordagens em quatro conhecidos projetos *open source*. Já Consentino et al. [41] apresentam uma revisão sistemática ampla, sobre o processo de desenvolvimento baseado em *pull requests* do GitHub, citando alguns pontos da revisão de código. Apesar de importantes para compreensão da área de pesquisa, as contribuições anteriores não contam com o rigor de uma revisão sistemática da literatura, prejudicando aspectos de reprodutibilidade, adaptabilidade, replicabilidade. Assim, justificamos a necessidade de conduzir uma revisão sistemática para dar suporte teórico à este trabalho, evitando ainda viés por parte dos autores e nutrindo conclusões baseadas em evidências [42].

3.1 Revisão sistemática da literatura

As diretrizes para embasar pesquisas sistemáticas em Ciência da Computação foram propostas por Kitchenham [38], ao adaptar abordagens e outras áreas, especialmente das Ciências Méticas. Enquanto o mapeamento traz um panorama geral do desenvolvimento de uma área, a revisão sistemática foca em questões mais específicas e objetivas, muitas vezes relacionadas aos resultados dos estudos. [42]. Apesar de geralmente seguir o mesmo protocolo, o escopo, critérios e questões de pesquisa são distintos. Levando em consideração os objetivos deste trabalho a configuração atual da área de pesquisa, ambas as abordagens podem ser úteis. Todas etapas do estudo foram concretizadas por quatro diferentes especialistas, em análises distintas. Todos os critério de inclusão e exclusão dos trabalhos e eventuais divergências foram discutidos em profunidade antes da definição dos resultados.

3.1.1 Questões de Pesquisa

No trabalho conduzido, as seguintes questões foram escolhidas para caracterizar a área de recomendação de revisores de código:;

3.1.1.1 **MQ1: Quem são os principais autores na área?**

Ao identificar os principais autores, embasamos os futuros pesquisadores da área com um ponto de partida para leitura e acompanhamento.

3.1.1.2 **MQ2: Quais são os principais meios de publicação na área?**

O tipo de publicação e conceituação do meio podem ser evidências da maturidade do campo de pesquisa e atenção direcionada pela comunidade acadêmica.

3.1.1.3 **MQ3: Em quais contextos a recomendação de revisores de código é utilizada? (Desenvolvimento global de sofware, indústria, Código Livre)**

O objetivo é endender quais contextos potencializam a necessidade de métodos automatizados de recomendação.

Buscando por um panorama mais espeífico e objetivo, foram propostas as seguintes questões de pesquisa para serem analisadas através da revisão sistemática:

3.1.1.4 **RQ1: Como a eficiência da recomendação dos revisroes é mensurada?**

Para propor novos métodos de recomendação é importante conhecer as métricas de avaliação empregadas nos trabalhos anteriores de forma a fundamentar a comparação dos resultados.

3.1.1.5 **RQ2: Quais informações dos repositórios de software são utilizadas para recomendar os revisores?**

Para propor novos métodos de recomendação é importante entender quais informações foram utilizadas em propostas anteriores, sendo assim possível estendê-las e discutir novas abordagens e aplicações.

Levando em consideração objetivos da abordagem sistemáticas (definidas pelo processo GQM [26]) e os termos definidos pelo PICOC [43] a *string* de busca foi construída. Termos similares e sinônimos foram incluídos para obter um espectro maior de pulicações. Termos como *"pull-request"* foram adicionados para garantir que trabalhos sobre "pull-based software development" (como ilustrado na seção 2.1.3) fossem encontrados. Assim, a seguinte *string* foi gerada:

> *("software developer"AND developer) AND ("reviewer recommendation"OR "commenter recommendation") AND (method OR tool OR solution OR framework) AND ("code review"OR "pull-request"OR "pull request")*

A *string* de busca proposta foi revisada pelos autores do trabalho anterior separadamente, assim como por um pesquisador externo com o objetivo de validar a sua composição, relevância e emprego e termos correlatos. Buscas *ad-hoc* foram conduzidas para validar o uso dos termos na área e foram encontradas em trabalhos relevantes.

A busca foi executada, com pequenas modificações de sintaxe para aderir aos padrões das diferentes bases de dados. Os resultados foram explorados com o auxílio da ferramenta Parsifal [1], muito útil para acompanhar o processo sistemático proposto por Kitchenham [38]. Na ferramenta foi possível remover trabalhos duplicados, classificar os restantes de acordo com o critério de exclusão e categorizar os restantes de acordo com a relevância para o trabalho. A figura 2 descreve este processo.

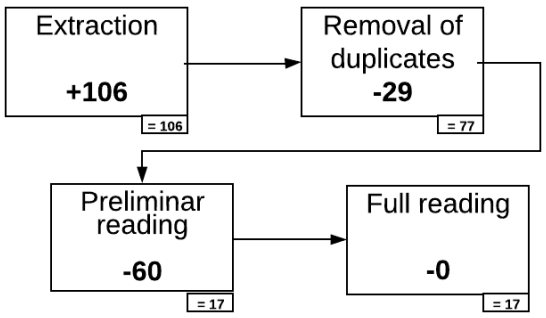

Figura 2 – Condução do protocolo sistemático

[1] http://parsif.al

Ao todo, 106 trabalhos foram encontrados. Destes, 29 foram considerados duplicados, enquanto 60 foram rejeitados de acordo com os critérios de exclusão. A principal razão foi não apresentar um método de recomendação com evidências empíricas de eficiência. Os 17 trabalhos restantes passaram pela leitura completa, constatando-se que todos cumpriam os critérios para inclusão na análise. Respondendo a **MQ2**, a tabela 1 indica os trabalhos selecionados separados por meio de publicação. o *h5-index* indica o fator de citação da de cada veículo nos últimos 5 anos.

Reference	Channel	h5-index
[44][45]	ICSE - IEEE/ACM International Conference on Software Engineering	68
[46]	IEEE Transactions on Software Engineering	52
[47]	GLOBECOM - IEEE Global Communications Conference	48
[48]	SIGSOFT - ACM International Symposium on Foundations of Software Engineering	43
[49]	Journal of Intelligent Information Systems	22
[50]	Journal of Computer Science and Technology	22
[51]	IEICE Transactions on Information and Systems	17
[52]	SANER - IEEE International Conference on Software Analysis, Evolution and Reengineering	13
[21, 53]	APSEC Asia-Pacific Software Engineering Conference	12
[54]	IEEE/ACM International Workshop on Software Mining	*
[55]	International Workshop on CrowdSourcing in Software Engineering	*
[56][57][58]	ICSME - IEEE International Conference on Software Maintenance and Evolution	*
[59]	IWCSN - International Workshop on Complex Systems and Networks	*

Tabela 1 – Publicações por veículos

Os resultados obtidos através da análise da **MQ3** servem para direcionar ferramentas de recomendação em contextos onde este tipo de técnica se faz mais relevante. É possível observar que a recomendação de revisores etá relacionada ao desenvolvimento destribuído e ao desenvolvimento "pull-based". Esta associação parece ser embasada aos desafios interentes do desenvolvimento global. Ying et al. [55] justificam este fenômeno através do maior número de revisões e revisores, o que dificulta a seleção manual dos envolvidos. Yu et al. [21] mostram que encontrar o melhor revisor para um "pull request" é um trabalho tipicamente de "crowdsourcing". Assim, para a escolha manual é necessário validar a opinião de muitas pessoas, o que dificulta o processo. Neste contexto também é comum contar com uma riqueza maior de informações graças ao emprego massivo de ferramentas de suporte ao desenvolvimento, onde é a agregação e utilização destes dados ainda é um desafio [44].

A maior parte dos trabalhos listados utilizou repositórios de dados *"Open Source"* para testar os métodos propostos, enquanto alguns foram além de utilizaram mecanismos e informações específicas deste tipo de desenvolvimento para criar ferramentas especializadas neste contexto. Devido ao ambiente de rápida interação e grande quantidade de contribuidores esporádicos, as revisões adequadas (muitas vezes movidas pela escolha dos revisores adequados) podem aumentar a retenção de desenvolvedores e influenciar no sucesso do projeto [59]. Neste tipo de processo produtivo é comum existir um grupo seleto de desenvolvedores experientes que agem como guardiões da qualidade, padronização e objetivos dos projetos durante o processo de revisão. Como estes são responsáveis por

um conjunto muito grande de revisões [53], o atraso no assimilação do código proposto à codebase principal pode atrasar as entregas e desencorajar novas contribuições [50]. Assim, recomendaços automáticas podem distribuir melhor a carga de revisão e aproveitar melhor as especialidades de cada membro da equipe.

Diante do exposto, os métodos de recomendação deste trabalho levam em consideração aspectos do desenvolvimento distribuído em sua concepção. O desenvolvimento *"Open Source"* é um dos contextos mais maduros e carantes deste tipo de abordagem, e por isso são o principal alvo da ferramenta e campo para avaliação do proposto. Além disso, este tipo de repositório é mais acessível, o que auxilia na realização de experimentos auditáveis e reprodutíveis.

3.2 Categorização dos principais trabalhos

Com o objetivo de responder à **RQ2**, a revisão sistemática separou os trabalhos selecionados em quatro categorias distintas, de acordo com os dados nos quais cada um dos métodos apresentados utiliza.

3.2.1 Experiência dos revisores

Os métodos classificados nesta categoria utilizam a experiência de um revisor como principal informação para recomendá-lo. A lógica é que se ele atuou em revisões parecidas (tanto em região do código quanto em outros aspectos), ele será adequado para a revisão atual.Enquanto Fejzer et al. [49] propõem utilizar a similradidade do código submetido com o perfil do revisor, Liao et al. [47] definem um conjunto de tópicos que descrevem a experiência e que pode ser comparado com a necessidade da revisão atual. Thongtanunam et al. [52] apresenta um abordagem onde o caminho dos arquivos revisados são comparados com o histórico de revisão dos potenciais indicados.

3.2.2 Experiência do Desenvolvedor

Alguns métodos inferem que a experiência do desenvolvedor em contribuições no passado podem indicar na sua capacidade em revisar modificações parecidas. Costa et al. [48] analisam quais foram os responsáveis por mudanças desta região para encontrar candidatos. Rahman et al. [44] apresentam o CORRECT, um método que analisa a experiência dos desenvolvedores em projetos diferentes mas que contam com tecnologias compartilhadas para recomendar.

3.2.3 Redes Sociais

Uma das abordagens compartilhada entre os métodos mais novos é utilizar os relacionamentos entre desenvolvedores para encontrar os revisores adequados. Ou seja, a

partir do histórico de interação (revisão, colaboração, comentários, respostas) é possível inferir quais seriam os melhores revisores para uma nova mudança. Estes relacionamentos são geralmente representados como grafos, construídos com ajuda de análises textuais, proximidade semântica e tópicos das interações passadas. Fu et al. [59] propõe utilzar um grafo baseado nas relações sociais dos desenvolvedores. Xia et al. [54] captura relações implícitas e valoriza interaços mais recentes. Yu et al. [21, 56] estende métodos tradicionais em classificação de ocorrências (defeitos, solicitações e etc) para aplicação em recomendação revisores, extraindo informações de discussões e comentários em desenvolvimento global. Yang et al. [51] emprega análise de redes sociais para encontrar revisores baseado na intensidade dos relacionamentos.

3.2.4 Abordagens Híbridas

Alguns métodos reúnem diferentes classes de informação para tentar assimilar as melhores características de cada uma delas. Através da análise de um grafo, Ying et al. [55] consideram tanto a experiência dos desenvolvedores quanto a autoridade no processo de desenvolvimento. Através da ferramenta CoreDevRec, Jiang et al. [50] utilizam o caminho dos arquivos modificados, a relação pregressa e a atividade dos envolvidos.

Ainda de acordo com a revisão sistemática conduzida, as abordagens baseadas em redes sociais são mais novas e alvo dos trabalhos com mais repercussão na área. Assim, para o escopo deste trabalho foi decidido explorar este tipo de abordagem, que está intimamente relacionado com o desenvolvimento distribuído.

3.3 Métricas de avaliação

Respondendo a **RQ1**, o trabalho de revisão sistemática identifica que a maior parte dos trabalhos utiliza métricas clássicas de sistemas de recomendação para avaliar os métodos, como Top-k *Precision*, *Recall* e *Hit*. Ou seja, é avaliada a proximidade do conjunto de tamanho k que foi recomendado pelo método em relação ao conjunto que um especialista escolheu. Assim, é uma avaliação que compara a capacidade do algoritmo proposto em relação ao que um ser humano especialista faria. a métrica de precisão (*Precision*) mostra o quão parecidos são os conjuntos, enquanto o *recall* aponta quantos items do conjunto do especialista foram "esquecidos" pelo método proposto. Já o *hit* indica se pelo menos um membro do conjunto apontado pelo especialista foi considerado pelo algoritmo.

Em contraponto, a revisão sistemática mostrou que alguns autores propõem métodos mais sofisticados para avaliação dos métodos. Esta necessidade se justifica ao considerar que os humanos que indicam revisores não necessariamente são especialistas nesta tarefa. Isso acontece especialmente no desenvolvimento distribuído, onde não é fácil reunir informações sobre todos os envolvidos para embasar estas decisões. Além disso,

aspectos de disponibilidade, especialidade, horários e questões culturais deixam ainda mais improvável que os grupos de revisão definidos possam ser considerados como os ideais. Assim, a proximidade do conjunto sugerido com o conjunto efetivamente formado não é suficiente para avaliar os métodos.

Diversas avaliações alternativas são citadas. Yang et al. [51] leva em consideração qual o impacto das recomendações na interatividade das revisões. Outros trabalhos também discutem o impacto positivo dos seus métodos na efiência do processo de revisão [54, 47]. A análise focada na eficência e impacto dos métodos de revisão é motivação do presente trabalho, e fomenta aspectos que são detalhados na seção 3.4

3.4 Métricas para avaliação do Code Review

Para propor modelos de avaliação voltados para o impacto dos métodos em relação à eficiência do processo de revisão, é necessário estabelecer parâmetros de comparação que embasem tais análises. Além dos aspectos como número de comentários e respostas e tempo de resposta discutidos por Yang et al. [51]. Trabalhos prévios apresentam outras métricas relacionadas à avaliação do processo de revisão, que podem ser aplicadas no contexto da recomendação ao comparar o desempenho de dois grupos distintos de participações em revisões: os sugeridos pelo método e aqueles que não foram. Assim é possível medir se os revisores recomendados tiveram um desempenho melhor durante o processo.

Bosu et al. [60] conduziram um estudo empírico na Microsoft onde avaliaram manualmente a pertinência de milhares de comentários de revisões. Assim, buscaram descobrir quais características das revisões, revisores e comentários que levaram às interações mais positivas no processo: mais mudanças (frutos de discussões relvantes), adesão aos padrões de desenvolvimento e receptividade por parte dos autores. Já Rahman et al. [61] analisaM os textos das discussões e encontra correlação da pertinência do revisor com comentários mais úteis. As principais métricas que podem ser objetivamente avaliadas são:

- Tempo curto de respostas;
- baixo índice de “stop words”;
- interatividade (respostas, comentários, etc);
- sentimento positivo;
- tamanho dos comentários.
- capacidade de gerar mudanças

Através de métricas como tempo de resposta e interatividade, é possível avaliar se os revisores indicados potencializam a colaboração e a produção de conhecimento através

da discussão, reflexos da revisão de código que devem ser fomentados [8, 9, 10]. Aspectos como sentimento positivo, baixo índice de "stop words" e tamanho dos comentários e a capacidade de gerar mudanças podem indicar que o revisor escolhido foi capaz de interagir positivamente com o processo, gerando valor agregado em soluções melhores e teve perfil e experiências compatíveis para contribuir com código em discussão [60].

Tendo como base os resultados da revisão sistemática, da leitura dos trabalhos relacionados, as análises contidas nos capítulos anteriores e o direcionamento aos principais desafios da pesquisa em recomendação de revisores de código, foi possível definir os seguintes tópicos basilares para o trabalho:

- O método proposto é direcionado ao contexto de desenvolvimento distribuído, com particularidades para aplicação e avaliação no desenvolvimento *Open Source*;
- o método proposto utiliza aspectos e análises de redes sociais e relacionamentos entre os desenvolvedores para realizar as recomendações;
- o processo de avaliação é pautado na performance dos indicados e de suas interações no processo de revisão.

Com estes direcionamentos, o capítulo 4 apresenta a análise exploratória dos datasets escolhidos e embasamento empírico dos métodos propostos.

4 METODOS DE RECOMENDAÇÃO

Em convergência com as diretrizes basilares lançadas nos capítulos anteriores, esta seção descreve as análises da colaboração entre desenvolvedores em repositórios "OpenSource" que culminaram na proposta de métodos de reomendação para revisão de código. Além da compreensão das características das redes de interação formadas nos repositórios, foram definidos mecanismos para evitar enviesamento dos resultados e excesso de restrição da validade das propostas a determinadas tecnologias ou projetos.

4.1 Redes de desenvolvedores

Interações durante do desenvolvimento de software caracterizam o surgimento de redes sociais de desenvolvedores, denominadas *"social networks"* ou *"developer networks"* [62]. Observar tais estruturas como grafos é o ponto de partida de diversas aplicações deste tipo de análise, como previsão de defeitos [63], triagem de tarefas de manutenção [64] e encontrar os melhores profissionais para responder determinada dúvida [65]. Diversas informações estão disponíveis para a modelagem de tais redes, tanto diretamente das ferramentas de controle versão quanto em ambientes de desenvolvimento mais sofisticados, como o GitHub[1] e o GitLab[2]. Assim, de acordo com o contexto de cada pesquisa, é possível selecionar quais tipos de interação e seus atributos serão utilizados como vértices, arestas e pesos das representações.

4.2 Modelagem da rede

Uma rede social pode ser representada como um conjunto de nós conectados entre si. O objetivo é refletir nesta estrtura a relação entre os indivíduos e a forma que eles interagem. Dentre os dados disponíveis, existem diferentes óticas nas quais os dados podem ser modelados e analisados, de acordo com as conclusões esperadas. No caso deste trabalho, o interesse recai nas interações entre desenvolvedores no processo de *"pull request"* e de revisão de código. Durante esta etapa do desenvolvimento, o revisor interage com o autor comentando em partes específicas do código buscando sanar dúvidas, melhorar a implementação ou evitar que mudanças fora do padrão de codificação sejam admitidas no repositório. A figura 3 representa tal interação.

Através da API RESTful[3] do GitHub é possível interagir com o repositório e obter estas informações automaticamente, possibilitando a modelagem da rede.

[1] https://github.com
[2] https://gitlab.com
[3] https://developer.github.com/v3/

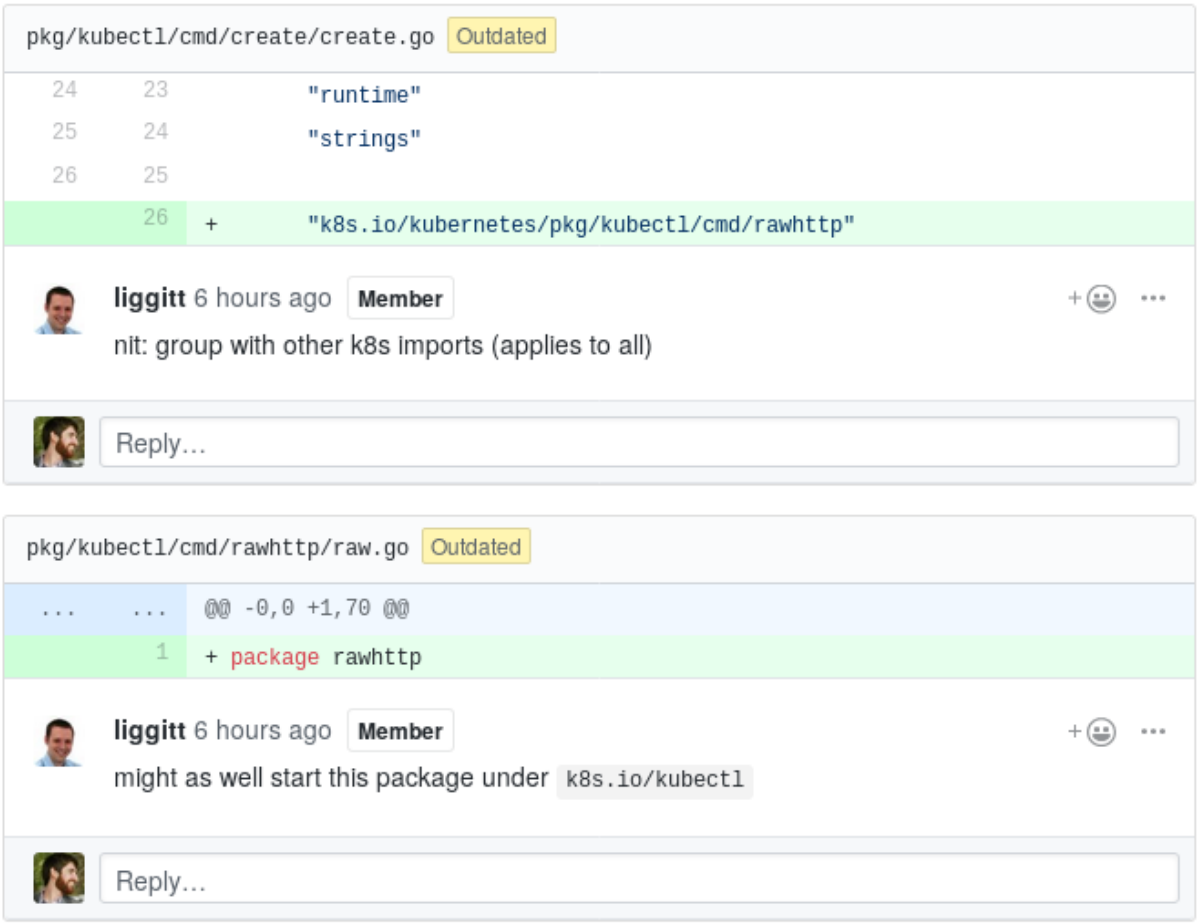

Figura 3 – Interação entre autor e revisor durante a revisão

4.2.1 Principais características

No modelo proposto, os desenvolvedores são os nós e as arestas direcionadas entre eles representam os comentários de revisão durante o *"pull request"*. Quando um desenvolvedor cria um comentário em uma revisão, um relacionamento entre ele e o autor é criado ou atualizado. Assim, o modelo é um grafo direcionado $G = (V, E)$ onde $V = v_0, v_1, ..., v_n$ representa o conjunto de n desenvolvedores e E o conjunto de triplas (arestas) $e_{ij} = (v_i, v_j, w)$ entre indivíduos v_i e v_j. O peso w é formulado para representar como determinado desenvolvedor influencia outro na perspectiva da revisão. Este valor representa o quão influente é um desenvolvedor sob outros em detrimento do conjunto todo. Ele é calculado considerando cada contribuição k do desenvolvedor v_i para o v_j onde $contrib(v_i, v_j, v_k)$. Valores mais altos de w indicam maior influência de v_i em v_j. A figura exemplifica como são representadas as relações entre os desenvolvedores 4.

Para evitar que interações antigas de desenvolvedores que podem nem estar ativos mais no projeto enviesem as análises, tais ocorrências são penalizadas de modo a valer menos do que interações mais novas. Essa situação é comum em projetos *"Open Source"* [66], além dos casos onde novos desenvolvedores chegam para ocupar as lacunas deixadas

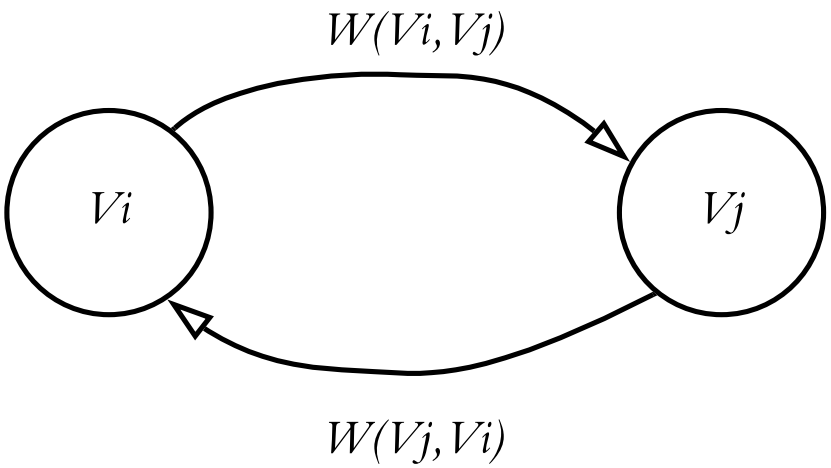

Figura 4 – Representação dos vértices e arestas da rede

neste processo. A figura mostra como esta situação se dá com o tempo, nos gráficos de contribuições (número de *commits*). Enquanto o maior contribuidor (e criador) do Node.js deixa o projeto em 2014, o segundo maior contribuidor chega e continua até os dias de hoje. O #3 chegou começa seu interação com o projeto mais tarde, enquanto o #4 foi conteporâneo do idealizador do projeto mas se desligou pouco tempo após sua saída.

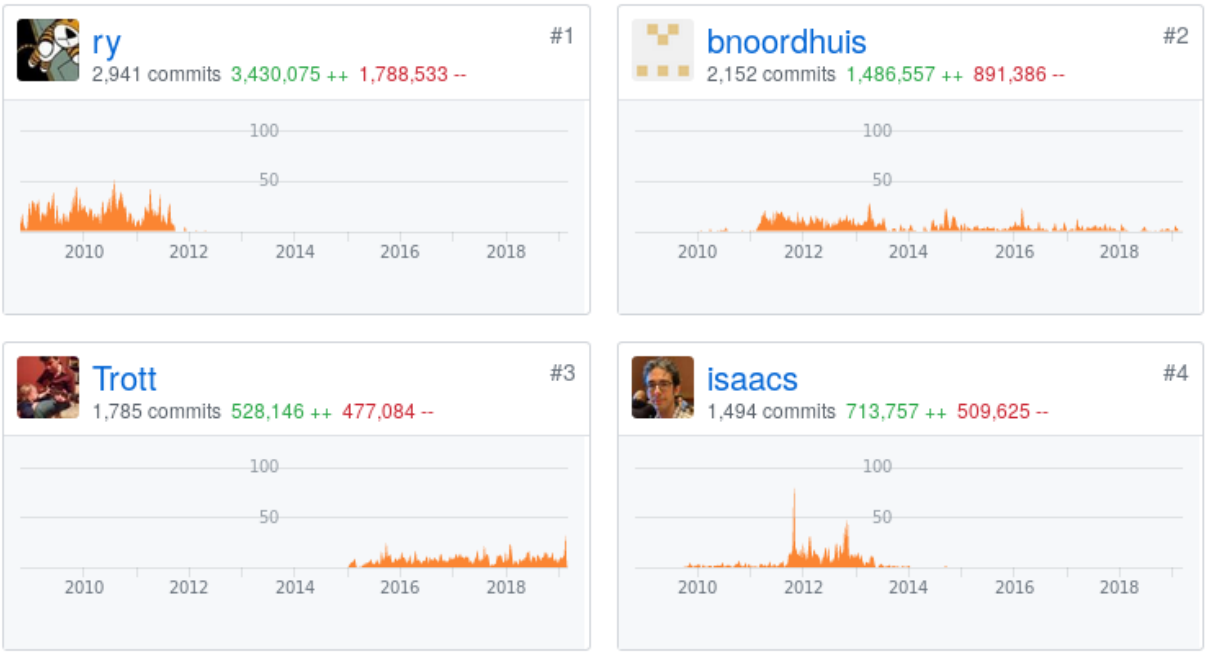

Figura 5 – Troca de participação entre os principais membros do projeto

Assim, o valor de cada comentário k de v_i para v_j decai exponencialmente dquanto mais antigo ele se torna. O valor total $P(v_i, v_j)$ eq:penalization é a soma de todas as

contribuições de v_i para v_j. A função $inter(v_i, v_j, k)$ retorna cada interação k entre os indivíduos v_i e v_j.

Com os valores agregados definidos por $P(v_i, v_j)$, o peso w_{ij} é então calculado eq:weight. $W(v_i, v_j)$ representa qual a participação das interações em direção à v_j vieram de v_i. Assim, w_{ij} é sempre um número entre $(0, 1]$. Quanto mais próximo w_{ij} é de 1, maior a influencia de v_i sobre v_j. Desta forma, $w_{ij} = 1$ significa que v_i é responsável por todas as interações que v_j recebeu.

$$P(v_i, v_j) = \sum_{k=1}^{n} \frac{1}{\exp days(inter(v_i, v_j, k))} \tag{4.1}$$

$$W(v_i, v_j) = \frac{P(v_i, v_j)}{\sum\limits_{k=1}^{n} P(v_j, v_k)} \tag{4.2}$$

4.3 Análise exploratória da rede proposta

Para avaliar a topologia da rede proposta, é necessário instanciar o modelo utilizando dados de projetos de software reais. Assim é possível compreender se as distribuições, cararacterísticas e comportamentos do grafo são compatíveis com o descrito na literatura e se tal contexto permite conclusões relevantes para os objetivos deste trabalho.

4.3.1 Escolha dos repositórios

A escolha dos repositórios influencia na validade das conclusões no contexto estudado e na relevância dos métodos propostos. Por consequência, as seguintes diretrizes foram traçadas para guiar a busca e avaliação dos projetos cujos dados serão território de avaliação e análise desta pesquisa. São elas:

1. O conjunto de repositórios deve ser composto por projetos diversos em tecnologia;
2. os projetos escolhidos devem ser conhecidos e de popularidade verificável em seus respectivos nichos;
3. cada projeto deve conter quantidades razoáveis de *"pull requests"* e colaboradores ativos;
4. os projetos escolhidos devem fornecer regras claras de contribuição, responsabilidade e governança.
5. os projetos devem estar disponíveis publicamente no GitHub

A diretriz 1 auxilia a diminuir o viés a determinada linguagem, propósito, topologia e outros fatores técnicos. A diretriz 2 possibilita a verificação dos resultados com mais

naturalidade e leva à potencialização da diretriz 3, que evita que os resultados sejam enviesados para projetos muito pequenos. A diretriz 4 possibilita que diferentes análises possam ser avaliadas de acordo com particularidades do processo de trabalho de cada repositório. Ao seguir a diretriz 4, é possível garantir que os dados serão acessíveis através da interface homogênea disponível para este fim. Os projetos selecionados constam no top-10 "projetos mais revisados"[4] do GitHub em 2017. A lista é sumarizada na tabela 4 e apresentada com mais detalhes nas subseções posteriores.

Projeto	**Linguagem Principal**	***"Pull requests"***	**Estrelas**	**Contribuidores**
Node.js	JavaScript	18.106	62.521	2.490
Kubernetes	Go	49.946	54.849	2.186
Symfony	PHP	20.002	21.088	1.891
Tensorflow	C++	11.258	130.341	2.055

Tabela 2 – Projetos e principais características

4.3.1.1 Node.js

Node.js[5] é um framework JavaScript construído sobre o motor V8 do Google Chrome[6]. Entre suas principais aplicações está a construção de aplicações "server-side" assíncronas e não bloqueantes, voltadas para atender um grande número de clientes simultaneamente. É um projeto ativo no GitHub, sendo o décimo projeto JavaScript com mais estrelas e o 24º global[7]. Além disso possui uma estrutura bem definida de governança[8] que dispõe da tomada de decisões importantes, contribuições da comunidade e organização de trabalho. Estas características fazem com que as análises das interações de seus participantes possa ser avaliada de acordo com critério objetivos e respaldados pelos responsáveis pelo projeto.

4.3.1.2 Kubernetes

Kubernetes[9] é um projeto *"Open Source"* escrito em Go voltado para gerenciar softwares em *containers* distribuídos. O projeto provê mecanismos básicos para implantação, manutenção e escalabilidade destas aplicações. Apresenta políticas de segurança bem definidas e documentação das fases do processo de revisão[10] que ajudam a avaliar as hipóteses levantadas durante este trabalho. Contém praticamente 50.000 *"pull requests"*,

[4] https://octoverse.github.com/2017/
[5] https://github.com/nodejs
[6] https://v8.dev/
[7] https://github.com/search?q=stars%3A%3E1&s=stars&type=Repositories
[8] https://github.com/nodejs/node/blob/master/GOVERNANCE.md
[9] https://github.com/kubernetes/kubernetes
[10] https://github.com/kubernetes/community/blob/master/contributors/guide/owners.md

maior número entre os projetos avaliados. É também o campeão em discussões de toda a plataforma GitHub.

4.3.1.3 Symfony

Symfony[11] é um dos mais populares e antigos frameworks PHP, com extensa utilização em aplicações web. Possui políticas claras de contribuição em casos de vulnerabilidades[12] e é a base de diversos *Content management system* (CMS) famosos, como o Drupal e o Joomla.

4.3.1.4 Tensorflow

Tensorflow é o projeto mais popular dos analisados neste trabalho, além de ter o maior número de *forks* do GitHub e ser o quinto com mais contribuidores. A tecnologia é desenvolvida em C++ com interface em Python com o objetivo de dar suporte para técnicas de aprendizado de máquina. O processo de revisão é bem documentado e a participação da comunidade são encorajadas diretrizes claras de contribuição, como mostra a figura 6.

4.3.2 Resultados

Os dados foram extraídos através da API do GitHub e carregados em uma instância do Neo4j[13], um sistema gerenciador de banco de dados orientado a grafos. Para entender como os indivíduos interagem nas redes propostas, algumas análises foram feitas e os principais resultados são detalhados nesta seção.

Ao calcular a distribuição de grau da rede, é possível entender como os indivíduos partilham a responsabilidade das revisões entre eles. No grafo direcionado, esse valor representa com quantos indivíduos um revisor atuou. Por exemplo no Node.js, como mostra a figura 7, é possível observar que um pequeno grupo é responsável pela maior parte das revisões. Este cenário se acentua ainda mais quando a distribuição leva em consideração o peso de cada uma das arestas, como retrata a figura 8. Estas redes são frequentemente classificadas como aleatórias, livres de escala, modulares, entre outras, de acordo com a distribuição do grau [67]. A distribuição é próxima da lei de potência, o que caracterizaria esta rede como livre de escala.

Essa tendência acompanha todos os projetos analisados. Apenas 60% dos usuários revisados no Symfony revisaram outro *"pull request"*, enquanto apenas 4.5% deles interagiram com mais de 10 outros indivíduos. Os 95% que menos interagiram são responsáveis por apenas 6% das interações entre eles. É possível observar essa tendência na figura 9,

[11] https://github.com/symfony/symfony/
[12] https://symfony.com/doc/master/contributing/code/security.html
[13] https://neo4j.com/

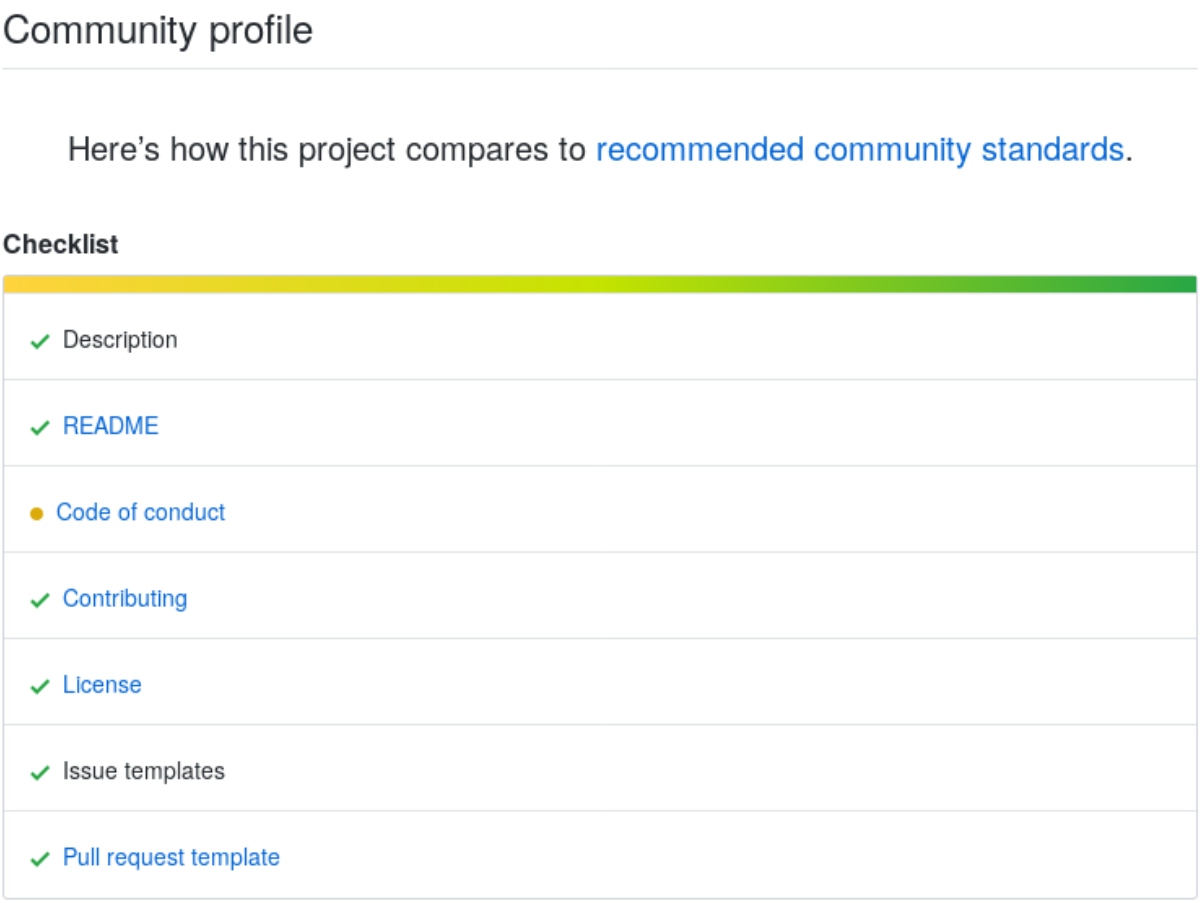

Figura 6 – Documentação para dar suporte à contribuição no Tensorflow

representação da rede do projeto Tensorflow. O tamanho dos nós é dado pelo seu grau de saída.

As distribuições encontradas mostram que poucos indivíduos são responsáveis pela maior parte das interações em revisões dos projetos selecionados. Praticamente toda interação ocorre de alguma forma associada aos principais nós da rede. Por exemplo, no Symfony e no Node.js, existem apenas quatro componentes conexas em toda o grafo. Esta estrutura converge com reportado por trabalhos anteriores [68]. Estes especialistas que conduzem a maior parte do processo são podem ser responsáveis pelo projeto como um todo, módulos específicos [69], ou tecnologias/atribuições mais específicas [70]. A identificação automatizada da semântica que fundamenta tais distribuições pode levar aos melhores revisores em situações específicas, aumentando a colaboração na revisão. É com esse objetivo que foi aplicada uma abordagem de clusterização nos grafos instanciados, que será detalhada na próxima seção e utilizada como base para alguns dos métodos de recomendação propostos.

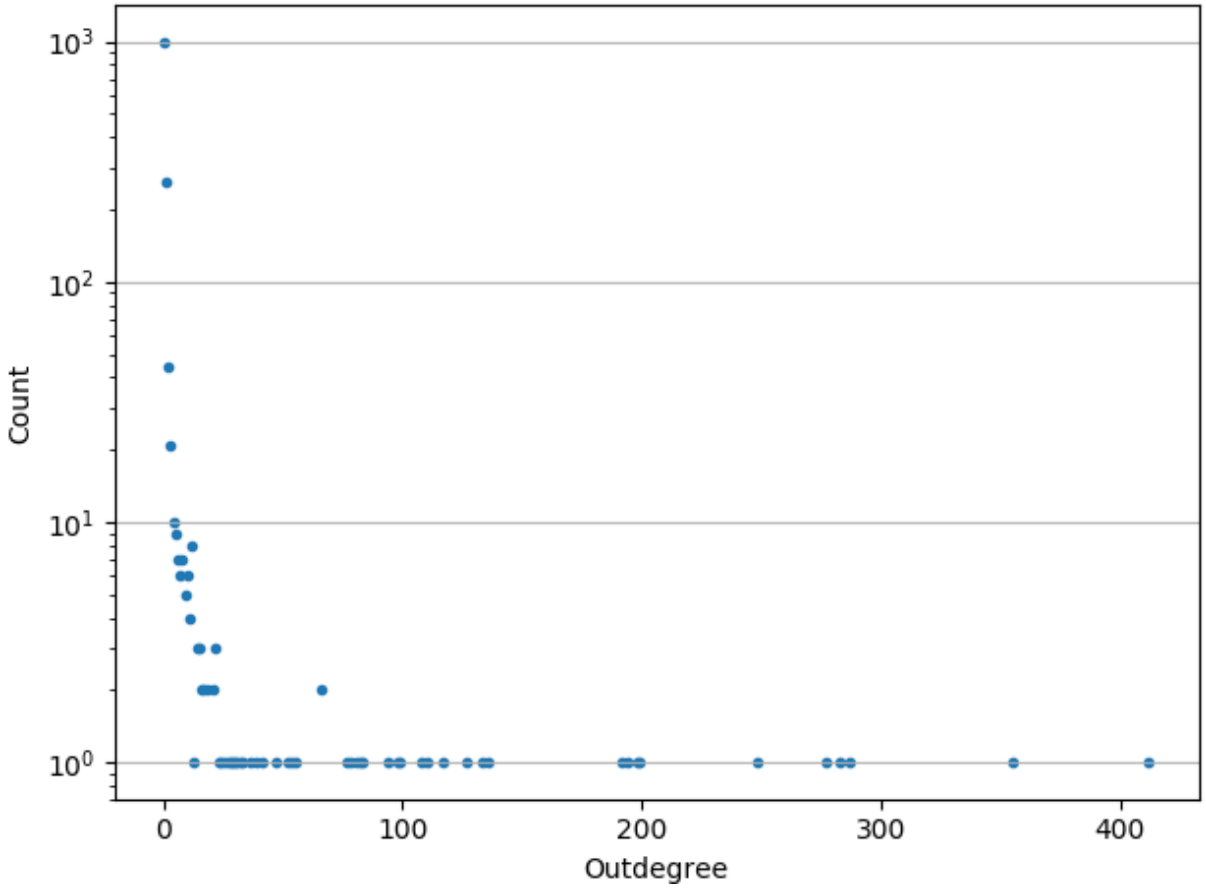

Figura 7 – Distribuição do grau de saída do Node.js

4.4 Clusterização

Uma das formas de entender melhor a estrutura de uma rede social é através de métodos de clusterização. Neste caso a proximidade dos indivíduos é calculada pelas suas interações, com um método de agrupamento [71]. No perfil de projetos-alvo deste estudo é importante que o método escolhido leve em considerações duas particularidades pertinentes:

1. É preciso que os indivíduos possam se enquadrar em mais de um grupo;
2. o número de grupos (ou *clusters*) deve ser inferido automaticamente.

Estas exigências se dão devido a natureza da organização e forma de trabalho dos projetos *open source*. Especialmente os experts acabam por participar de diversas frentes de trabalho, devido à mão de obra reduzida. Além disso, a quantidade de grupos pode variar drásticamente de um projeto para outro, inclusve dentro do mesmo projeto mas em momentos diferentes do ciclo de vida de desenvolvimento. A próxima seção detalha o algorimo escolhido e como ele trata as questões levantadas.

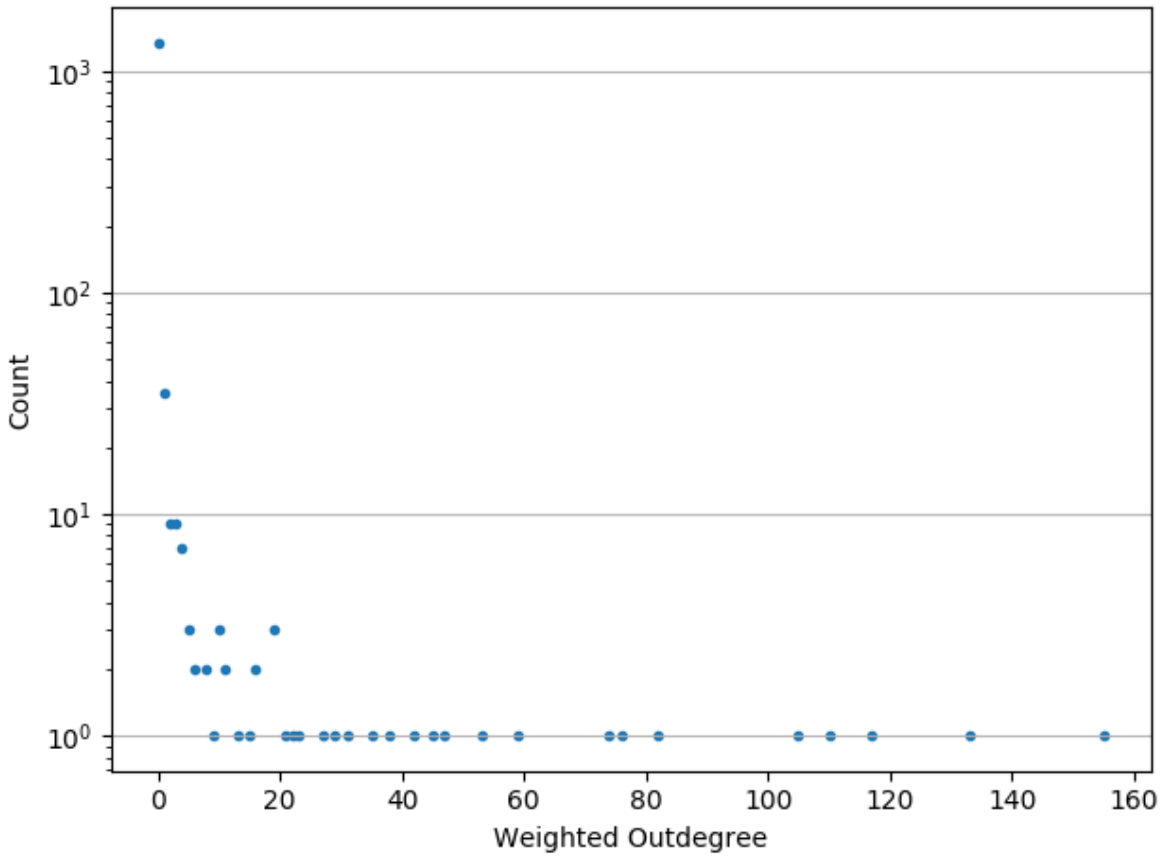

Figura 8 – Distribuição do grau de saída ponderado do Node.js

4.4.1 NetSCAN

NetSCAN [72] é um algoritmo de clusterização baseado em densidade, que estende o conhecido DBSCAN [73]. As principais diferenças são que o NetSCAN considera o direcionamento das arestas e a possibilidade de um indivíduo participar de mais de um grupo. Como é esperado encontrar grupos tênues, informais e com divisões de responsabilidades sutis, NetSCAN atende os objetivos. Os indíviduos *core* de cada cluster são identificados, podendo estes também estarem sobrepostos em outros silos.

NetSCAN espera dois parâmetros, *eps* e *minPts*. O primeiro indica o peso mínimo de uma aresta para ser considerada no processo de agrupamento, enquanto a segunda é o *threshold* que indica a pontuação para um indivíduo ser considerado como *core*. O primeiro permite evitar que nós de baixa relevância influenciem na formação dos grupos, enquanto o segundo permite que apenas indivíduos de alta participação sejam considerados como peças centrais dos silos.

Para proporcionar a melhor escolha de parâmetros, foram testadas diferentes combinações em busca de otimizar a silhueta dos grupos [74]. Essta métrica usa a distância entre os nós contidos em um *cluster* para comparar a similaridade entre eles. A ideia é que a proximidade deles seja maior que em relação a membros externos dos grupos. O

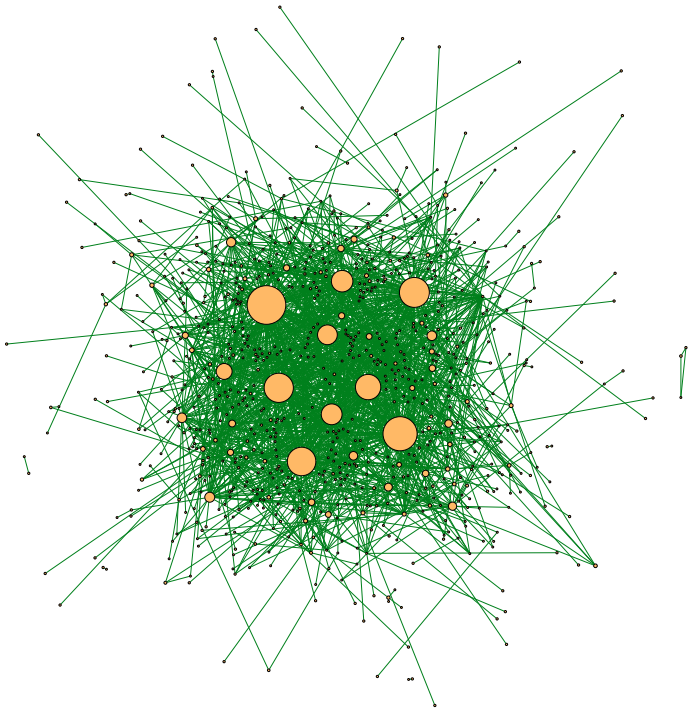

Figura 9 – Representação gráfica da rede do Tensorflow

índice varia entre -1 e 1 onde valores mais altos indicam grupos mais coesos. [75]. Na figura 10 é possível ver a distribuições de silhuetas na clusterização otimizada do Node.js. O número de *clusters* é inferido automaticamente pelo NetSCAN tendo como referência as características da rede.

A processo de otimização foi conduzido para todas os projetos, como apresenta a tabela 3. Estes são os valores utilizados todas as análises descritas posteriormente neste trabalho.

Projeto	***eps***	***minPts***
Node.js	0.45	10
Kubernetes	0.45	12
Symfony	0.38	14
Tensorflow	0.3	10

Tabela 3 – Valores otimizados pela silhueta de cada projeto

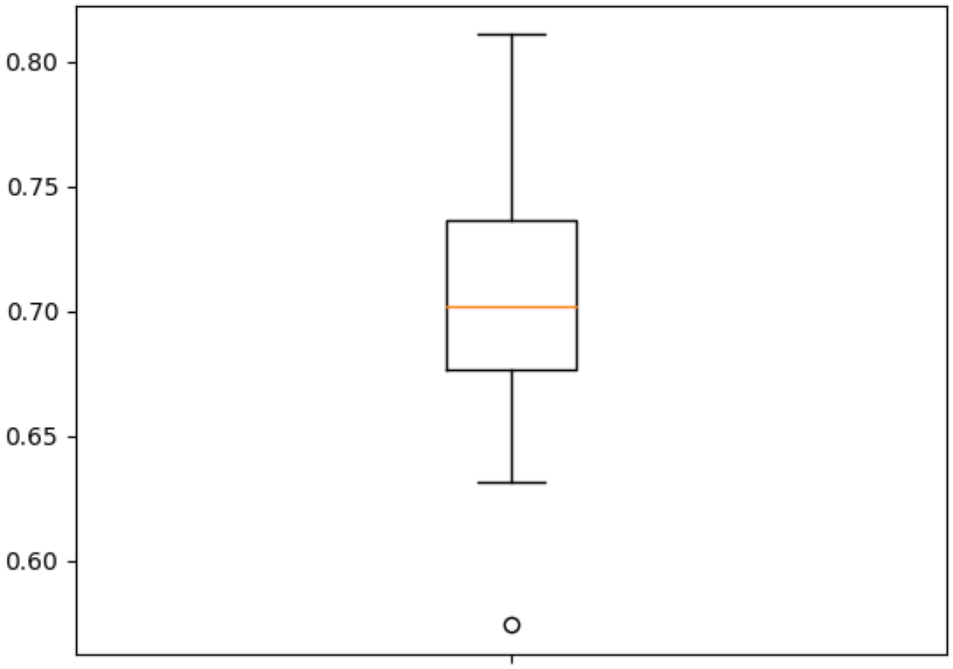

Figura 10 – Silhuetas do Node.js

NetSCAN é encapsulado em um plugin para o Neo4j[14], e pode ser executado diretamente através do Cypher com parâmetros dinamicamente definidos. Este processo é detalhado na seção seguinte.

4.4.2 Execução

A execução dos métodos de clusterização é contida, junto com outras funcionalidades, na ferramenta cuja arquitetura é detalhada na seção 5. Os dados são automaticamente carregados para uma instância Neo4j e o algoritmo aje com os parâmetros debatidos nas seções anteriores. A figura 11 ilustra um dos *clusters* encontrados.

O os nós em azul são os participantes do *cluster*, todos tiveram uma contribuição revisada pelo indivíduo *core* representado em vermelho. A entidade caracterizada em verde é o *cluster* em si, que pode ser utilizado para identificar os seus participantes através da relação do tipo *CONTAINS*.

Idealmente, os grupos encontrados devem corresponder à grupos reais de trabalho dentro de um projeto. Ou seja, deve existir uma corroboração semântica do que foi encontrado: equipes que frequentemente trabalham em conjunto, grupos focais em tecnologias ou áreas específicas, times que se destacam em módulos ou funcionalidades, entre outras divisões. A próxima seção propõe métodos objetivos para avaliar tais caracetísticas que podem indicar maior efetividade da clusterização como base para algoritmos de detcção de especialistas e recomendação para revisão.

[14] https://github.com/vitorhorta/netscan-neo4j

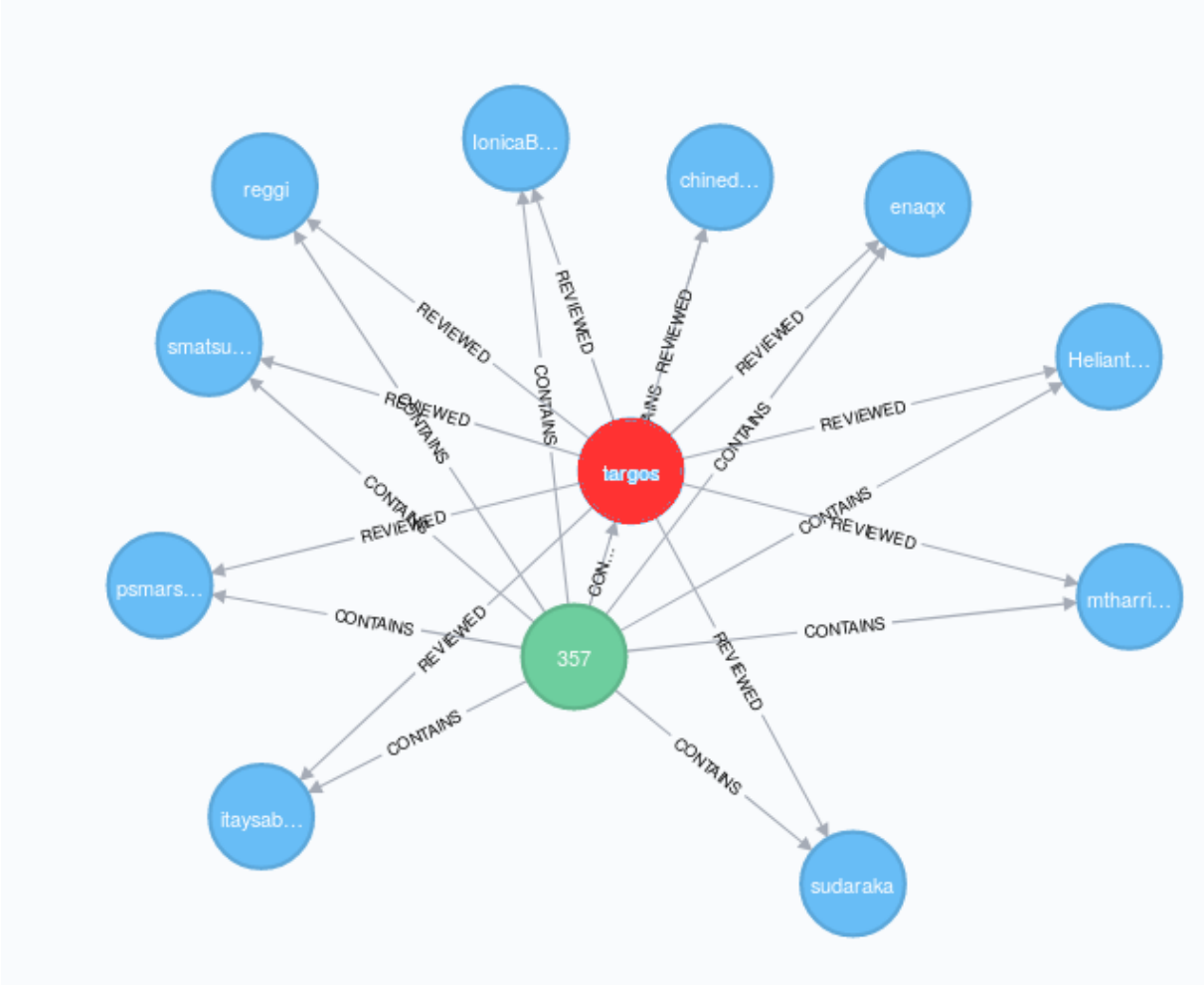

Figura 11 – Representação de um dos *clusters* do Kubernetes

4.4.3 Avaliação da clusterização

Duas formas de avaliação semântica foram aplicadas neste trabalho. A primeira diz respeito à pertinência dos *cores* encontrados enquanto desenvolvedores influentes e responsáveis pelo projeto. A segunda verifica a diferença entre as atividades e os principais focos dos grupos destacados.

Na primeira abordagem, podemos utilizar duas informações dos repositórios como base: as reações positivas e negativas que os comentários dos *cores* geram e se eles são oficialmente listados como responsáveis/membros do projeto, com poder formal sobre as decisões e revisões.

É possível verificar que os *cores* são alvos da maior parte das reações durante o processo. Esta proporção no Symfony é visualizada em escala logarítmica na figura 12. Enquanto as maiores diferenças em relação aos indivíduos comuns é observada nas reações positivas como "+1" (conhecida também como *"thumbs up"*), a diferença é pequena para expressão de sentimentos negativos. Esse é um comportamento que se repete em todos os projetos. No node.js, 81% dos *cores* indicados estão presentes no conjunto que recebeu

mais reações positivas. A lista tem 27 elementos, o mesmo número de *cores* encontrados. No Symfony esse valor é de 75%, enquanto no Kubernetes chega a 78%.

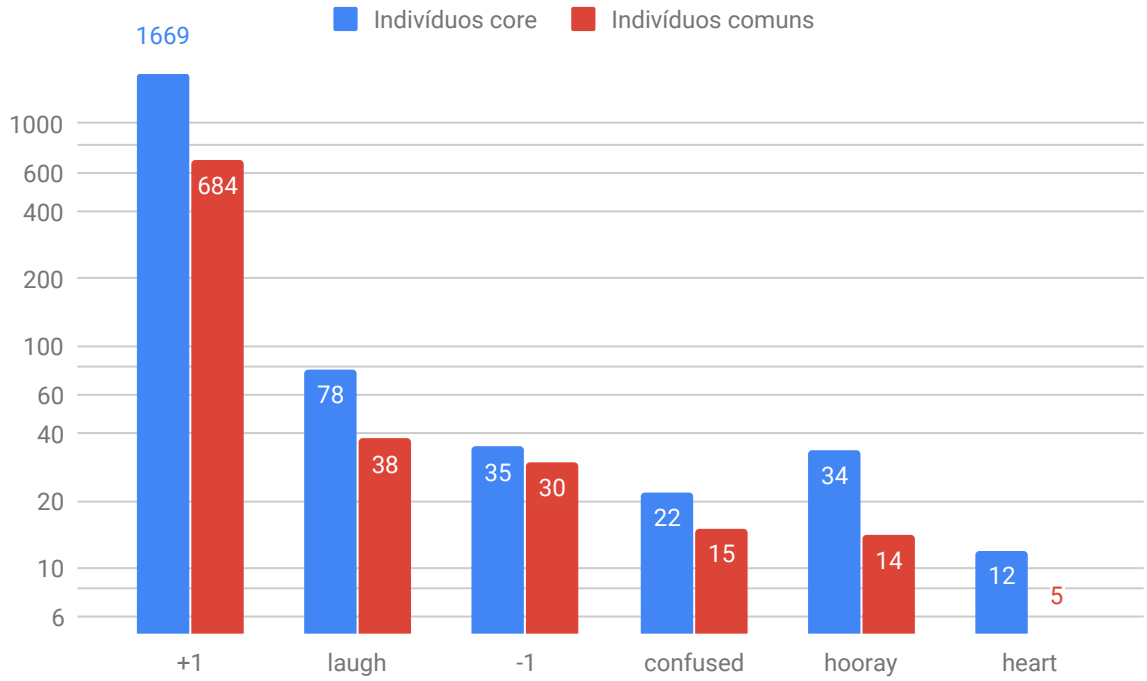

Figura 12 – Comparação das reactions recebidas por usuários *cores* e comuns no Symfony

Quanto à responsabilidade dos desenvolvedores *core* no projeto, é possível analisar se eles possuem permissões de aprovar *"pull requests"* e de outras tarefas administrativas no projeto. A tabela 4 mostra os resultados da proporção de *cores* detectados que efetivamente são dotados desse nível de permissão. É possível que participantes não possuem tal concessão hoje já tenham tido em um passado recente, mas a informação histórica não está disponível para consulta.

Projeto	**Proporção**
Node.js	78%
Kubernetes	77%
Symfony	69%
Tensorflow	69%

Tabela 4 – Proporções de *cores* detectados com direitos administrativos

Observar distinções entre os grupos do ponto de vista da natureza das tarefas executadas é uma das formas de analisar se os silos detectados são de fato pertinentes no mundo real. No caso dos projetos analisados, as categorias nas quais os *"pull requests"* foram enquadrados formam uma indicação para tal avaliação. No GitHub essas categorias são dadas como forma de *Tags* ou *Labels*, que são associadas aos *"pull requests"* no momento de sua criação, como exemplifica a figura 13.

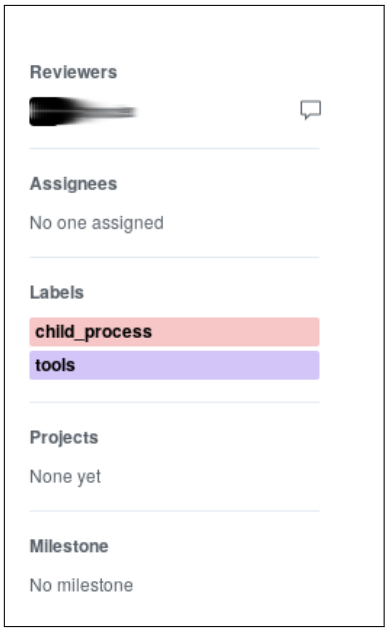

Figura 13 – *Labels* associadas a um *"pull request"* no Node.js

Para avaliar tal distinção, foram comparados quais *clusters* trabalham com quais *Labels*. Foram selecionadas as 20 *Labels* mais utilizadas de cada projeto e quais clusters trabalham com tais categorias entre suas 20 mais utilizadas. O resultado pode ser representado como u grafo na figura 14. Em todos os projetos analisados, foi possível observar a tendência de determinados tópicos de trabalho serem conduzidos principalmente por um conjunto reduzido de grupos.

No grafo da figura 14, os grupos são representados na cor verde e as *Labels* em rosa. O tamanho da representação de cada *Label* se dá pelo seu grau de entrada. Assim, as categorias maiores estão associadas à mais grupos. É o caso das categorias que representam aspectos de documentação, processo de trabalho e testes, como o caso de *"kind/feature"* e *"kind/design"*, que são divisões mais genéricas. Já as tarefas de àreas específicas, como *"area/kubectl"* e *"area/api"* são conduzidas por menos grupos, mostrando uma especialização das atribuições.

As avaliações conduzidas apontam que os grupos e *cores* encontrados podem ser reflexos das divisões de trabalho, influência e responsabilidade dentro dos projetos selecionados. Tal asserção fundamenta a utilização da rede modelada e da técnica de clusterização escolhida nos métodos de recomendação de revisores escolhidas, uma vez que a capacidade técnica e experiência dos especialistas pode aumentar a colaboração durante o processo [12, 76].

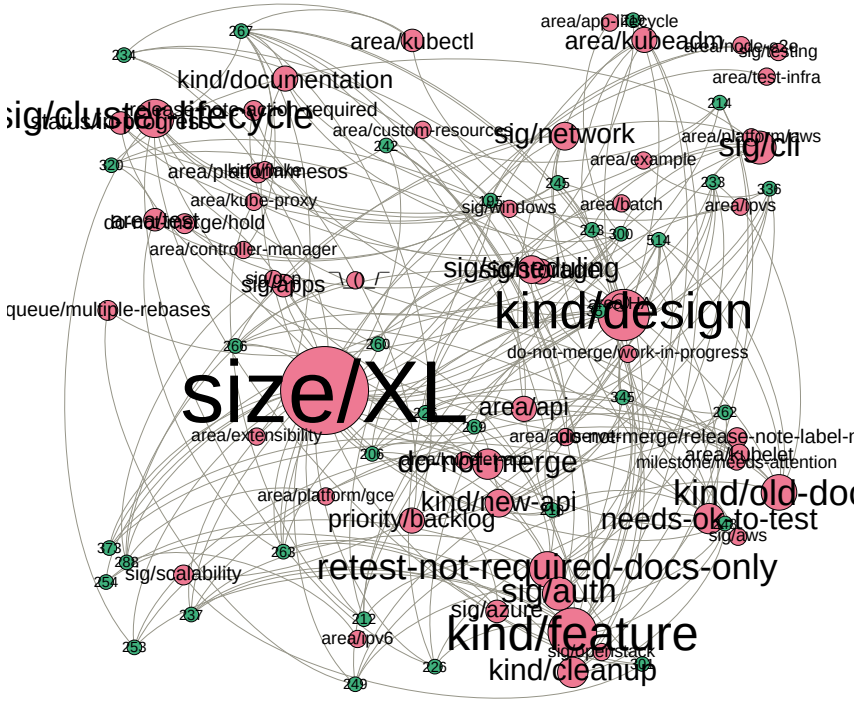

Figura 14 – *Clusters* associados a cada *"Label"* no Kubernetes

4.5 Métodos de recomendação

Os métodos propostos utilizam características observadas na rede modelada e nos *clusters* detectados para recomendar revisores que possam potencializar a colaboração durante o processo de revisão de código, de acordo com os objetivos definidos deste trabalho. A proposição de três métodos distintos e complementares se fundamenta em aproveitar as diferentes particularidades dentro do contexto de pequisa para que os utilizadores das abordagens possam escolher os resultados que mais se encaixem com os objetivos da organização. Todos os métodos recebem apenas um parâmetro k, que representa quantas recomendações deverão ser executadas.

4.5.1 RandomCore

Este método emprega os *cores* encontrados durante a clusterização como referências não só dos grupos de trabalho, mas do projeto como um todo. Esse raciocínio se aplica especialmente a tarefas que não dedicam conhecimento em tecnologias muito específicas, mas sim àquelas aplicadas ao projeto como um todo e atividades ligadas aos processos de desenvolvimento, como documentação e testes. A utilização de *cores* aleatórios, sem especificação de ordem ou prioridade dentro do conjunto de tamanho k se dá para evitar que os principais desenvolvedores sejam indicados muitas vezes e não tenham o tempo para se dedicar à revisão. Fatores como grau de saída e grau de saída ponderado poderiam ser utilizados para tal ordenação, bem como a distância entre os nós. O algoritmo 1 demonstra o comportamento deste método.

Algoritmo 1: Recomendação de revisores através do método RandomCore

```
   Entrada: int k
   Saída: set_k de revisores recomendados
1  início
2  |  set cores = get_cores();
3  |  se k > size(cores) então
4  |  |  k = size(cores);
5  |  fim se
6  |  return random_sample(cores, k);
7  fim
```

A única restrição do método é que k não deve ser maior que o número total de *cores*; nesse caso o método limita o tamanho do conjunto de recomendação ao tamanho do conjunto destes indivíduos. A função $get_cores()$ varre o grafo buscando estes nós, enquanto a função $random_sample(set, k)$ retorna k elementos aleatórios e não repetidos do conjunto.

A utilização da estrutura *set* neste e em outros algoritmos indica que não existe repetições dentro da coleção. Em caso da tentativa de inserir itens duplicados, a estrutura de dados *set* mantém seu conteúdo inalterado.

4.5.2 CoreSameCluster

Com intuito de aumentar a relevância dos resultados do algoritmo RandomCore, o método *CoreSameCluster* busca recomendar *cores* que são referência dentro do grupo de trabalho do autor do *"pull request"* submetido. O raciocínio se baseia em duas premissas: a) Ao estarem no mesmo grupo, tais desenvolvedores já colaboram e têm mais probabilidade de continuar colaborando na atual atividade, e b) O *core* do mesmo cluster é um especialista no conjunto de tópicos que aquele grupo já trabalha, e por isso poderá ser mais eficiente durante o processo. O algoritmo 2 mostra como é feita a seleção dos recomendados neste

método. Diferentemente do método anterior, esta abordagem depende de quem está realizando o *"pull request"*.

Algoritmo 2: Recomendação de revisores através do método CoreSameCluster

Entrada: int k
Entrada: string *login*
Saída: set_k de revisores recomendados

```
1  início
2      set clusters = get_clusters(k);
3      set sameCluster = ();
4      para cores in clusters faça
5          para core in cores faça
6              sameCluster.append(core)
7          fim para
8      fim para
9      se size(sameCluster) < k então
10         int j = k − size(sameCluster);
11         cores = get_cores();
12         notSameCluster = cores − sameCluster;
13         randomCores = (random_sample(notSameCluster, j));
14     senão
15         sameCluster = random_sample(sameCluster, k)
16     fim se
17     return sameCluster + randomCores;
18 fim
```

O principal objetivo deste algoritmo é retornar k indivíduos para revisão, que sejam *cores* de clusters que o autor do *"pull request"*. Como muitas vezes o universo destes nós é pequeno, para dar suporte à valores de k mais altos o algoritmo complementa a lista com cores aleatórios, assim como o método *RandomCore* do algoritmo 1. Ou seja, o ganho de performance dele é em relação ao método anterior para ks menores. Na linha 12 do algoritmo é excluída a insterseção entre os sets de *cores* e os já indicados. Isso faz com que os nós aleatórios não incluam os já selecionados, evitando que o conjunto final seja menor que o esperado.

Nos casos que o autor nunca submeteu um *"pull request"* antes, ou que não tenha sido influente (ou influenciado) suficiente para ser agrupado, não existem *cores* relacionados a ele para serem indicados. Nesta condição de "partida fria", o desempenho deste método é o mesmo do algoritmo 1.

4.5.3 LabelPartners

Enquanto o *RandomCore* não garante recomendações de pessoas próximas do autor e do escopo de trabalho e o *CoreSameCluster* só diferencia a abordagem para valores de k mais baixos e tem uma severa condição de partida fria, a concepção do *LabelPartners*

pretende equilibrar as duas abordagens. Ao invés de indicar apenas os *cores*, este método indica invíduos na seguinte ordem de prioridade:

1. Aqueles com maior influência sobre o autor nas *labels* do *"pull request"*;
2. os principais influenciadores nas *labels* do *"pull request"* de maneira global no projeto.

Ou seja, se k for menor que o tamanho do conjunto dos revisores regidos pelo item 1, apenas desenvolvedores com histórico de interação com o autor e conhecimento nos tópicos necessários serão indicados. Nos casos dos revisados pela primeira vez, os indicados ao menos serão especialistas na categoria na qual o *"pull request"* se insere. A única restrição é que o autor escolha as *labels* antes de solicitar pelos revisores, o que pode ser facilmente adotado como uma política de contribuição dos projetos. O método é descrito através do algoritmo 3. A verificação de influência em determinadas tags utiliza a mesma equação 4.1 de instanciação da rede, só que retorna pesos distintos para cada uma das *labels* que compõe a relação entre dois nós.

Algoritmo 3: Recomendação de revisores através do método LabelPartners

```
Entrada: int k
Entrada: string login
Entrada: int pr_id
Saída: set_k de revisores recomendados
1  início
2      set labels = get_labels(pr_id);
3      set partners = get_partners(login, labels);
4      se size(partners) < k então
5          int j = k − size(partners);
6          set extra = ();
7          para label in labels faça
8              extra.append(get_experts(label));
9          fim para
10         extraNotPartner = extra − partners;
11         se extraNotPartner > j então
12             extra = (random_sample(extra, j));
13         fim se
14     senão
15         partners = random_sample(partners, k)
16     fim se
17     return partners + extra;
18 fim
```

Além de evitar os casos negativos dos algoritmos anteriores, o LabelPartners permite a ordenação das recomendações na saída. Ou seja, é possível que o conjunto que atende o primeiro caso (com interação direta com o autor no passado) seja ordenado por

exemplo pela média do peso destas interações. O segundo conjunto também pode ser ordenado, onde o maior especialista nas *labels* viria primeiro. Na linha 14, o valor retornado corresponde a união dos conjuntos selecionados, sempre de tamanho k. Para evitar que indivíduos já indicados o sejam novamente, na linha 10 é excluída a interção entre os grupos da análise de especialsitas, de maneira análoga ao executado no algoritmo 2.

4.5.4 Resumo dos métodos

Os métodos foram propostos de acordo com recomendações de trabalhos pregressos em relação ao uso de dados de colaboração e análises de redes sociais (SNA) [59, 54] «citar revisão sistemática». Tais diretrizes condizem com os objetivos do trabalho de indicar revisores que aumentem a colaboração no contexto alvo. As abordagens foram concebidas não com a meta de se tornarem concorrentes, mas sim de se complementarem para cobrir particularidades dos projetos, mesmo que inseridos no mesmo contexto. Para avaliação dos métodos, foi desenvolvido um framework para aferir as métricas clássicas da área e especialmente os aspectos que tratam sobre a qualidade da revisão de código, cobertos na seção 3.4. Detalhes da implementação da ferramenta de avaliação e de recomendação são o tema do próximo capítulo.

5 SOLUÇÃO DESENVOLVIDA

A solução desenvolvida engloba a automação das tarefas de instanciação, download e avaliação dos métodos, bem como a ferramenta de recomendação baseada nos métodos propostos. Levando em consideração os modelos atuais de revisão de código exploradas na seção 2.1 são propostos os seguintes requisitos funcionais da ferramenta:

- Dada uma revisão de código, a ferramenta deve apresentar sugestões de pares técnicos para o o papel de revisor;
- o tamanho da lista deve ser escolhido pelo operador da ferramenta;
- os perfis dos indicados devem estar acessíveis (via link).

Os requisitos não funcionais estão relacionados às caracterísitcas técnicas da ferramenta que suportam os requisitos funcionais, como por exemplo disponibilidade, portabilidade e manutenibilidade. Dentre tais aspectos, pode-se destacar como mais relevantes:

- Interoperabilidade
  - A ferramenta proposta deve-se comunicar o repositório de código fonte e ser compatível diferentes IDEs .
- Manutenibilidade
  - Existem diversas ferramentas de mercado para revisão, além de ferramentas de repositório de código fonte. Por isso a ferramenta deve ser modelada para permitir evoluções futuras que permitam a agregação de novas integrações que sejam importantes em outros domínios, como por exemplo software proprietário.
- Portabilidade
  - A ferramenta deve funcionar em diferentes sistemas operacionais e de forma auto contida, independente de instalação prévia de tecnologias.
- Usabilidade
  - As principais funcionalidade de recomendação da ferramenta devem estar disponíveis em interface web para operação sem necessidade conhecimento técnico avançado.

Além dos requisitos funcionais e não funcionais caracterizados para atender os objetivos da pesquisa, é importante observar questões relativas á reprodutibilidade e avaliação do estudo. Ao conduzir as etapas de desenvolvimento e modelagem da arquitetura

da ferramenta, é possível aderir a determinadas recomendações da literatura para agregar a capacidade do experimento de ser replicado, corroborado e estentido por pesquisas futuras. A próxima seção apresenta como tais aspectos foram levados em consideração.

5.1 Aspectos de Reprodutibilidade

Cada uma das tecnologias apresentadas é encapsulada através da tecnologia de virtualização Docker[1]. esta abordagem é uma resposta às iniciativas de reprodutibilidade na ciência, buscando maior transparência, confiabiliade e possibilidade de extensão nos experimentos [77]. Os três componentes (banco de dados relacional, orientado a grafo e os scripts de extração, transformação e carga) são encapsulados em contâiners distintos, como mostra a figura 15.

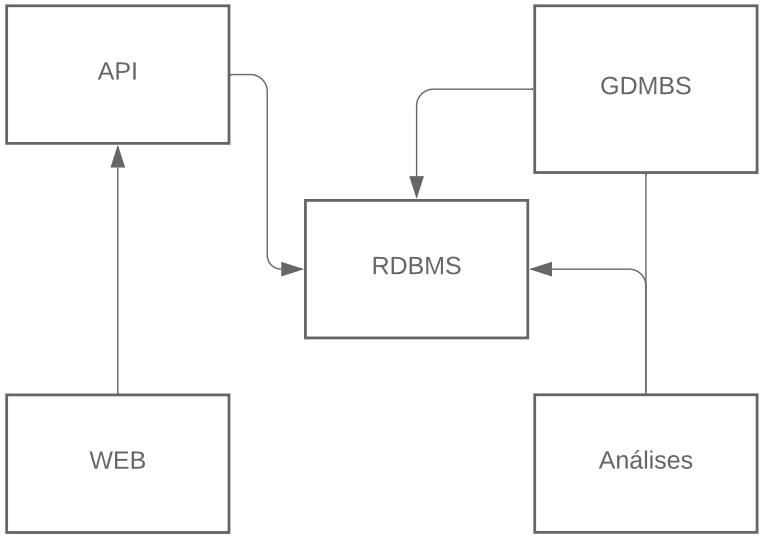

Figura 15 – Modelo de contâiners

Os componentes WEB e API se comunicam para oferecer a interface de operação da ferramenta para os usuários. A fonte de dados para todas as execuções é o Sistema Gerenciador de Banco de Dados Relacional (RDMBS), apoiado pela estrutura orientada a grafos (GDBMS). O contâiner das análises fica responsável executar os métodos de recomendação, otimização, ETL e outros que apoiam a descoberta de conhecimento na plataforma. Mais detalhes sobre cada um dos contâiners e suas funcionalidades estão descritos na seção 5.2 e 5.3. Estes componentes são orquestrados através do docker-compose[2], que configura os parâmetros e acessos necessários para o funcionamento do sistema sem que o usuário precise realizar nenhuma ação extra.

[1] https://www.docker.com/
[2] https://docs.docker.com/compose/

Com um conjunto grande de dependências, tecnologias e minúncias que estão envolvidas neste tipo de experimento, o código fonte e a descrição ainda que detalhada dos dados não é suficiente para alcançar níveis adequados de reprodutibilidade [78]. Com auxílio dos containers Docker, é possível criar instâncias executáveis dos experimentos que vão funcionar em diferentes computadores, arquiteturas e situações, sem necessidade de conhecimento técnico por parte do executor das tecnologias utilizadas [79].

De acordo com Sinha et al. [80], a maturidade da reprodutibilidade de um experimento científico computacional pode ser medido de acordo o nível dos seguintes aspectos que foram trabalhados em sua disponibilização. A figura 16 mostra o espectro de reprodutibilidade, que encara esta característica como um constante trabalho de evolução não binário, podendo uma pesquisa se tornar mais ou menos reprodutível ao se utilizar determinadas técnicas. A figura 17 mostra o detalhamento de cada aspecto que contribui para que esta característica seja evidenciada.

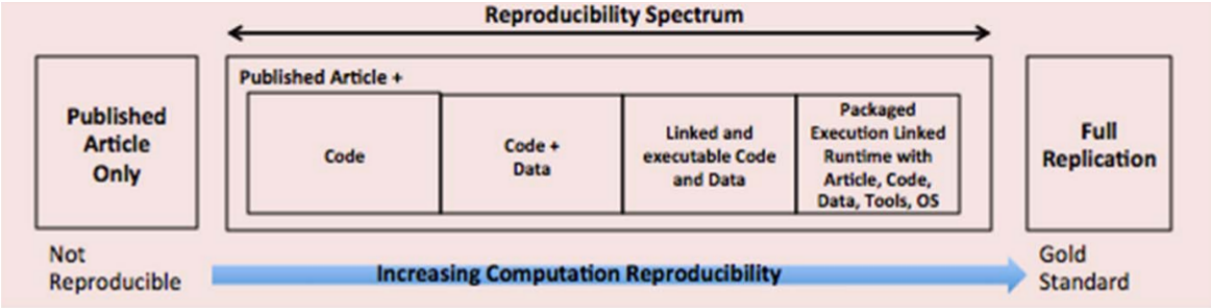

Figura 16 – Espectro de Reprodutibilidade

	Level-0	Level-1	Level-2	Level-3
Data - Primary / secondary empirical evidences	Not Provided	Descriptive Stats	Shared Data	Non-Repudiable data sharing or specimen sharing
Model and Parameters (including Methodology)	Not Provided	Reference to existing mechanisms	Described with parameters and options	Code with documentation, parameters, missing value treatment, transformations done, distributions along with results
Analytical Code Software	Not Provided	Proprietary/COTS	Open Source	Free/ replicable license enabled
Computing system (required h/w and s/w)	Not Provided	Disclosure	Virtualized	Cloud Computing Packaged Environment
Presentation Artifacts	Not Provided	Text	Literate Statistical programming	Collaborative Reproducible Writing

Figura 17 – Escala de Maturidade de Reprodutibilidade

5.1.1 Dados - Evidências Primárias/Secundárias

Esta categoria analisa a disponibilidade dos dados utilizados para futuros pesquisadores. Pode variar de simplesmente não dispobilizado até a disponibilização íntegra das informações, valendo-se de meios para uma oferta não repudiável e com garantias de

integridade e veracidade. Neste trabalho, todos os dados (em suas diferentes agregações) são disponibilizados para uso futuro. Além disso os mecanismos de extração são automatizados, permitindo buscar outros projetos ou atualizar os dados existentes. Como não há garantias de não repúdio e autenticação dos dados, o estudo se classifica como nível 2 nesta categoria.

5.1.2 Modelo e Parâmetros

Verificação dos modelos e parâmetros utilizados no experimento. São essenciais para a discussão e reprodução do experimento, além de qualquer tentativa de otimização. Varia de não disponibilizado até a disponibilização plena com documentação adequada, interface robusta que trate valores fora do domínio (como nulos) e também prática, facilitando os testes com parâmetros e dados distintos. Os modelos de dados e de execução são detalhados neste trabalho. Além disso, todas as análises são automatizadas e encapusladas através de comandos executáveis dentro de um contâiner Docker. Por conseguinte, o trabalho pode ser classificado como nível 3 nesta categoria

5.1.3 Código Fonte

Disponibilidade do código fonte utilizado nas análises. Pode variar de não disponível (proprietário) até open source com direito de extensão e modificação. Neste projeto todo o código fonte é opensource e disponibilizado sob a permissiva licença MIT [81], categorizando assim o nível 3.

5.1.4 Sistema computacional requerido

Detalhamento das informações de hardware e software necessários, como por exemplo memória, arquitetura, processador, versão e plataforma. O nível ideal é a disponibilização do experimento em ambiente virtualizado em nuvem, em arquitetura que permita tanto a reprodução "as is"quanto extensão e modificação de parâmetros e dados de entrada. Como todo o pipeline deste projeto está disponibilizado na infraestrutura Docker, a única restrição da máquina do pesquisador é ter o Docker instalado. Em adição, serviços de nuvem como a Amazon AWS fornecem infraestutura para execução de containers Docker sem necessidade de configurações extras. Informações do hardware utilizado nos experimentos estão disponibilizadas nas próximas seções. Assim, o projeto pode ser classificado como nível 3 nesta categoria.

5.1.5 Artefatos de apresentação

Todos os artefatos de apresentação, tais como artigos, dissertação e posters oriundos deste trabalho foram construídos com o LaTeX[82] e o código fonte disponibilizado. Gráficos

e tabelas foram produzidos diretamente através de scripts e/ou com ferramentas *Open Source*, como o Gephi[3]. Nesta categoria o estudo também se caracteriza como nível 3.

De acordo com a escala proposta [80] e com as informações prestadas nesta seção, concluímos que a pesquisa está no nível 2 de maturidade em reprodutibilidade, já que o autor propõe que o trabalho seja classificado de acordo com a menor nível atingido em uma das categorias.

5.2 Arquitetura

A ferramenta proposta é estruturada em diferentes componentes, de acordo com sua função e tecnologias escolhidas. A figura 18 mostra essa divisão, onde os quatro principais componentes possuem subcomponentes com tarefas mais específicas autocontidas. As subseções seguintes tratam de cada um dos componentes em particular.

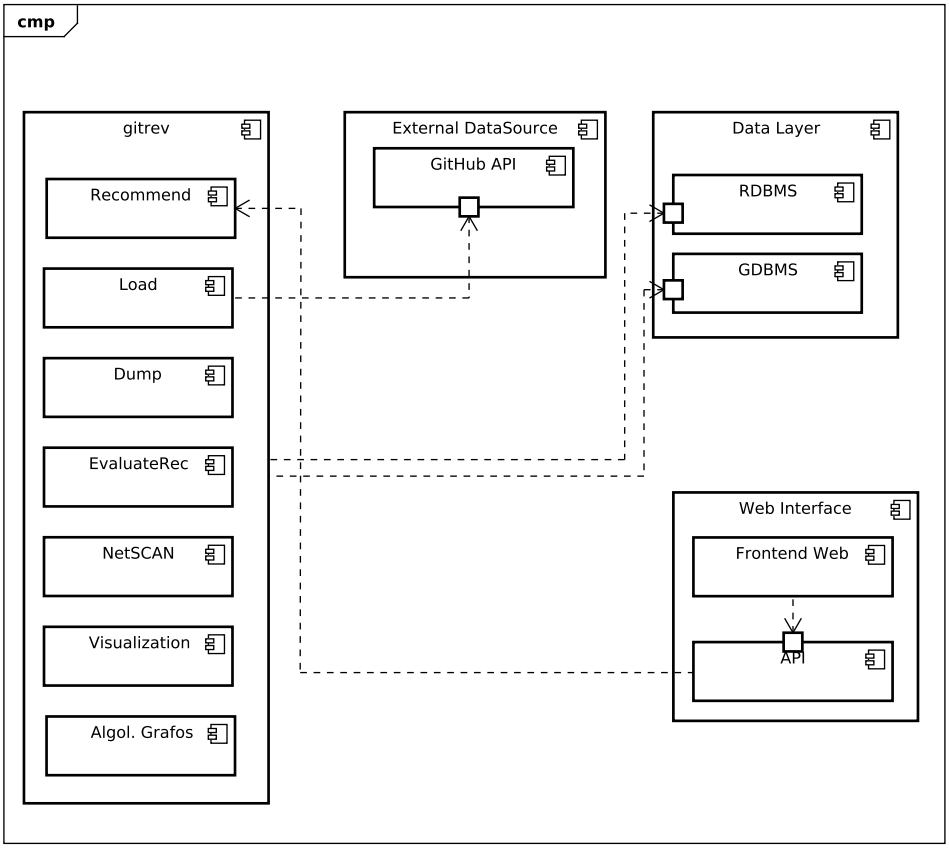

Figura 18 – Diagrama de Componentes da Ferramenta

5.2.1 External DataSource

O módulo de dados externo é incorporado pela API do GitHub, fonte dos dados brutos extraídos e armazenados no componente de persistência de dados. A versão estável

[3] https://gephi.org/

para integração foi a v3, que implementa o padrão arquitetural RESTful[83]. Existe a API em recém lançada v4[4], que utiliza da especificação GraphQL para exposição dos serviços. Esta não foi utilizada devido a documentação ainda recente e pela preferência ao padrão REST.

5.2.2 Data Layer

Neste componentes os dados normalizdos oriundos do GitHub são persistidos e tidos como base para as outras funcionalidades existentes. o Diagrama de Entidade Relacionamento (DER) na figura 19 representa as informações disponíveis. As colunas salvas são as mesmas que a API do GitHub disponibiliza e foram restringidos pensando no espaço, já que só a entidade *"pull request"* tem 37 campos. Todas as tabelas possuem um identificador (id) gerado sequencialmente, além das informações de identificação do próprio GitHub. O DBMS escolhido para este componente foi o PostgreSQL 10.5, que é *"Open Source"* e implementa as características do padrão ACID [84] de atomicidade, consistência, isolamento e durabilidade.

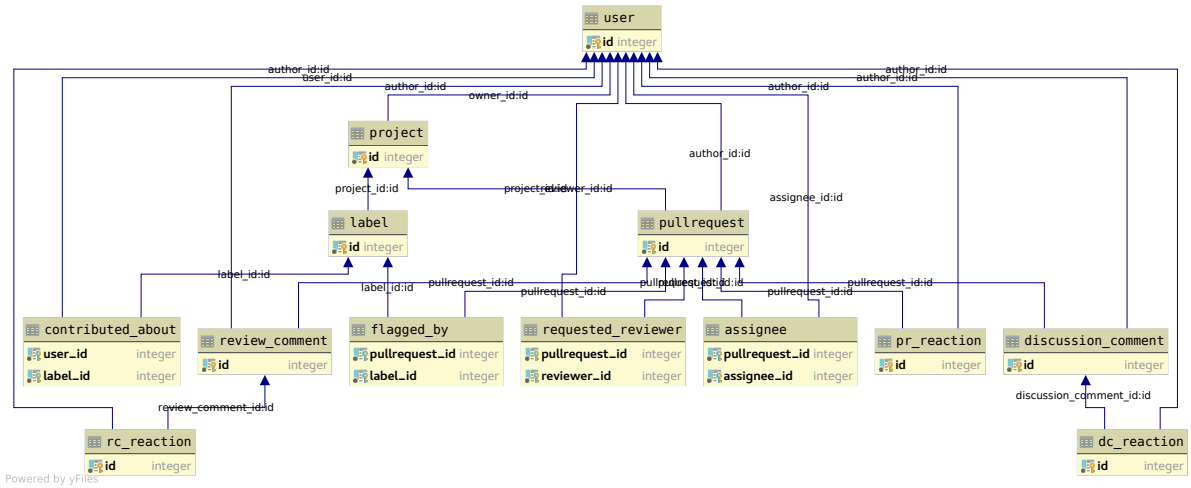

Figura 19 – Diagrama de Entidade Relacionamento (DER)

Na figura 19 é possível ver que a entidade *user* é compartilhada em diferentes projetos, e assume o papel de autor de várias outras. Isso significa que mesmo que um autor contribua em projetos diferentes será representado uma única vez, o que permite análises mais complexas entre projetos distintos. Os *"pull requests"*, outra categoria chave para as análises, possui dois tipos de comentários: aqueles feitos no formato de discussão (*discussion_comment*) e aqueles que apontam para partes específicas do código (*review_comment*). A segunda categoria que é utilizada neste trabalho. Estes comentários (e o próprio *"pull request"*) recebem as respectivas reações, utilizadas como representações de sentimentos positivos ou negativos em relação às interações. Os *"pull requests"* se

[4] https://developer.github.com/v4/

relacionam *n:m* com as *labels*, indicando as categorias nas quais foram classificados. A tabela *requested_reviewer* representa solicitações de revisores que quando aceitas viram registros na tabela *assignee*. Estes dados não foram utilizados para avaliar a recomendação porque apenas os indivíduos com permissão são referenciados como "atribuídos", o que iria enviesar os resultados das recomendações de *cores*.

O segundo componente da camada de persistência é a rede de desenvolvedores instânciada em um Sistema Gerenciador de Banco de Dados Orientado a Grafos (GDBMS), que disponibiliza os dados relevantes do modelo relacional na visualização de grafos, permitindo as análises e métodos de clusterização serem executados com maior facilidade. A tecnologia escolhida neste caso foi o Neo4j[5] que além de possuir o plugin do NetSCAN permite a integração com o Gephi para produzir filtros, análises e visualizações mais elaboradas. Apenas os dados básicos dos indivíduos e suas relações foram instanciadas nele, enquanto consultas mais elaboradas precisam contar com o suporte do PostgreSQL.

5.2.3 Gitrev

Gitrev é o nome da plataforma e do principal componente do sistema, sendo responsável pela interação com os dados durante todo o processo. Todo o componente foi desenvolvido em Python, aproveitando da extensibilidade, ecossitema e flexibilidade da linuagem. A interface com o RDBMS é suportada pela biblioteca SQLAlchemy[6], sendo o pilar da camada de acesso aos dados (*Data access objetcs*, ou DAO). Para a interação com o GDBMS a biblioteca Py2neo[7] provê interface semelhante ao seu par relacional. O módulo *Dump* inicia o processo de extração ao acessar a API do GitHub, com auxílio da biblioteca requests[8]. O algortimo de extração dos dados foi projetado com uma série de cuidados para garantir a integridade e disponibilidade das informações, diante das nuances do repositório e do domínio de dados. Os principais aspectos observados são:

5.2.3.1 Inconsistência nos dados

Para evitar dados inconsistentes, toda a extração é feita dentro de uma transaction relacional do RDBMS, garantindo que todas as ações serão executadas (ou canceladas) de forma atômica. As restrições de chave primária também são a primeira linha de defesa para evitar que informações inconsistentes sejam armazenadas (como por exemplo um comentário relacionado a um *"pull request"* que não existe).

[5] https://neo4j.com/
[6] https://www.sqlalchemy.org/
[7] https://py2neo.org/v4/
[8] https://2.python-requests.org/en/master/

5.2.3.2 Tratamento das exceções da API

O repositório de dados pode se comportar de maneira anormal durante a extração dos projetos, por diversos motivos: indisponibilidade dos serviços, lentidão do cliente, problemas de conexão entre outros. Assim é necessário tratar erros (como 400, 500 e 502) e evitar que o dump seja interrompido. Para isso, as respostas são tratadas, e detectados erros temporários como estes, o request é agendado para se repetir em uma janela de tempo definida.

5.2.3.3 Ratelimit da API

A API do GitHub limita o número de requests a 5000/hora. Por isso, é necessário que o cliente controle o número de requests para que não seja bloqueado. Assim, o componente de extração limita o seu número de requests, entrando em espera quando o número se aproxima ao limite imposto.

5.2.3.4 Unicidade de usuários

Usuários são entidades espalhadas entre os diferentes projetos. Para possibilitar análises globais e estatísticas realísticas de tamanhos de projetos, é necessário que não haja duplicação entre os usuários. O componente de extração também garante que não hajam usuários duplicados.

O módulo *Load* é responsável por gerar a rede especificada na seção 4.2, a partir dos dados encontrados no RDBMS. Além do peso relatado na defingão do grafo, os pesos segregados por *label* necessários para o algoritmo 3 de recomendação já são introduzidos. Uma propriedade única é que o peso global de determinada relação é a média ponderada dos pesos por *label*.

O componente NetSCAN reúne as funcionalidades de criação, avaliação e otimização dos *clusters*. Nele é possível encontrar a combinação de parâmetros que geram a melhor silhueta, como explicado na seção 4.4.1. Os utilitários que avaliam a qualidade dos grupos detectados e sua relevância semântica em relação ao processo produtivo do projeto também são encapsuladas neste módulo. Muitas vezes tais análises são executadas com auxílio do módulo *Algortimos de Grafos* onde consta um conjunto de métodos para avaliar as redes e os relacionamentos encontrados.

Na camada *Visualization* é possível encontrar os algortimos que se apóiam nas ferramentas matplotlib[9] e numpy[10] para gerar gráficos de distribuição de grau, *boxplot* das silhuetas entre outros.

[9] https://matplotlib.org/index.html
[10] https://www.numpy.org/

No componente *Recommendation* são efetuadas as recomendações tendo como base os métodos propostos. As informações são extraídas e do GDBMS e disponibilizadas para o componente de interface com o usuário. A avaliação das recomendações é feita no módulo *EvaluateRec*, que executa recomendações para revisões dentro de um período escolhido e as analisa de acordo com os parâmetros definidos. Este é também o responsável por avaliar a significância estatística das diferenças entre as abordagens, com apoio da biblioteca ScyPy[11].

5.2.4 Web Interface

A interface web é destinada à operação cotidiana da ferramenta em busca dos revisores adequados. A aplicação se estrutura em dois componentes. O *backend* é provido pelo Flask[12], microframework com capacidade de dar suporte à interface RESTful modelada, e é acessado através de um proxy provido pelo servidor web Nginx[13]. O backend acessa o componente *Recommend* e os *"pull requests"* persistidos no RDBMS para responder às solicitações do *frontend*. Este componente foi desenvolvido com o apoio do framework reativo Vue.js[14] e é representado na figura 20. O componente de seleção dos *"pull requests"* é dinâmico e realiza a busca de acordo com a entrada do usuário. Os links nos usuários levam aos seus perfis no GitHub.

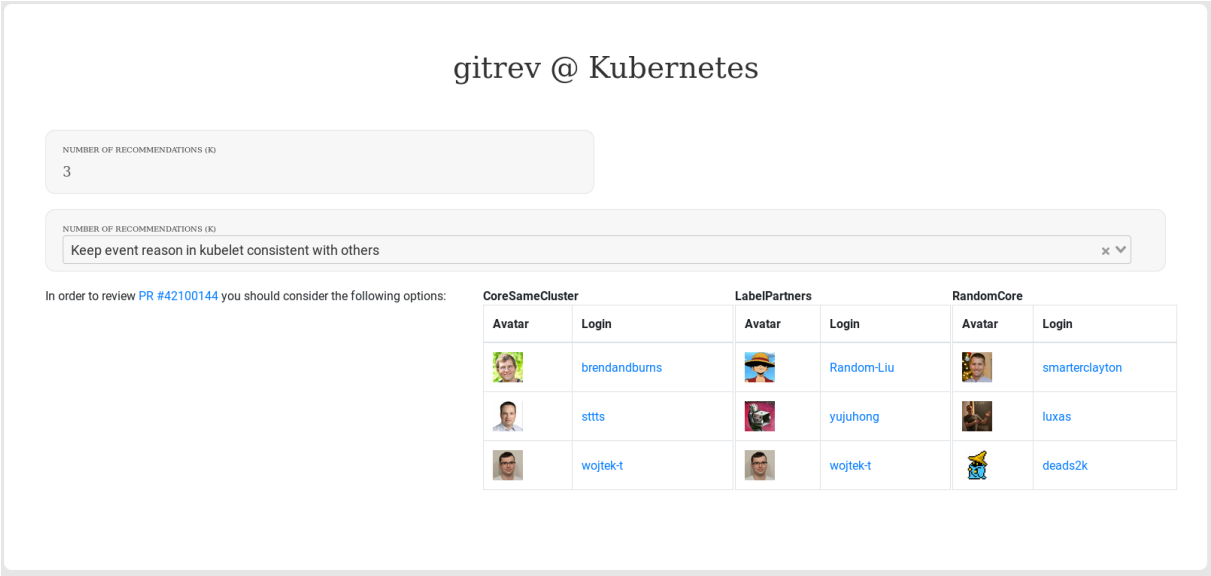

Figura 20 – Funcionalidade de recomendação no projeto Kubernetes

11 https://www.scipy.org/
12 http://flask.pocoo.org/
13 https://www.nginx.com/
14 https://vuejs.org/

5.3 Funcionalidades

A utilização da ferramenta pode ser dividida em duas etapas principais: na preparação e na recomendação. A preparação pode ser executada de tempos em tempos, para atualizar a base oriunda do GitHub e testar novos parâmetros. É uma tarefa que pode ser agendada no processo de trabalho da organização e por isso é disparada via linha de comando (CLI). O diagrama de atividades na figura 21 descreve o processo.

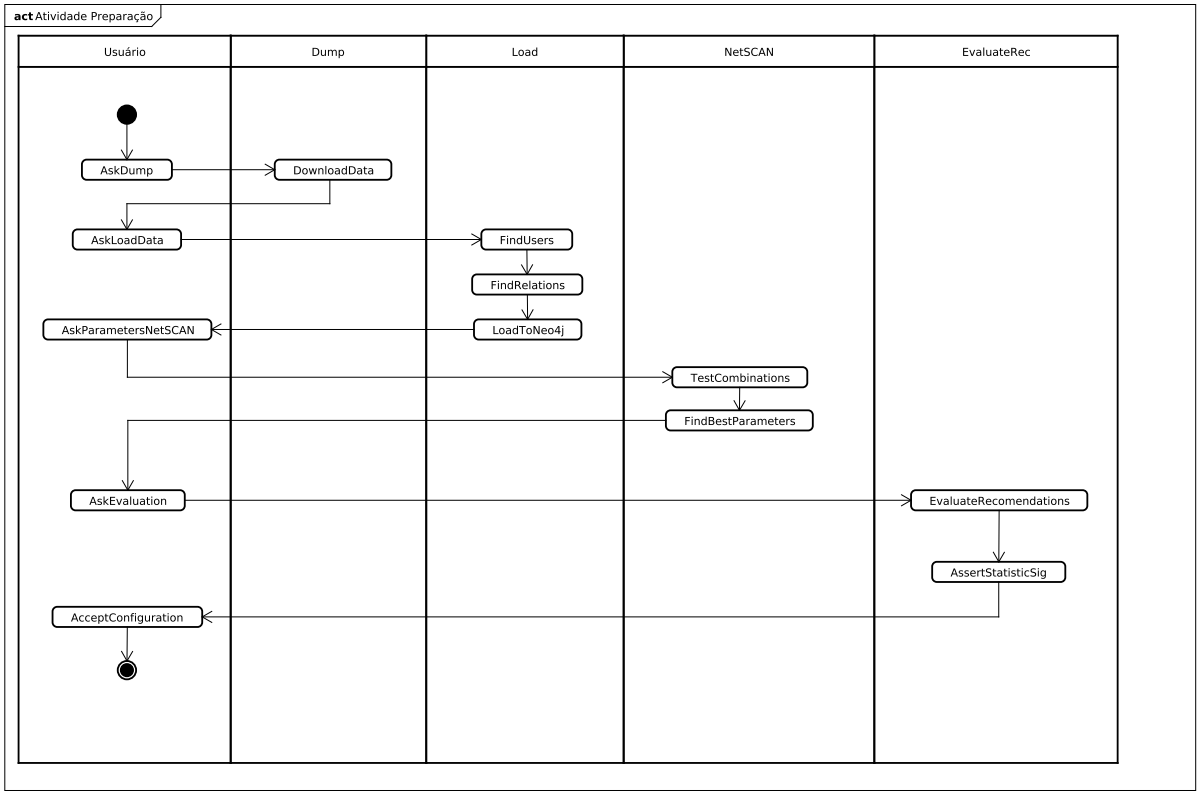

Figura 21 – Diagrama Atividades - Preparação

O responsável deve solicitar o *download* de um ou mais projetos, que serão extraídos via API. O RDBMS foi projetado para persistir dados de quantos projetos forem solicitados. Contudo cada instância do Neo4j representa, nesta versão, dados de apenas um projeto. O usuário então pode solicitar a *clusterização* da rede, passo necessário para os algortimos de recomendação 1 e 2. Os parâmetros utilizados podem ser obtidos através da otimização, ou salvos para utilização recorrente, caso seja assim preterido. Avaliação dos métodos é um passo opcional que pode ajudar o operador a escolher o melhor método para cada situação. Ao finalizar este fluxo, os usuários estão livres para solicitar as recomendações na plataforma web, o que também é possível através da interface de linha de comando `gitrev PULL_REQUEST_ID [k]`. O valor padrão de k, se não informado, é 1, indicando que apenas um revisor será recomendado.

O fluxo de operação cotidiano ferramenta foi projetado visando compatibilidade com o processo de revisão, especialmente no modelo *pull based* [1] introduzido na seção 2.1.3.

A figura 5.3 representa o diagrama de sequência da comunicação entre o autor do *"pull request"* (responsável pela escolha dos revisores), o GitHub e o utilitário de recomendação.

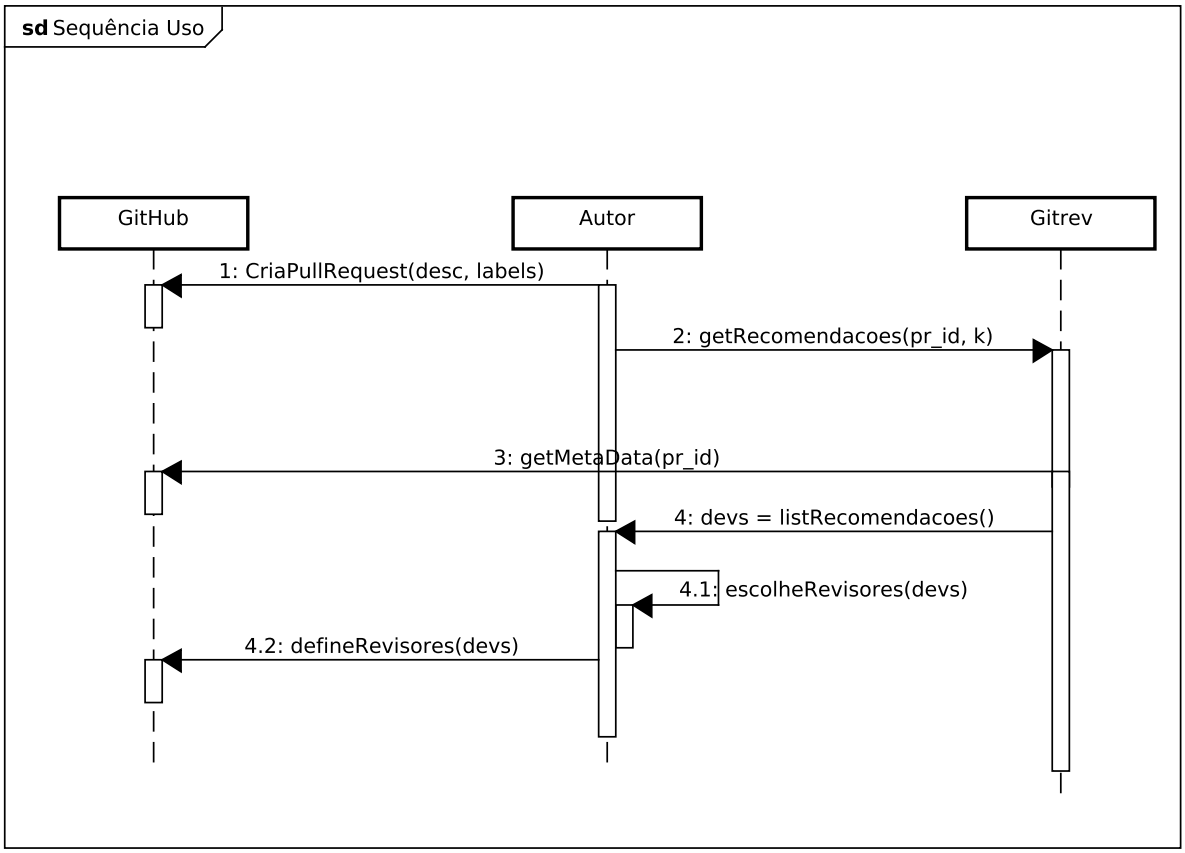

Figura 22 – Diagrama de Sequência da ferramenta - utilização

O primeiro passo consiste no autor criar a revisão com os dados relevantes, como por exemplo descrição, link com issues solucionadas, dúvidas e possíveis discussões que estejam relacionadas ao código submetido. Ao executar o utilitário, este irá acessar os metadados do Github para encontrar outras revisões de contexto semelhante e extrair informações de revisores que estiveram envolvidos nelas. A partir daí a recomendação é feita e disponibilizada para o revisor, que poderá convidar um subconjunto da lista apresentada para o papel de revisor.

Com a ferramenta dispondo do sistema de recomendação e de avaliação, é possível conduzir um processo para averiguar a pertinência dos resultados de acordo com os objetivos deste trabalho. O processo de avaliação é descrito no próximo capítulo.

6 AVALIAÇÃO DA SOLUÇÃO

A avaliação da solução proposta passa por três etapas distintas: a extração dos dados, a preparação das estruturas e a apreciação das métricas escolhidas e a significância estatísitica dos resultados. A figura 6 mostra o procedimento.

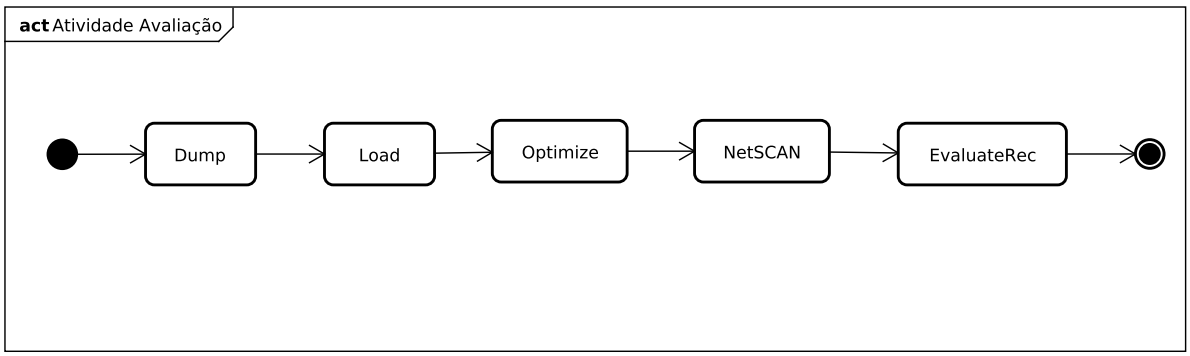

Figura 23 – Processo de avaliação

As atividades estão todas contidas na ferramenta, com uma ordem de execução similar ao uso exemplificado na seção 5.3. A partir dos dados extraídos relativos aos 4 projetos selecionados na subseção 4.3.1, as respectivas redes são construídas. É possível escolher o intervalo de tempo para limitar quais dados serão utilizados no processo. Tal mecanismo serve para possibilitar que os dados utilizados na avaliação sejam diferentes dos utilizados para criar a rede, diminuindo o risco de viés nas análises, sem ter que acessar novamente a API do GitHub. A otimização dos parâmetros pode ser feita uma única vez e aproveitada para quantas execuções da avaliação por projeto forem conduzidas. A última etapa executa a recomendação para todos os *"pull requests"* dentro de um intervalo de tempo, avalia as métricas selecionadas e ainda realiza os testes estatísticos pertinentes. Esta etapa é foco das duas próximas seções, que descorrem das métricas utilizadas para avaliação e do processo de apreciação da significância estatísitca dos resultados obtidos.

6.1 Métricas de avaliação

Duas categorias de métricas foram escolhidas: aquelas comuns em avaliação de sistemas de recomendação e especialmente no âmbito de recomendação de revisores e aquelas que envolvem a qualidade da participação dos indivíduos indicados no processo de revisão. O desempenho no primeiro grupo não reflete necessariamente os objetivos do trabalho, mas foram incluídas por serem comuns neste ramo de pesquisa «citar revisão». São a porcentagem de precisão e acerto, como explicado na seção 3.3, e doravante referenciadas como "métricas de proximidade". Já as métricas do segundo grupo são baseadas nos estudos de Bosu et al. [60] e Rahman et al. [61] detalhadas na seção 3.4 e doravante denominadas "métricas de eficiência". São métricas de avaliação dos comentários e interações dos

indivíduos durante o processo de revisão, e por isso atendem melhor aos objetivos do trabalho. Elas são detalhadas nas seguintes subseções.

6.1.1 Número de Comentários

Representa o número médio de interações realizadas pelos indivíduos em determinada revisão. Pode mostrar maior atividade e comprometimento dele em relação ao processo. Não são computados comentários do próprio autor do *"pull request"*, já que esse também não é considerado para os algortimos de recomendação. É referenciado pela variável *num_comments*.

6.1.2 Número de Reações

Mede o número médio de reações (os populares *emojis*) que foram recebidos pelos indivíduos em determinada revisão. É uma representação para reputação e impacto que determinado indivíduo pode causar no processo, sendo este capaz de gerar comentários mais relevantes [60]. Não são computados comentários do próprio autor do *"pull request"*, já que esse também não é considerado para os algortimos de recomendação. É referenciado pela variável *num_reactions*.

6.1.3 Número de repostas

Os comentários podem ser diretamente respondidos, mostrando possível impacto e capacidade de gerar mudanças, características importantes de participações no processo de revisão [60]. Neste caso as respostas do autor do *"pull request"* são contabilizadas. É referenciado pela variável *num_replies*.

6.1.4 Tamanho dos comentários

O tamanho dos comentários tem relação com sua relevância [61]. Não são computados comentários do próprio autor do *"pull request"*, já que esse também não é considerado para os algortimos de recomendação. É referenciado pela variável *size_comments*.

6.1.5 Proporção de *"non stop words"*

As *"stop words"* (em tradução livre, palavras de parada) são construções comuns a diversos tipos de texto, e por isso sem relevância específica no contexto em estudo*wilbur1992*. A alta proporção de palavras que não se encaixam nesta descriação podem ser um sinal de um texto com mais substância e significativo, estando associado a relevância do comentário ao processo de revisão [61]. É referenciado pela variável *rate_non_stop_words*.

Outra diferença entre os dois conjuntos de métricas se dá na forma de comparação. As "métricas de proximidade" só podem ser apreciada entre duas formas de recomendação

distintas pois descrevem o quão parecida foi a recomendação em relação ao efetivamente escolhido pelo autor. Já as "métricas de eficiência" são autocontidas, pois podem ser confrontadas entre o comportamento dos indivíduos que foram indicados e os que não foram. Ou seja, é possível aferir se quando os recomendados interagem no *"pull request"* em questão, existe diferença em relação à qualidade das interações realizadas. Estes grupos são diferenciados pelo símbolo r para recomendados e w para não recomendados em cada métrica. Por exemplo $num\_comments_r$ referencia o número médio de comentários dos recomendados em determinada revisão.

6.2 Significância estatística

Para estabelecer juízo de valor sobre o experimento é necessário, primariamente, avaliar a relevância estatísitica sobre os dados coletados durante sua execução. A literatura acerca deste assunto indica possível subutilização de ferramentas estatísitcas adequadas na Engenharia de Software [85], o que pode acarretar em trabalhos com resultados questionáveis e sem fundamentação sólida [86].

As análises dispostas nesta seção foram conduzidas de acordo com o trabalho de Araújo et al. [87] e aplicado em Schettino et al. [27], com objetivo de observar se a implantação da prática de *code review* no processo de desenvolvimento da organização alvo surtiu efeitos estatísticamente significativos nas métricas estruturais dos projetos alvo. O nível de significância (valor p ou *p-value*) escolhido foi 0,05, muito utilizado na Engenharia de Software e outras ciências naturais.

Levando em consideração o grande número de amostras analisadas nestra trabalho, não são demonstrados os passos de cada uma das análises, para manter o tamanho do trabalho dentro de um limite razoável. Nesta seção são utilizadas amostras de terceiros para ilustrar os tratamentos estatísticos utilizados, enquanto na seção 6.4 os resultados são apresentados em forma de tabela.

O primeiro passo é verificar a normalidade dos dados. Como pode-se observar que todas métricas contém menos de 30 amostras, aplica-se o teste de *Shapiro-Wilk*. São consideradas as seguintes hipóteses:

- H0 (hipótese nula): as amostras apresentam distribuição normal;
- H1 (hipótese alternativa): as amostras não apresentam distribuição normal.

Caso o *p-value* do teste seja maior que 0,05, deve-se aceitar a hipótese nula, uma vez que não há indícios que apontem para a hipótese alternativa. A figura 24 mostra o resultado deste tipo de teste.

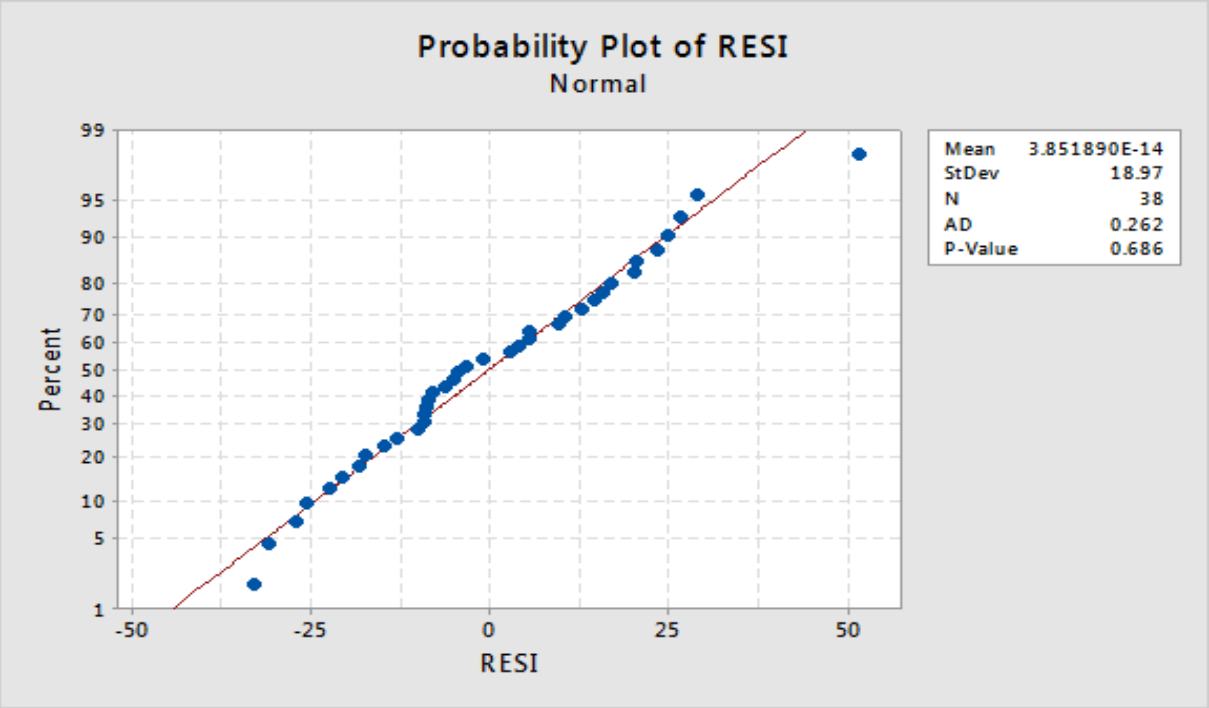

Figura 24 – Gráfico representando amostra normal segundo o teste de *Shapiro-Wilk* [2]

Da mesma forma, verifica-se se as amostras são homocedásticas. Através do teste de *Levene*, observou-se o *p-value* e foram avalaidas duas hipóteses:

- H0 (hipótese nula): as amostras homocedásticas.
- H1 (hipótese alternativa): as amostras não são homocedásticas;

Ao verificar resultado maior que a significância estabelecida de 5%, aceita-se a hipótese de que os dados são homocedásticos. É o caso da amostra representada pela figura 25

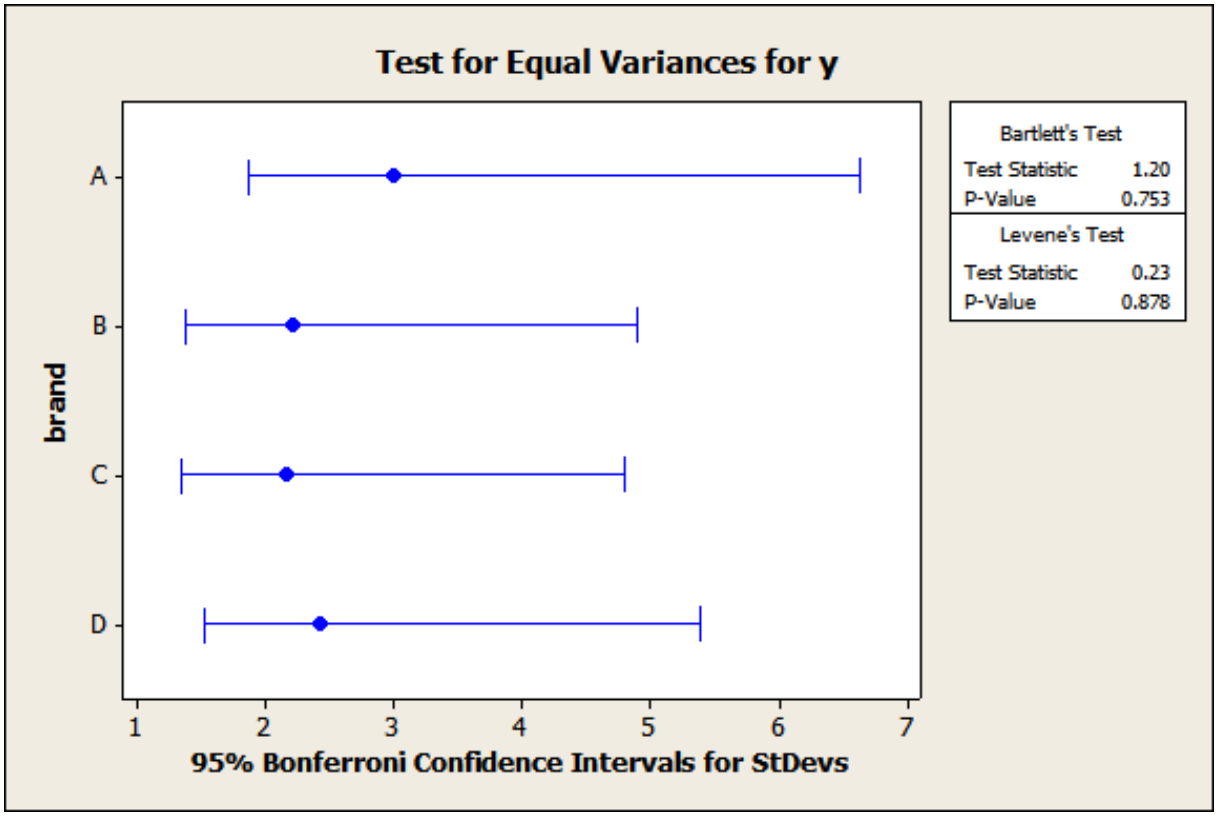

Figura 25 – Teste de *Levene* apontando amostra homocedástica [3]

Ao garantir que as amostras são normais e homocedásticas, pode-se aplicar um teste paramétrico para verificar se as médias das amostras antes e depois do *code review* são significativamente distintas, do ponto de vista estatístico. Elegeu-se o *Teste T* com os fatores "antes" (ACR) e "depois" (DCR) da aplicação do *code review*. São as hipóteses:

- H0 (hipótese nula): não há diferença entre as médias;
- H1 (hipótese alternativa): há diferença entre as médias.

Um *p-value* maior que 0,05 indica que deve-se descartar a hipótese nula, como é exemplificado na figura 26. Caso os dados não sejam normais e homocedásticos, aplica-se testes não paramétricos para chegar a esta conclusão. Para estas situações aplica-se o teste de *Mann-Whitney*, o que pode ser visto na figura 27.

Paired T-Test and CI: Before, After

```
Paired T for Before - After

               N      Mean     StDev   SE Mean
Before        10   5.38000   3.20583   1.01377

After         10   4.86000   3.10312   0.98129
Difference    10  0.520000  0.407704  0.128927


95% lower bound for mean difference: 0.283662
T-Test of mean difference = 0 (vs > 0): T-Value = 4.03  P-Value = 0.001
```

Figura 26 – *Teste T* para asserção da significância estatística da amostra [4]

Mann-Whitney Test and CI: Female_Multitask, Male_Multitask

```
                   N  Median
Female_Multitask  10   75.00
Male_Multitask    10   55.00


Point estimate for ETA1-ETA2 is 10.00
95.5 Percent CI for ETA1-ETA2 is (-9.99,30.01)
W = 120.0
Test of ETA1 = ETA2 vs ETA1 > ETA2 is significant at 0.1365
The test is significant at 0.1271 (adjusted for ties)
```

Figura 27 – Teste de *Mann-Whitney* para asserção da significância estatística da amostra [5]

6.3 Execução

6.4 Apresentação dos resultados

6.5 Discussão dos resultados

7 CONCLUSÃO

7.1 Ameaças

7.2 Trabalhos futuros

7.3 Considerações finais

REFERÊNCIAS

[1] GOUSIOS, G.; PINZGER, M.; DEURSEN, A. v. An exploratory study of the pull-based software development model. In: ACM. *Proceedings of the 36th International Conference on Software Engineering*. [S.l.], 2014. p. 345–355.

[2] Pennsylvania State University. *Tests for Error Normality*. 2017. [Online; Accessed 12/06/2017]. Disponível em: <https://onlinecourses.science.psu.edu/stat501/node/366>.

[3] WANG, J. *Using MINITAB*. 2009. [Online; Accessed 12/06/2017]. Disponível em: <http://www.stat.wmich.edu/wang/664/egs/ MTBrust.html>.

[4] Pennsylvania State University. *Lesson 9: Comparing Two Groups*. 2017. [Online; Accessed 12/06/2017]. Disponível em: <https://onlinecourses.science.psu.edu/stat200/book/export/html/57>.

[5] FROST, J. *Using Hypothesis Tests to Bust Myths about the Battle of the Sexes*. 2013. [Online; Accessed 12/06/2017]. Disponível em: <http://blog.minitab.com/blog/adventures-in-statistics-2/using-hyp othesis-tests-to-bust-myths-about-the-battle-of-the-sexes>.

[6] BOEHM, B.; BASILI, V. R. Software defect reduction top 10 list. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 34, n. 1, p. 135–137, 12 2001. ISSN 0018-9162.

[7] KEMERER, C. F.; PAULK, M. C. The impact of design and code reviews on software quality: An empirical study based on psp data. *IEEE Transactions on Software Engineering*, v. 35, n. 4, p. 534–550, 07 2009. ISSN 0098-5589.

[8] MENEELY, A. et al. An empirical investigation of socio-technical code review metrics and security vulnerabilities. In: . [S.l.: s.n.], 2014. p. 37–44.

[9] MORALES, R.; MCINTOSH, S.; KHOMH, F. Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. *2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 171–180, 2015.

[10] BAVOTA, G.; RUSSO, B. Four eyes are better than two: On the impact of code reviews on software quality. In: . [S.l.: s.n.], 2015. p. 81–90.

[11] BAYSAL, O. et al. The influence of non-technical factors on code review. In: . [S.l.: s.n.], 2013. p. 122–131.

[12] BOSU, A.; CARVER, J. Impact of developer reputation on code review outcomes in oss projects: An empirical investigation. In: . [S.l.: s.n.], 2014.

[13] KONONENKO, O. et al. Investigating code review quality: Do people and participation matter? In: . [S.l.: s.n.], 2015. p. 111–120.

[14] BACCHELLI, A.; BIRD, C. Expectations, outcomes, and challenges of modern code review. In: *Proceedings of the 2013 International Conference on Software Engineering*. [S.l.]: IEEE Press, 2013. (ICSE '13), p. 712–721. ISBN 978-1-4673-3076-3.

[15] GOUSIOS, G.; STOREY, M.-A.; BACCHELLI, A. Work practices and challenges in pull-based development: The contributor's perspective. In: IEEE. *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on*. [S.l.], 2016. p. 285–296.

[16] AUDY, J. L. N.; PRIKLADNICKI, R. *Desenvolvimento distribuído de software*. [S.l.]: Elsevier, 2007.

[17] COSTA, A. M. Nicolaci-da; PIMENTEL, M. Sistemas colaborativos para uma nova sociedade e um novo ser humano. *Sistemas colaborativos. PIMENTEL, M.; FUKS, H.(Orgs.). Rio de Janeiro: Elsevier*, 2011.

[18] CASEY, V. Virtual software team project management. *Journal of the Brazilian Computer Society*, Springer, v. 16, n. 2, p. 83–96, 2010.

[19] PAGE, S. E. *The difference: How the power of diversity creates better groups, firms, schools, and societies*. [S.l.]: Princeton University Press, 2008.

[20] PRIKLADNICKI, R. et al. The best software development teams might be temporary. *IEEE Software*, v. 34, n. 2, p. 22–25, Mar 2017. ISSN 0740-7459.

[21] YU, Y. et al. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In: IEEE. *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*. [S.l.], 2014. v. 1, p. 335–342.

[22] XIA, X. et al. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In: *IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016)*. [S.l.: s.n.], 2015. p. 261–270.

[23] JIANG, J. et al. Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology*, Elsevier, v. 84, p. 48–62, 2017.

[24] BIRD, C.; CARNAHAN, T.; GREILER, M. Lessons learned from building and deploying a code review analytics platform. In: . [S.l.: s.n.], 2015. v. 2015, p. 191–201.

[25] FUKS, H. et al. Do modelo de colaboração 3c à engenharia de groupware. *Simpósio Brasileiro de Sistemas Multimídia e Web–Webmidia*, p. 0–8, 2003.

[26] BASILI, V. R.; WEISS, D. M. A methodology for collecting valid software engineering data. *IEEE Trans. Softw. Eng*, SE-10, no. 6, p. 728–738, 1984.

[27] SCHETTINO, V. J.; ARAúJO, M. A. P. Implantação da prática de code review em um modelo de desenvolvimento de software: um estudo de caso. 2017.

[28] BELLER, M. et al. Modern code reviews in open-source projects: Which problems do they fix? In: . [S.l.: s.n.], 2014. p. 202–211.

[29] MCINTOSH, S. et al. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In: . [S.l.: s.n.], 2014. p. 192–201.

[30] FAGAN, M. E. Design and code inspections to reduce errors in program development. *IBM Syst. J.*, IBM Corp., Riverton, NJ, USA, v. 15, n. 3, p. 182–211, 09 1976. ISSN 0018-8670.

[31] BARR, E. et al. Cohesive and isolated development with branches. *Fundamental Approaches to Software Engineering*, Springer, p. 316–331, 2012.

[32] GOUSIOS, G. et al. Work practices and challenges in pull-based development: the integrator's perspective. In: IEEE PRESS. *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. [S.l.], 2015. p. 358–368.

[33] BAYSAL, O. et al. The secret life of patches: A firefox case study. In: *2012 19th Working Conference on Reverse Engineering*. [S.l.: s.n.], 2012. p. 447–455. ISSN 1095-1350.

[34] MENS, T.; CATALDO, M.; DAMIAN, D. The social developer: The future of software development [guest editors' introduction]. *IEEE Software*, IEEE, v. 36, n. 1, p. 11–14, 2019.

[35] STADLER, M. et al. Agile distributed software development in nine central european teams: Challenges, benefits, and recommendations. *International Journal of Computer Science & Information Technology (IJCSIT) Vol*, v. 11, 2019.

[36] HERBSLEB, J. D.; MOITRA, D. Global software development. *IEEE software*, IEEE, v. 18, n. 2, p. 16–20, 2001.

[37] CARMEL, E.; AGARWAL, R. Tactical approaches for alleviating distance in global software development. *IEEE software*, IEEE, v. 18, n. 2, p. 22–29, 2001.

[38] KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004.

[39] YANG, C. et al. An empirical study of reviewer recommendation in pull-based development model. In: *Proceedings of the 9th Asia-Pacific Symposium on Internetware*. New York, NY, USA: ACM, 2017. (Internetware'17), p. 14:1–14:6. ISBN 978-1-4503-5313-7. Disponível em: <http://doi.acm.org/10.1145/3131704.3131718>.

[40] HANNEBAUER, C. et al. Automatically recommending code reviewers based on their expertise: An empirical comparison. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: ACM, 2016. (ASE 2016), p. 99–110. ISBN 978-1-4503-3845-5. Disponível em: <http://doi.acm.org/10.1145/2970276.2970306>.

[41] COSENTINO, V.; IZQUIERDO, J. L. C.; CABOT, J. A systematic mapping study of software development with github. *IEEE Access*, v. 5, p. 7173–7192, 2017. ISSN 2169-3536.

[42] WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012.

[43] PETTICREW, M.; ROBERTS, H. *Systematic reviews in the social sciences: A practical guide*. [S.l.]: John Wiley and Sons, 2008.

[44] RAHMAN, M. M.; ROY, C. K.; COLLINS, J. A. Correct: code reviewer recommendation in github based on cross-project and technology experience. In: IEEE. *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*. [S.l.], 2016. p. 222–231.

[45] BALACHANDRAN, V. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In: IEEE. *Software Engineering (ICSE), 2013 35th International Conference on*. [S.l.], 2013. p. 931–940.

[46] ZANJANI, M. B.; KAGDI, H.; BIRD, C. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering*, IEEE, v. 42, n. 6, p. 530–543, 2016.

[47] LIAO, Z. et al. Topic-based integrator matching for pull request. *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, p. 1–6, 2017.

[48] COSTA, C. et al. Tipmerge: recommending experts for integrating changes across branches. In: ACM. *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. [S.l.], 2016. p. 523–534.

[49] FEJZER, M.; PRZYMUS, P.; STENCEL, K. Profile based recommendation of code reviewers. *Journal of Intelligent Information Systems*, Aug 2017. ISSN 1573-7675. Disponível em: <https://doi.org/10.1007/s10844-017-0484-1>.

[50] JIANG, J.; HE, J.-H.; CHEN, X.-Y. Coredevrec: Automatic core member recommendation for contribution evaluation. *Journal of Computer Science and Technology*, Springer, v. 30, n. 5, p. 998–1016, 2015.

[51] YANG, X. et al. Peer review social network (person) in open source projects. *IEICE TRANSACTIONS on Information and Systems*, The Institute of Electronics, Information and Communication Engineers, v. 99, n. 3, p. 661–670, 2016.

[52] THONGTANUNAM, P. et al. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In: . [S.l.: s.n.], 2015. p. 141–150.

[53] LEE, J. B. et al. Patch reviewer recommendation in oss projects. In: *2013 20th Asia-Pacific Software Engineering Conference (APSEC)*. [S.l.: s.n.], 2013. v. 2, p. 1–6. ISSN 1530-1362.

[54] XIA, Z. et al. A hybrid approach to code reviewer recommendation with collaborative filtering. In: *2017 6th International Workshop on Software Mining (SoftwareMining)*. [S.l.: s.n.], 2017. p. 24–31.

[55] YING, H. et al. Earec: leveraging expertise and authority for pull-request reviewer recommendation in github. In: ACM. *Proceedings of the 3rd International Workshop on CrowdSourcing in Software Engineering*. [S.l.], 2016. p. 29–35.

[56] YU, Y. et al. Reviewer recommender of pull-requests in github. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. [S.l.: s.n.], 2014. p. 609–612. ISSN 1063-6773.

[57] XIA, X. et al. Who should review this change?: Putting text and file location analyses together for more accurate recommendations. In: IEEE. *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. [S.l.], 2015. p. 261–270.

[58] OUNI, A.; KULA, R. G.; INOUE, K. Search-based peer reviewers recommendation in modern code review. In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.: s.n.], 2016. p. 367–377.

[59] FU, C. et al. Expert recommendation in oss projects based on knowledge embedding. In: . [S.l.: s.n.], 2017. p. 149–155.

[60] BOSU, A.; GREILER, M.; BIRD, C. Characteristics of useful code reviews: An empirical study at microsoft. In: *Proceedings of the 12th Working Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE Press, 2015. (MSR '15), p. 146–156. ISBN 978-0-7695-5594-2. Disponível em: <http://dl.acm.org/citation.cfm?id=2820518.2820538>.

[61] RAHMAN, M. M.; ROY, C. K.; KULA, R. G. Predicting usefulness of code review comments using textual features and developer experience. In: *Proceedings of the 14th International Conference on Mining Software Repositories*. Piscataway, NJ, USA: IEEE Press, 2017. (MSR '17), p. 215–226. ISBN 978-1-5386-1544-7. Disponível em: <https://doi.org/10.1109/MSR.2017.17>.

[62] LOPEZ-FERNANDEZ, L. et al. Applying social network analysis to the information in cvs repositories. In: IET. *International workshop on mining software repositories*. [S.l.], 2004. p. 101–105.

[63] MENEELY, A. et al. Predicting failures with developer networks and social network analysis. In: ACM. *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. [S.l.], 2008. p. 13–23.

[64] ZHANG, T.; LEE, B. How to recommend appropriate developers for bug fixing? In: *2012 IEEE 36th Annual Computer Software and Applications Conference*. [S.l.: s.n.].

[65] LI, B.; KING, I. Routing questions to appropriate answerers in community question answering services. In: *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*. New York, NY, USA: ACM, 2010. (CIKM '10), p. 1585–1588. ISBN 978-1-4503-0099-5. Disponível em: <http://doi.acm.org/10.1145/1871437.1871678>.

[66] FOGEL, K. *Producing open source software: How to run a successful free software project*. [S.l.]: "O'Reilly Media, Inc.", 2005.

[67] CROSS, R. L.; CROSS, R. L.; PARKER, A. *The hidden power of social networks: Understanding how work really gets done in organizations*. [S.l.]: Harvard Business Press, 2004.

[68] BERGQUIST, M.; LJUNGBERG, J. The power of gifts: organizing social relationships in open source communities. *Information Systems Journal*, Wiley Online Library, v. 11, n. 4, p. 305–320, 2001.

[69] WIKI, M. *Contribute - Mozilla Wiki*. 2018. Accessed 2018-09-17.

[70] WIKI, D. *Teams - Debian Wiki*. 2018. Accessed 2018-09-17.

[71] MENG, Z. et al. Empirical Study on Overlapping Community Detection in Question and Answer Sites. In: *Advances in Social Networks Analysis and Mining (ASONAM), 2014 IEEE/ACM International Conference on*. Beijing, China: [s.n.], 2014. Disponível em: <https://hal.inria.fr/hal-01075944>.

[72] HORTA, V. et al. Analyzing scientific context of researchers and communities by using complex network and semantic technologies. *Future Generation Computer Systems*, Elsevier, v. 89, p. 584–605, 2018.

[73] ESTER, M. et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Kdd*. [S.l.: s.n.], 1996. v. 96, n. 34, p. 226–231.

[74] TAN, P.-N.; STEINBACH, M.; KUMAR, V. *Introduction to data mining. 1st*. [S.l.]: Boston: Pearson Addison Wesley. xxi, 2005.

[75] ALMEIDA, H. et al. Is there a best quality metric for graph clusters? In: SPRINGER. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. [S.l.], 2011. p. 44–59.

[76] KOVALENKO, V.; BACCHELLI, A. Code review for newcomers: Is it different? In: *Proceedings of the 11th International Workshop on Cooperative and Human Aspects of Software Engineering*. New York, NY, USA: ACM, 2018. (CHASE '18), p. 29–32. ISBN 978-1-4503-5725-8. Disponível em: <http://doi.acm.org/10.1145/3195836.3195842>.

[77] FREIRE, J.; BONNET, P.; SHASHA, D. Computational reproducibility: state-of-the-art, challenges, and database research opportunities. In: ACM. *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. [S.l.], 2012. p. 593–596.

[78] INCE, D. C.; HATTON, L.; GRAHAM-CUMMING, J. The case for open computer programs. *Nature*, Nature Research, v. 482, n. 7386, p. 485–488, 2012.

[79] BOETTIGER, C. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, ACM, v. 49, n. 1, p. 71–79, 2015.

[80] SINHA, R.; SUDHISH, P. S. A principled approach to reproducible research: a comparative review towards scientific integrity in computational research. In: IEEE. *Ethics in Engineering, Science and Technology (ETHICS), 2016 IEEE International Symposium on*. [S.l.], 2016. p. 1–9.

[81] OPEN SOURCE INITIATIVE AND OTHERS. *The MIT license*. Disponível em: <https://opensource.org/licenses/MIT>.

[82] LAMPORT, L. *LATEX: a document preparation system: user's guide and reference manual*. [S.l.]: Addison-wesley, 1994.

[83] FIELDING, R. T.; TAYLOR, R. N. Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, ACM, v. 2, n. 2, p. 115–150, 2002.

[84] GILBERT, S.; LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News*, ACM, v. 33, n. 2, p. 51–59, 2002.

[85] MILLER, J. et al. Statistical power and its subcomponents—missing and misunderstood concepts in empirical software engineering research. *Information and Software Technology*, Elsevier, v. 39, n. 4, p. 285–295, 1997.

[86] DYBÅ, T.; KAMPENES, V. B.; SJØBERG, D. I. A systematic review of statistical power in software engineering experiments. *Information and Software Technology*, Elsevier, v. 48, n. 8, p. 745–755, 2006.

[87] ARAúJO, M. A. P. et al. Métodos estatísticos aplicados em engenharia de software experimental. In: SOFTWARE, X. S. B. de Engenharia de (Ed.). *ANAIS do XX SBBD*. [S.l.], 2006. p. 325.