UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Vinicius Junqueira Schettino

Recomendação de Revisores de Código no Desenvolvimento Distribuído de Software

Juiz de Fora
2019

Vinicius Junqueira Schettino

Recomendação de Revisores de Código no 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: Victor Ströele de Andrade Menezes

Coorientador: Marco Antônio Pereira Araújo

Juiz de Fora

2019

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

Junqueira Schettino, Vinicius.
Recomendação de Revisores de Código no Desenvolvimento Distribuído de Software / Vinicius Junqueira Schettino. – 2019.
93 f. : il.

Orientador: Victor Ströele de Andrade Menezes
Coorientador: Marco Antônio Pereira Araújo
Dissertação (Mestrado) – Universidade Federal de Juiz de Fora, Instituto de Ciências Exatas. Programa de Pós-graduação em Ciência da Computação, 2019.

1. revisão de código. 2. desenvolvimento distribuído. 3. sistemas de recomendação. I. Ströele de Andrade Menezes, Victor, orient. II. Pereira Araújo, Marco Antônio, coorient. III. Título.

Vinícius Junqueira Schettino

"Recomendação de Revisores de Código no 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 à obtenção do grau de Mestre em Ciência da Computação.

Aprovada em 25 de novembro de 2019.

BANCA EXAMINADORA

Prof. Dr. Victor Ströele de Andrade Menezes – Orientador
Universidade Federal de Juiz de Fora

Prof. Dr. Marco Antônio Pereira Araújo – Coorientador
Universidade Federal de Juiz de Fora

Prof. Dr. José Maria Nazar David
Universidade Federal de Juiz de Fora

Prof. Dr. Rodrigo Pereira dos Santos
Universidade Federal do Estado do Rio de Janeiro

Dedico este trabalho a todos que, direta ou indiretamente, contribuíram com o meu aprendizado e vivência profissional.

Não quero lhe falar meu grande amor
Das coisas que aprendi nos discos
Quero lhe contar como eu vivi
E tudo o que aconteceu comigo...
(Belchior, Como Nossos Pais 1976)

RESUMO

A revisão de código é uma das principais técnicas de diminuição de defeitos de software. Intrinsecamente colaborativo, o processo envolve a análise das modificações no código fonte por um revisor, de acordo com as diretrizes de cada repositório. Contudo, a eficiência da técnica está diretamente associada à escolha dos pares técnicos adequados para realizar o escrutínio. Especialmente no Desenvolvimento Distribuído de Software (DDS), onde os membros da equipe de desenvolvimento estão espalhados geograficamente com restrições de horário e disponibilidade, além de diferenças culturais e técnicas, a escolha do revisor vira tarefa complexa sem auxílio de ferramentas computacionais. O objetivo do presente trabalho é apresentar o GitRev, uma plataforma que reúne diferentes métodos de recomendação de revisores para o desenvolvimento distribuído de software, potencializando a colaboração no processo de revisão. Levando em consideração as particularidades dos processos descentralizados de produção, são propostos métodos baseados nas interações históricas entre os desenvolvedores. Esta análise é realizada através da modelagem de uma rede social colaborativa tendo como base os dados oriundos dos repositórios *open source* hospedados no GitHub. Uma pesquisa histórica é conduzida para avaliar a solução apresentada, calcada na literatura relacionada e em métricas objetivas de eficiência da participação dos revisores e instanciado em quatro grandes projetos disponíveis na plataforma. Com os resultados obtidos, é possível observar que os revisores recomendados aumentam a interação e colaboração entre os envolvidos e aderem significativamente aos objetivos do processo de revisão.

Palavras-chave: Revisão de código. Desenvolvimento Distribuído de Software. Sistemas de Recomendação.

ABSTRACT

Code review is one of the main software defect mitigation techniques. Intrinsically collaborative, the process involves the analysis of source code modifications by technical peer in adherence to the guidelines of each repository. However, the efficiency of this approach is directly associated with choosing the appropriate technical peers to conduct the scrutiny. Especially in global software development, where members of the development team are geographically scattered with time and availability constraints and there are cultural and technical differences, choosing the reviewer becomes a complex task without the aid of computational tools. The aim of this paper is to present Gitrev, a tool that gathers reviewer recommendation methods for distributed software development that enhance collaboration in the review process. Taking into account the particularities of decentralised production processes, the proposed methods are based on historical interactions between developers, through the design of a collaborative network based on data from GitHub hosted open source repositories. The evaluation of the presented solution is a historic research based on the related literature and objective metrics of efficiency of the reviewers' participation and instantiated on four major projects available on the platform. The results shows the recommended reviewers enhance interaction and collaboration among team members and adhere significantly to the objectives of the review process.

Key-words: Code Review, Distributed Software Development, Recommendation Systems

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

UFJF	Universidade Federal de Juiz de Fora
DDS	Desenvolvimento Distribuído de Software
GSD	*Global Software Development*
DDS	*Desenvolvimento Distribuído de Software*

SUMÁRIO

1 INTRODUÇÃO

Organizações tecnológicas de diversos portes buscam se manter competitivas atuando não como fornecedores isolados, mas sim em uma rede complexa onde a interação entre produtos, tecnologias e pessoas de contextos distintos é a chave para qualquer abordagem de sucesso (JANSEN; FINKELSTEIN; BRINKKEMPER, 2009) (JANSEN; FINKELSTEIN; BRINKKEMPER, 2009). Tal prática é fomentada pelos objetivos de negócio, ao se aumentar a satisfação do cliente com mais funcionalidades disponíveis em menos tempo. Diversos desafios de gerenciamento, qualidade e colaboração, embora já existentes no desenvolvimento tradicional, se potencializam no cenário distribuído e incentivam equipes técnicas na busca por novas soluções (COSTA; PIMENTEL, 2011).

Diversas abordagens se solidificam como meios de se manter a qualidade dos componentes desenvolvidos (JANSEN; CAPELLEVEEN, 2013). O *code review* é considerado uma das principais técnicas para diminuição de defeitos durante o processo de desenvolvimento de software (BOEHM; BASILI, 2001). Nela, o autor de uma alteração em um projeto submete o código 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* (KEMERER; PAULK, 2009) de acordo com o nível de participação dos envolvidos e cobertura do código revisado (MENEELY et al., 2014; MORALES; MCINTOSH; KHOMH, 2015; BAVOTA; RUSSO, 2015). Reputação (BAYSAL et al., 2013; BOSU; CARVER, 2014) e experiência (KONONENKO et al., 2015) 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 (BACCHELLI; BIRD, 2013), principalmente em DDS. Dentro de workflows descentralizados (GOUSIOS; STOREY; BACCHELLI, 2016), a prática funciona como um *gateway* de qualidade que busca garantir que apenas alterações aderentes aos padrões de qualidade do projeto serão incorporados à base de código 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 ramo da Engenharia de Software vem modificando a relação entre empresas e sistemas, e estratégias de negócios (AUDY; PRIKLADNICKI, 2007). 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 desafios à colaboração co-localizada são potencializados e as soluções tradicionais não são suficientes para fomentar este aspecto das atividades distribuídas (COSTA; PIMENTEL, 2011). (CASEY, 2010) 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 DDS afetam o *code review* de duas formas distintas. Primeiro, o processo de revisão pode se tornar lento e ineficiente quando a colaboração é afetada, já que poucos podem participar (baixa participação) e parte do código pode deixar de ser analisada (baixa 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 trecho de código modificado (*patch*). Com um vasto número de opções e pouca informação consolidada sobre seus aspectos técnicos e gerenciais (e.g. tempo disponível), a falta de contato co-localizado e 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 demonstram que a fixação de grupos e responsabilidades pode não ser benéfica para o processo de desenvolvimento. Autores argumentam que a diversidade de experiências, visões e especialidades fazem com que grupos sejam mais eficientes (PAGE, 2008). (PRIKLADNICKI et al., 2017) apresentam indícios de que a formação de grupos temporários em detrimento ou em conjunto com permanentes é um fator de eficiência em projetos de software:

> "Apesar de colegas experientes contribuirem com conhecimento de processos e normas de projetos antigos, novos membros trazem ideias frescas que podem promover o desempenho e a criatividade do projeto. Os membros antigos podem não fazê-lo e ainda não dar espaço para os novatos implementarem suas ideias."

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* (BACCHELLI; BIRD, 2013).

Existem alguns trabalhos congêneres que demonstram métodos de recomendação de revisores (YU et al., 2014b; XIA et al., 2015b; JIANG et al., 2017). Esses trabalhos

foram estudados e levados em consideração para escrita do presente texto. Também foram revisadas pesquisas que apontam características de revisões, revisores e autores que podem potencializar a colaboração (KEMERER; PAULK, 2009; BIRD; CARNAHAN; GREILER, 2015; BAYSAL et al., 2013). Tais aspectos são apresentados e discutidos no Capítulo 2.

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 (FUKS et al., 2003).

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 eficiência 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 estendidas no Capítulo 3.

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. (BASILI; WEISS, 1984), postula-se o objetivo do trabalho como: **Implementar** métodos 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:

- Os métodos de recomendação apresentados podem 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 (BACCHELLI; BIRD, 2013). O GitHub é uma plataforma rica em repositórios de projetos de software. Muitos são de código aberto, disponíveis para mineração. Discussões sobre o *workflow* 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*" (GOUSIOS; PINZGER; DEURSEN, 2014). Essa abordagem é explorada na seção 2.1.1.

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 4.

A metodologia pode ser resumida em quatro etapas principais: (i) revisão da literatura; (ii) definição dos métodos de recomendação; (iii) modelagem e implementação da plataforma de recomendação; (iv) avaliação dos métodos propostos.

Na revisão da literatura são analisados estudos prévios relacionados, métodos propostos e quais lacunas ficam evidentes sob a ótica do desenvolvimento distribuído. Os resultados dessa etapa são utilizados para propor os métodos de recomendação, evidenciando as divergências e evoluções em relação ao estado da arte. A plataforma de recomendação é modelada de acordo com os requisitos funcionais observados na etapa de revisão da literatura e dos não funcionais extraídos das definições dos modelos de recomendação. Por fim, os métodos são avaliados em aderência com suas características intrínsecas e com as sugestões dos trabalhos relacionados.

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, calcado no método de pesquisa histórico e nos dados pregressos de projetos renomados, é devidamente discutido e aplicado no Capítulo 5, incluindo a apresentação dos experimentos e a revisitação da hipótese levantada neste capítulo. Por fim, o Capítulo 6 é dedicado ao fechamento do trabalho, incluindo a sugestão de trabalhos futuros e a discussão de ameaças a validade e generalização dos resultados apresentados.

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

2 REFERENCIAL TEÓRICO

Este capítulo tem como objetivo apresentar aspectos do contexto e do estado da arte da recomendação de revisores de código. Primeiro são apresentados os conceitos correlatos, buscando homogenizar o entendimento e os fundamentos do presente trabalho. As seções posteriores apresentam os principais resultados da revisão e mapeamento da literatura relacionada (SCHETTINO et al., 2019b)[1] que foram conduzidos.

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.

O *code review* está associado diretamente à detecção precoce de defeitos em produtos de software (SCHETTINO; ARAúJO, 2017; KEMERER; PAULK, 2009), sendo reconhecida como uma das principais técnicas com este fim (BOEHM; BASILI, 2001). Mais especificamente, é relatada maior eficiência quanto aos defeitos não-funcionais, enquanto os defeitos funcionais são menos afetados no processo (BELLER et al., 2014). Outros autores reportam a diminuição de defeitos através de estudos de caso (MCINTOSH et al., 2014; BAVOTA; RUSSO, 2015; MORALES; MCINTOSH; KHOMH, 2015). A atividade de revisão remonta da década de 80 (FAGAN, 1976), 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 (BACCHELLI; BIRD, 2013).

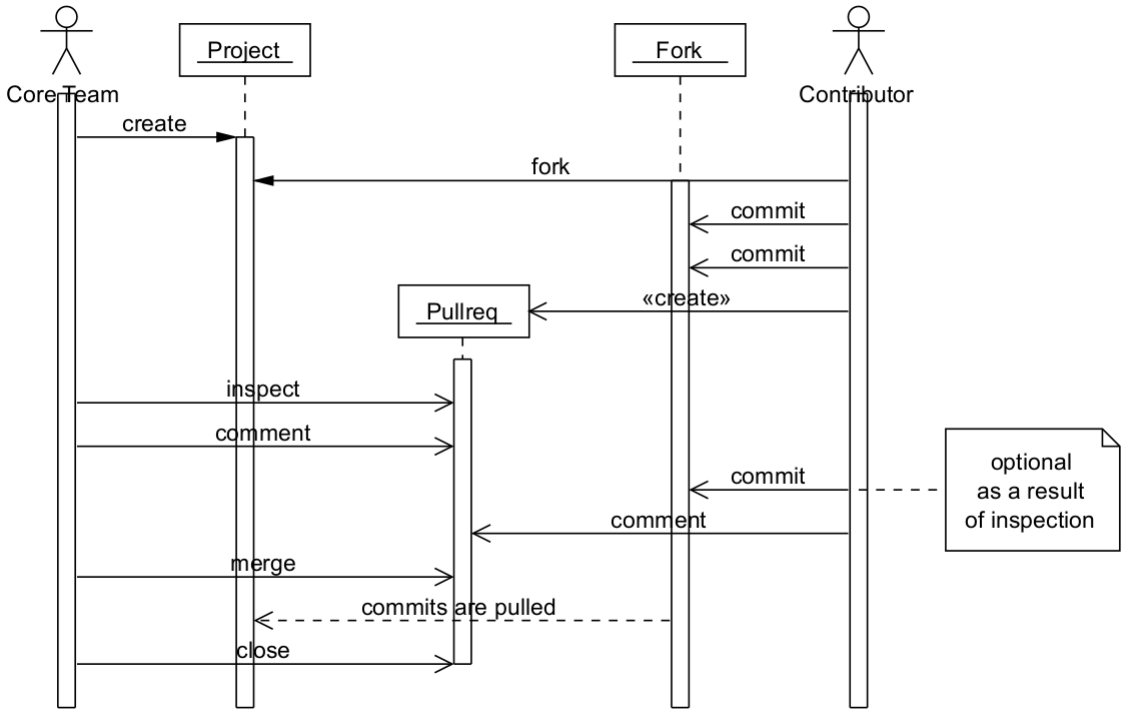
2.1.1 **Pull Based Method**

O conceito de *branches* é a base para sistemas de controle de versão descentralizados, como o Git[2] e o Mercurial[3]. Com as *branches* é possível desenvolver paralelamente, submetendo 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 (BARR et al., 2012).

[1] disponivel em https://github.com/vschettino/gitrev/raw/master/mapeamento.pdf
[2] https://git-scm.com/
[3] https://www.mercurial-scm.org/

Estas tecnologias permitiram o surgimento de um paradigma de desenvolvimento baseado em pulls, ou *pull-based method* (GOUSIOS; PINZGER; DEURSEN, 2014). 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 (GOUSIOS et al., 2015). A Figura 1 ilustra tal modelo de trabalho instanciado no GitHub[4], principal expoente que oferece este paradigma. Nele é representado um modelo comum em desenvolvimento OpenSource (BAYSAL et al., 2012), 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"* (GOUSIOS; PINZGER; DEURSEN, 2014)



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 (GOUSIOS; STOREY; BACCHELLI, 2016). Ao terminar, ele solicita a integração da branch do *fork* de volta ao projeto original. Essa solicitação 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*.

[4] https://github.com

Os membros do core também têm suas *branches* revisadas por um processo análogo (BAYSAL et al., 2012; BOSU; CARVER, 2014). 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 DDS vem em crescente utilização no cenário atual (MENS; CATALDO; DAMIAN, 2019). 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 (STADLER et al., 2019). Esta mudança no paradigma tradicional de desenvolvimento co-localizado 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 (HERBSLEB; MOITRA, 2001).

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 características inerentes ao desenvolvimento distribuído ameaçam a produtividade ao dificultar os processos que envolvem coordenação, comunicação e cooperação (CARMEL; AGARWAL, 2001).

2.2.1 **Motivação da revisão e do mapeamento sistemático**

De acordo com parâmetros e diretrizes estabelecidos (KITCHENHAM, 2004), 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. (YANG et al., 2017) apresentam diversos estudos e aplicam um método conhecido numa grande base de dados, tirando conclusões sobre como a eficiência 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.

Autores reconhecem que encontrar revisores adequados é um desafio e focam em comparar empiricamente oito diferentes abordagens em quatro conhecidos projetos *open source* (HANNEBAUER et al., 2016). Já (COSENTINO; IZQUIERDO; CABOT, 2017) 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 (WOHLIN et al., 2012).

2.3 REVISÃO SISTEMÁTICA DA LITERATURA

As diretrizes para embasar pesquisas sistemáticas em Ciência da Computação foram propostas por (KITCHENHAM, 2004), ao adaptar abordagens e outras áreas, especialmente das Ciências Médicas. 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 (WOHLIN et al., 2012). 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 as etapas do estudo foram concretizadas por quatro diferentes especialistas, em análises distintas. Todos os critérios de inclusão e exclusão dos trabalhos e eventuais divergências foram discutidos em profundidade antes da definição dos resultados.

2.3.1 **Questões de pesquisa**

No trabalho conduzido, as seguintes questões de mapeamento (MQ) foram escolhidas para caracterizar a área de recomendação de revisores de código:

- **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

- **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.
- **MQ3: Em quais contextos a recomendação de revisores de código é utilizada? (DDS, indústria, Código Livre)** O objetivo é entender quais contextos potencializam a necessidade de métodos automatizados de recomendação.

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

- **RQ1: Como a eficiência da recomendação dos revisores 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.
- **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 (BASILI; WEISS, 1984)) e os termos definidos pelo PICOC (PETTICREW; ROBERTS, 2008) 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 2.1.1) fossem encontrados. Assim, a seguinte *string* foi gerada:

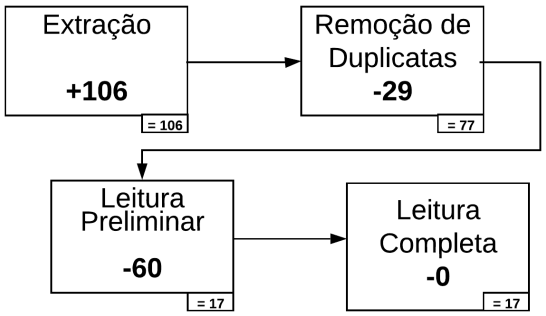
("software developer"OR developer) AND ("reviewer recommendation"OR "commenter recommendation") AND (method OR tool OR solution OR framework) AND ("code review"OR "pull-request"OR "pull request")

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 [5], útil para acompanhar o processo sistemático proposto por (KITCHENHAM,

[5] http://parsif.al

2004). Na ferramenta foi possível remover trabalhos duplicados, classificar 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



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.

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 distribuído e ao desenvolvimento "pull-based". Esta associação parece ser embasada aos desafios inerentes do desenvolvimento global. Autores justificam este fenômeno através do maior número de revisões e revisores, o que dificulta a seleção manual dos envolvidos (YING et al., 2016). Outros mostram que encontrar o melhor revisor para um "pull request" é um trabalho tipicamente de "crowdsourcing" (YU et al., 2014b). 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 (RAHMAN; ROY; COLLINS, 2016).

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

Tabela 1 – Publicações por veículos

Reference	Channel	h5-index
(RAHMAN; ROY; COLLINS, 2016)(BALACHANDRAN, 2013)	ICSE - IEEE/ACM International Conference on Software Engineering	68
(ZANJANI; KAGDI; BIRD, 2016)	IEEE Transactions on Software Engineering	52
(LIAO et al., 2017)	GLOBECOM - IEEE Global Communications Conference	48
(COSTA et al., 2016)	SIGSOFT - ACM International Symposium on Foundations of Software Engineering	43
(FEJZER; PRZYMUS; STENCEL, 2017)	Journal of Intelligent Information Systems	22
(JIANG; HE; CHEN, 2015)	Journal of Computer Science and Technology	22
(YANG et al., 2016)	IEICE Transactions on Information and Systems	17
(THONGTANUNAM et al., 2015)	SANER - IEEE International Conference on Software Analysis, Evolution and Reengineering	13
(YU et al., 2014b; LEE et al., 2013)	APSEC Asia-Pacific Software Engineering Conference	12
(XIA et al., 2017)	IEEE/ACM International Workshop on Software Mining	*
(YING et al., 2016)	International Workshop on Crowd-Sourcing in Software Engineering	*
(YU et al., 2014a)(XIA et al., 2015a)(OUNI; KULA; INOUE, 2016)	ICSME - IEEE International Conference on Software Maintenance and Evolution	*
(FU et al., 2017)	IWCSN - International Workshop on Complex Systems and Networks	*

adequados) podem aumentar a retenção de desenvolvedores e influenciar no sucesso do projeto (FU et al., 2017). 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 (LEE et al., 2013), o atraso no assimilação do código proposto à *codebase* principal pode atrasar as entregas e desencorajar novas contribuições (JIANG; HE; CHEN, 2015). Assim, recomendações 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 carentes 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.

2.4 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.

2.4.1 **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; PRZYMUS; STENCEL, 2017) propõem utilizar a similaridade do código submetido com o perfil do revisor, Liao et al. (LIAO et al., 2017) definem um conjunto de tópicos que descrevem a experiência e que pode ser comparado com a necessidade da revisão atual. Thongtanunam et al. (THONGTANUNAM et al., 2015) apresenta um abordagem onde o caminho dos arquivos revisados são comparados com o histórico de revisão dos potenciais indicados.

2.4.2 **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. (COSTA et al., 2016) analisam quais foram os responsáveis por mudanças desta região para encontrar candidatos. Rahman et al. (RAHMAN; ROY; COLLINS, 2016) apresentam o CORRECT, um método que analisa a experiência dos desenvolvedores em projetos diferentes mas que contam com tecnologias compartilhadas para recomendar.

2.4.3 **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. (FU et al., 2017) propõe utilzar um grafo baseado nas relações sociais dos desenvolvedores. Xia et al. (XIA et al., 2017) captura relações implícitas e valoriza interaços mais recentes. Yu et al. (YU et al., 2014b; YU et al., 2014a) 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. (YANG et al., 2016) emprega análise de redes sociais para encontrar revisores baseado na intensidade dos relacionamentos.

2.4.4 **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. (YING et al., 2016) consideram tanto a experiência dos desenvolvedores quanto a autoridade no processo de desenvolvimento. Através da ferramenta CoreDevRec, Jiang et al. (JIANG; HE; CHEN, 2015) 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.

2.5 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. (YANG et al., 2016) leva em consideração qual o impacto das recomendações na interatividade das revisões. Outros trabalhos também discutem o impacto positivo dos seus métodos na eficiência do processo de revisão (XIA et al., 2017; LIAO et al., 2017). A análise focada no desempenho e impacto dos métodos de revisão é motivação do presente trabalho, e fomenta aspectos que

são detalhados na seção 2.6

2.6 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. (YANG et al., 2016). 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. (BOSU; GREILER; BIRD, 2015) conduziram um estudo experimental 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. (RAHMAN; ROY; KULA, 2017) analisam os textos das discussões e encontram 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 (MENEELY et al., 2014; MORALES; MCINTOSH; KHOMH, 2015; BAVOTA; RUSSO, 2015). 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 (BOSU; GREILER; BIRD, 2015).

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 3 apresenta a análise dos datasets escolhidos e embasamento empírico dos métodos propostos.

3 MÉTODOS DE RECOMENDAÇÃO

A facilidade na disponibilização de recursos na Web proporciona um volume expressivo de informações disponíveis. Dessa forma, os usuários demonstram dificuldade em encontrar recursos que são de seu interesse e que sejam aderentes às suas necessidades. Neste contexto, os Sistemas de Recomendação (SR) realizam a filtragem de informações, analisando o perfil e o, contexto dos usuários, posteriormente recomendando-lhes recursos de potencial interesse. 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 recomendação 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.

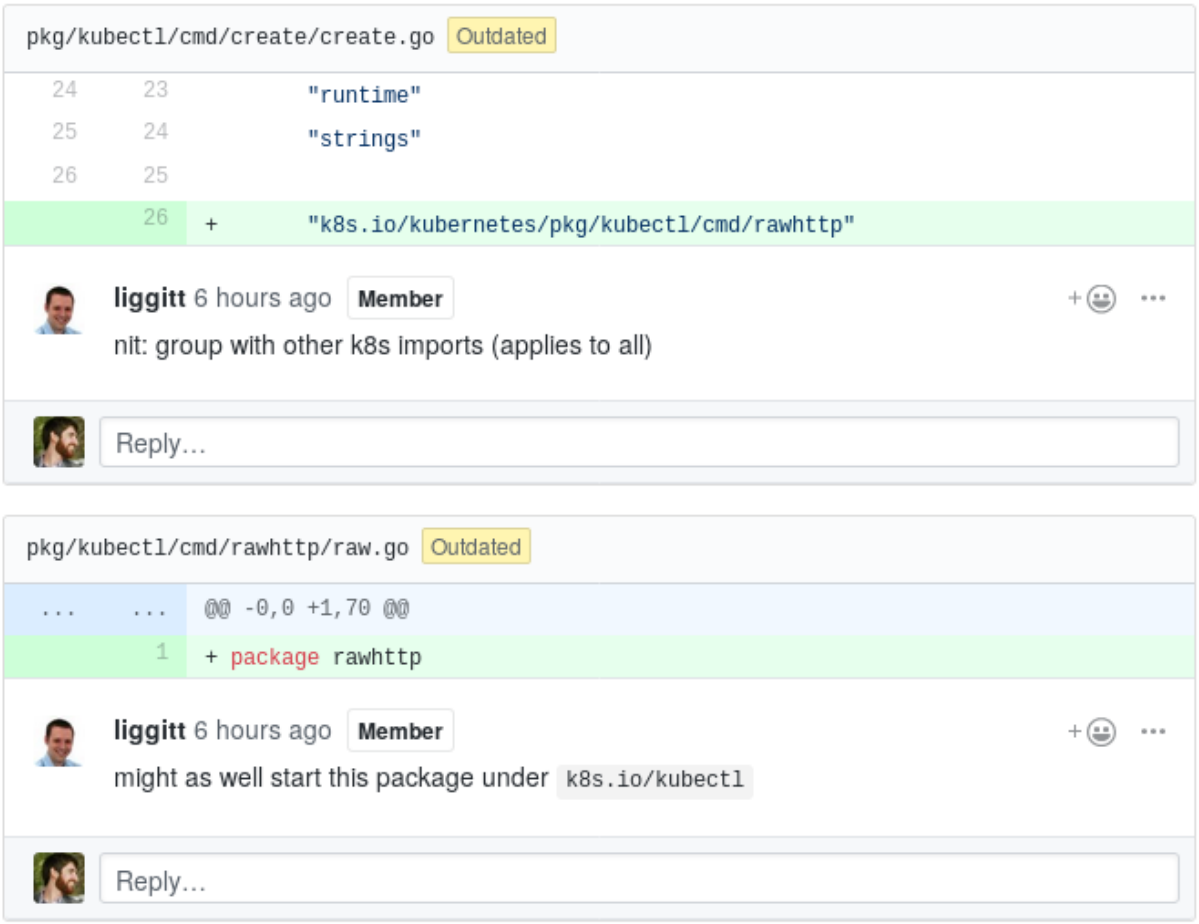
3.1 REDES DE DESENVOLVEDORES

Interações durante do desenvolvimento de software caracterizam o surgimento de redes sociais de desenvolvedores, denominadas *"social networks"* ou *"developer networks"* (LOPEZ-FERNANDEZ et al., 2004). Observar tais estruturas como grafos é o ponto de partida de diversas aplicações deste tipo de análise, como previsão de defeitos (MENEELY et al., 2008), triagem de tarefas de manutenção (ZHANG; LEE, 2012) e encontrar os melhores profissionais para responder determinada dúvida (LI; KING, 2010; HORTA et al., 2019). 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 (SCHETTINO et al., 2019b).

Uma rede social pode ser representada como um conjunto de nós conectados entre si. O objetivo é refletir nesta estrutura 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.

[1] https://github.com
[2] https://gitlab.com

Figura 3 – Interação entre autor e revisor durante a revisão



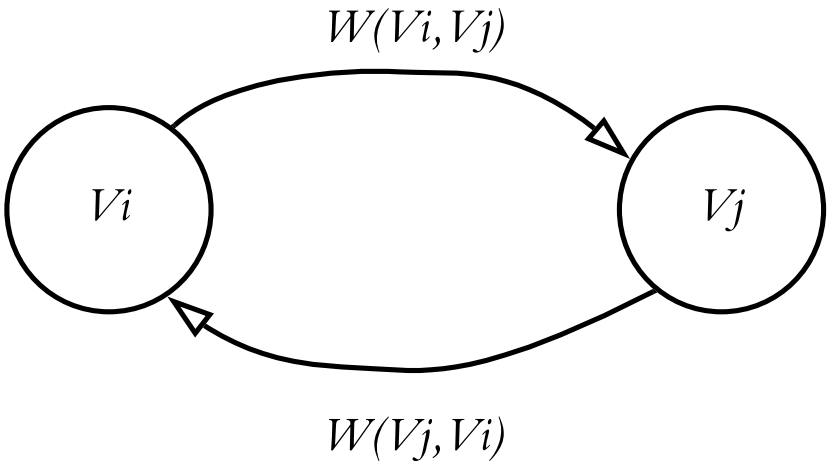
Através da API RESTful[3] do GitHub é possível interagir com o repositório e obter estas informações automaticamente, possibilitando a modelagem da rede.

3.2 MODELAGEM DA REDE

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.

[3] https://developer.github.com/v3/

Figura 4 – Representação dos vértices e arestas da rede

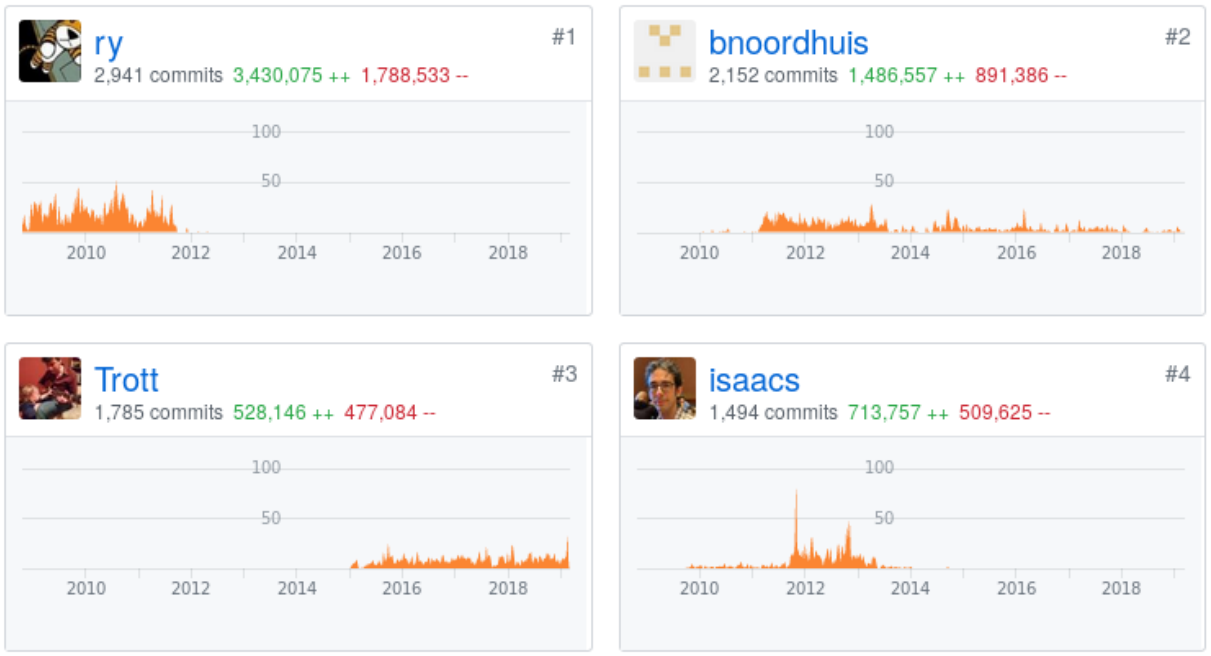


Para evitar que interações antigas de desenvolvedores que podem nem estar ativos mais no projeto enviesem as análises, tais ocorrências são penalizadas de modo a valer menos do que interações mais novas. Em sistemas de recomendação baseados em redes sociais, é comum aplicar esse tipo de tratamento nos valores para promover a importância dos indivíduos recomendados (CAPURUÇO; CAPRETZ, 2010). Não são raras as situações em projetos *"Open Source"*nas quais as responsabilidades e participações mudam com o tempo (FOGEL, 2005), além dos casos onde novos desenvolvedores chegam para ocupar as lacunas deixadas 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 contemporâneo do idealizador do projeto mas se desligou pouco tempo após sua saída.

Assim, o valor de cada comentário k de v_i para v_j decai exponencialmente quanto mais antigo ele se torna. O valor total $P(v_i, v_j)$ (3.1) 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 (3.2). $W(v_i, v_j)$ 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

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



que v_j recebeu.

$$P(v_i, v_j) = \sum_{k=1}^{n} \frac{1}{\exp days(inter(v_i, v_j, k))} \tag{3.1}$$

$$W(v_i, v_j) = \frac{P(v_i, v_j)}{\sum_{k=1}^{n} P(v_j, v_k)} \tag{3.2}$$

3.3 ANÁLISE 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, características e comportamentos do grafo são compatíveis com o descrito na literatura e se tal contexto permite conclusões relevantes para os objetivos deste trabalho.

3.3.1 **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 são 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.

Tabela 2 – Projetos e principais características

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

3.3.1.1 *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

[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

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.

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

3.3.1.3 *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.

3.3.1.4 *Tensorflow*

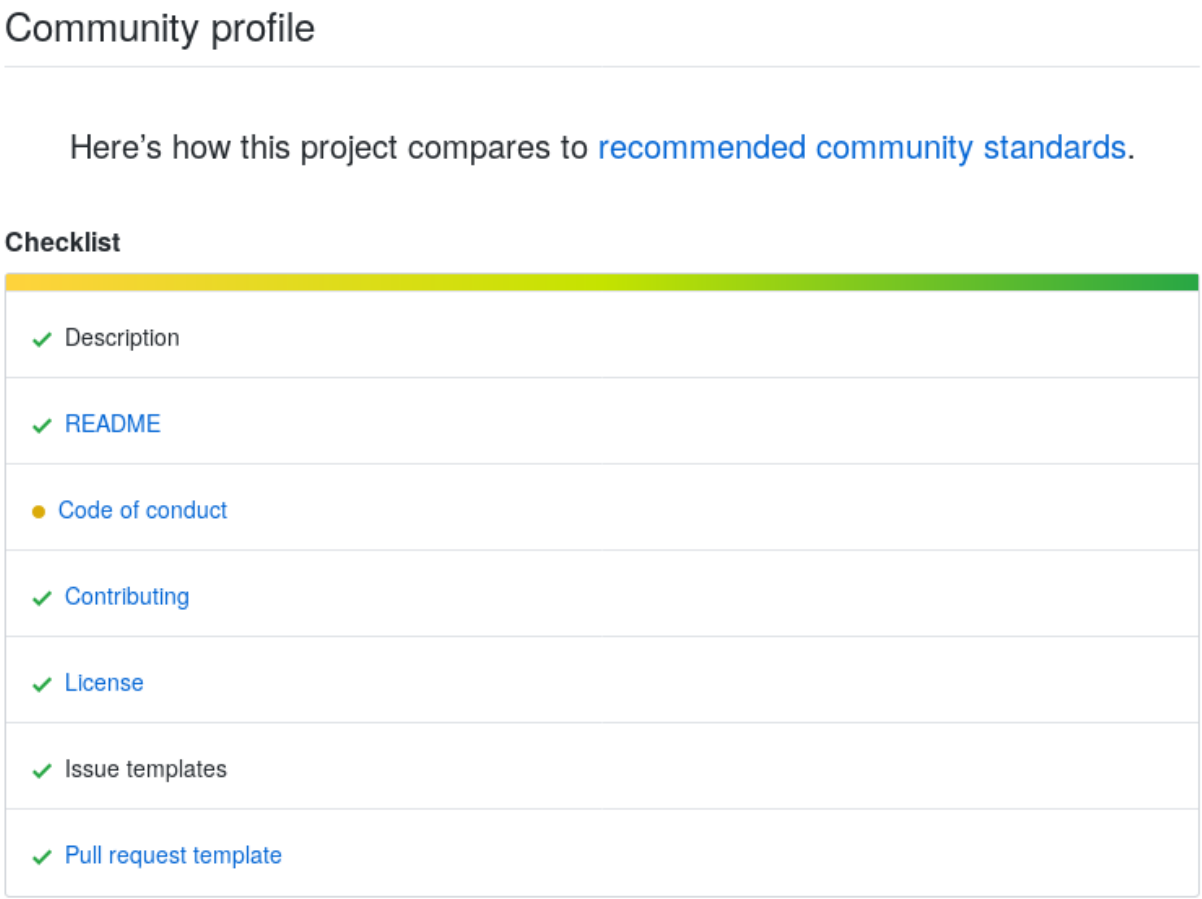
Tensorflow é o projeto mais popular dos analisados neste trabalho, além de ter o maior número de *forks* do GitHub e ser o quinto com mais contribuidores. A tecnologia é desenvolvida em C++ com interface em Python com o objetivo de dar suporte para técnicas de aprendizado de máquina. O processo de revisão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

3.3.2 **Extração dos Dados**

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.

[9] https://github.com/kubernetes/kubernetes
[10] https://github.com/kubernetes/community/blob/master/contributors/guide/owners.md
[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

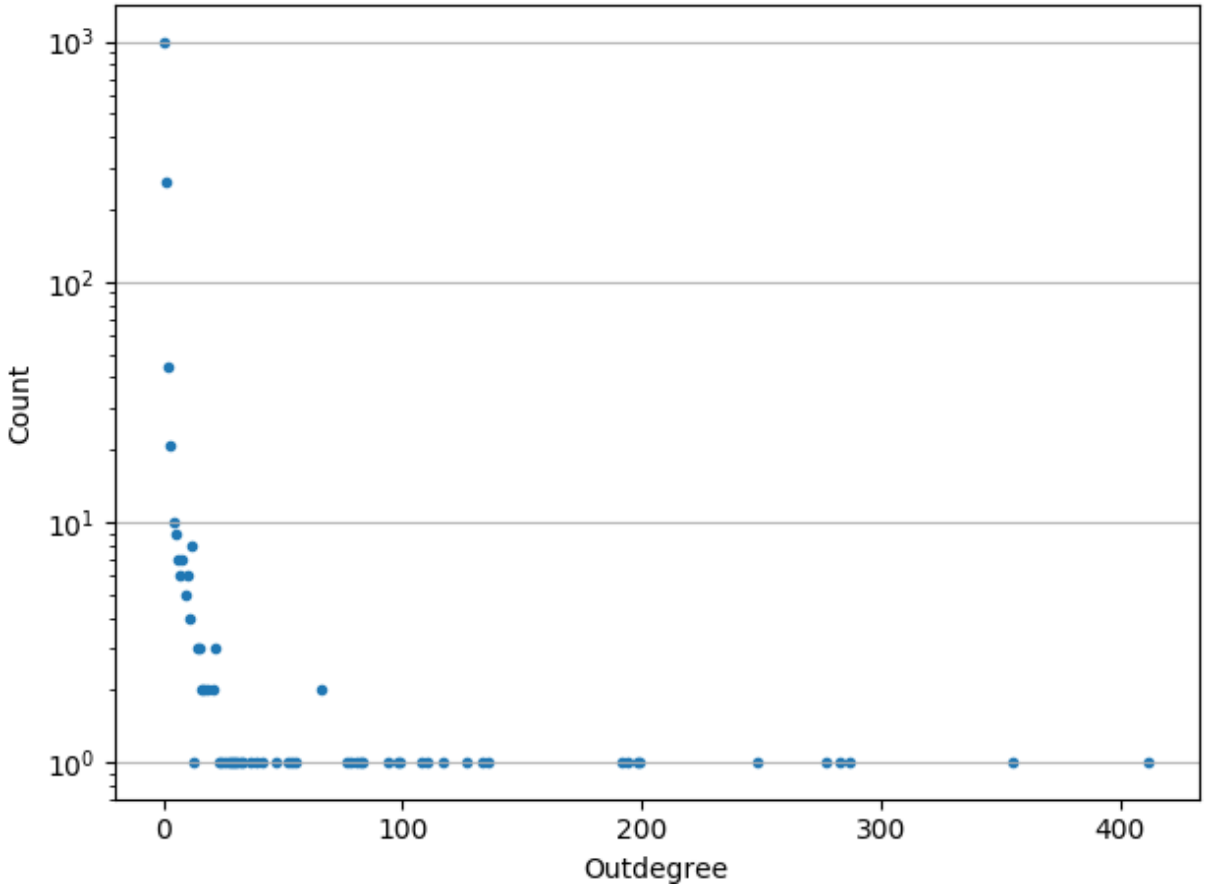


Ao calcular a distribuição de grau da rede, é possível entender como os indivíduos partilham a responsabilidade das revisões entre eles. No grafo direcionado, esse valor representa com quantos indivíduos um revisor atuou. Por exemplo no Node.js, como mostra a Figura 7, é possível observar que um pequeno 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 (CROSS; CROSS; PARKER, 2004). 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, 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

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

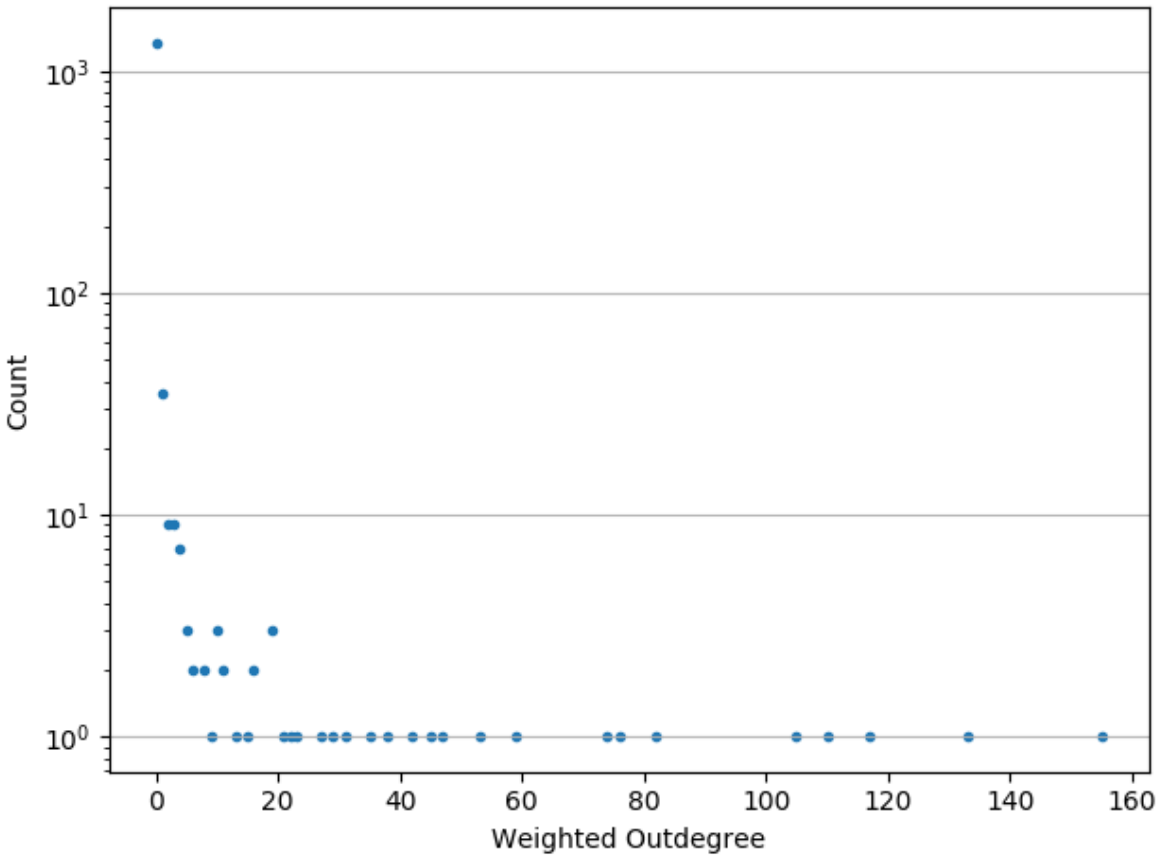


toda interação ocorre de alguma forma associada aos principais nós da rede. Por exemplo, no Symfony e no Node.js, existem apenas quatro componentes conexas em toda o grafo. Esta estrutura converge com reportado por trabalhos anteriores (BERGQUIST; LJUNGBERG, 2001). Estes especialistas que conduzem a maior parte do processo são podem ser responsáveis pelo projeto como um todo, módulos específicos (WIKI, 2018b), ou tecnologias/atribuições mais específicas (WIKI, 2018a). 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.

Uma das formas de se compreender a rede é avaliar a distribuição das interações nos *"pull requests"*. Quantos aos comentários de revisão, principal foco deste trabalho, é possível observar que de maneira geral poucos revisores participam do processo. São comentários ligados diretamente ao código e por isso mais técnicos, sendo alvo de um grupo reduzido de indivíduos. A Figura 10 mostra em formato de histograma a distribuição do número de revisores por *"pull request"*.

É possível observar em todos os projetos que a maior parte dos casos menos de três revisores participam do processo. Existem muitos casos onde não há nenhum tipo

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



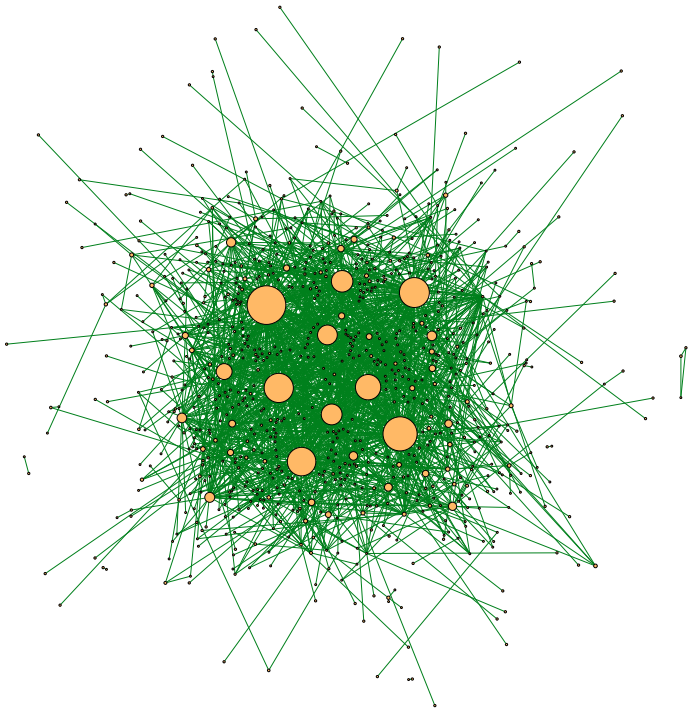
de interação de revisão, possivelmente em *"pull requests"* menores ou mais simples. Este comportamento se mostra menos acentuado em alguns projetos quanto o assunto são comentários gerais, de discussões que não necessariamente são técnicas durante o processo. É o que demonstra a Figura 11b, análoga às anteriores mas contando o número de autores de comentários de discussão nos *"pull requests"*.

3.4 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 (MENG et al., 2014). 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.

Figura 9 – Representação gráfica da rede do Tensorflow

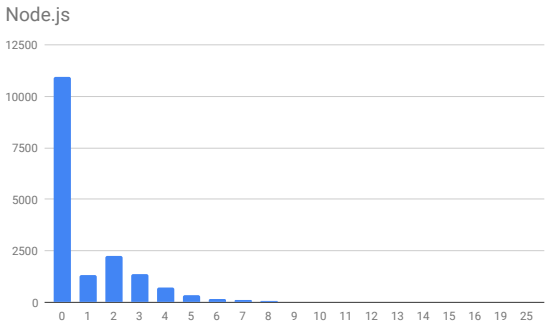


Estas exigências se dão devido a natureza da organização e forma de trabalho dos projetos *open source*. Especialmente os especialistas acabam por participar de diversas frentes de trabalho, devido à mão de obra reduzida. Além disso, a quantidade de grupos pode variar drasticamente de um projeto para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

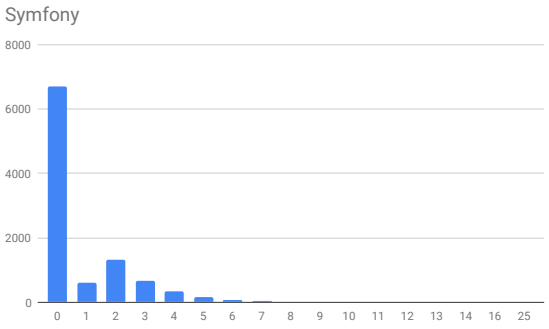
3.4.1 **NetSCAN**

NetSCAN (HORTA et al., 2018) é um algoritmo de clusterização baseado em densidade, que estende o conhecido DBSCAN (ESTER et al., 1996). 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 indivíduos *core* de cada cluster são identificados, podendo estes também estarem

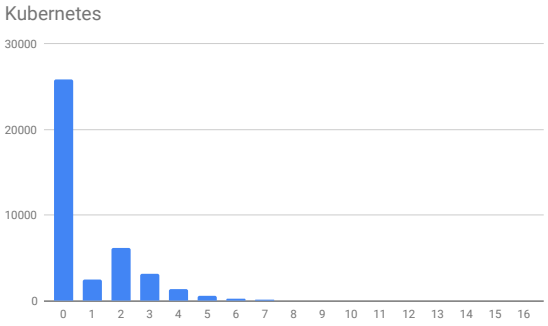
Figura 10 – Distribuição de autores de comentários de revisão nos projetos selecionados



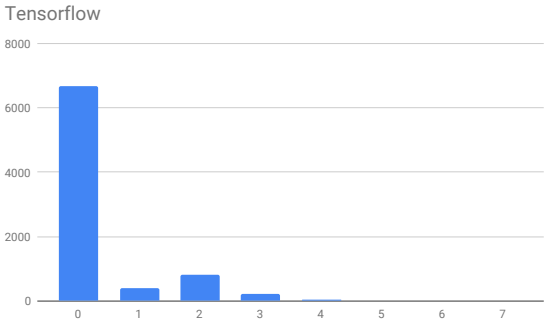
(a) Node.js



(b) Symfony



(c) Kubernetes



(d) Tensorflow

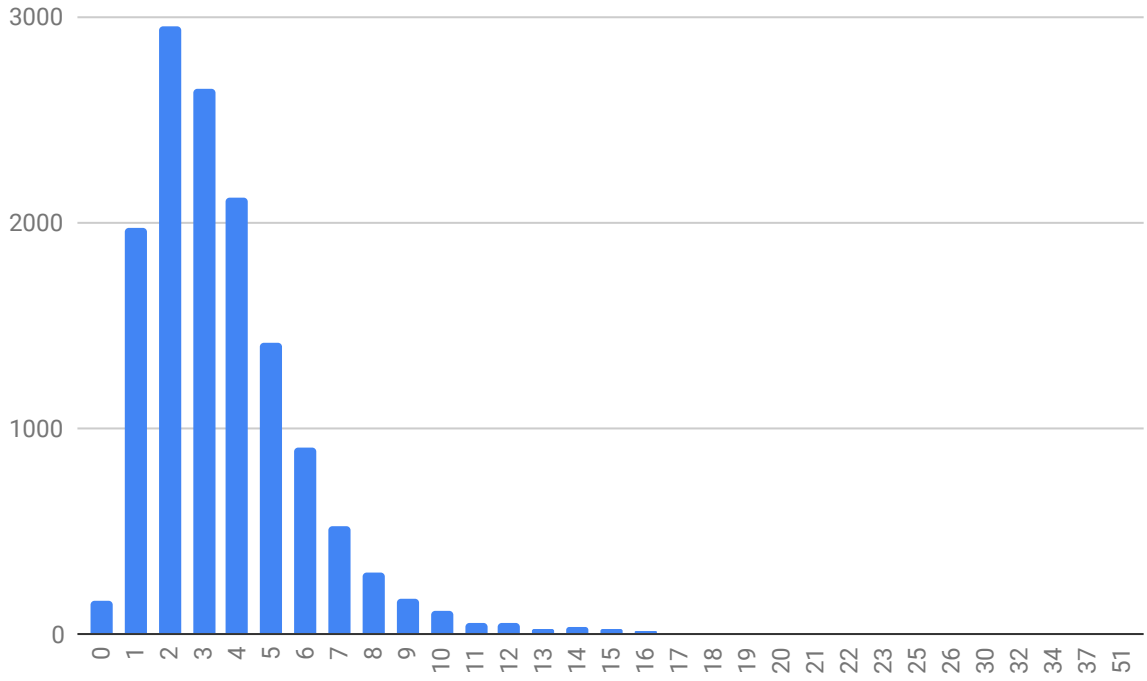
sobrepostos em outros silos.

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.

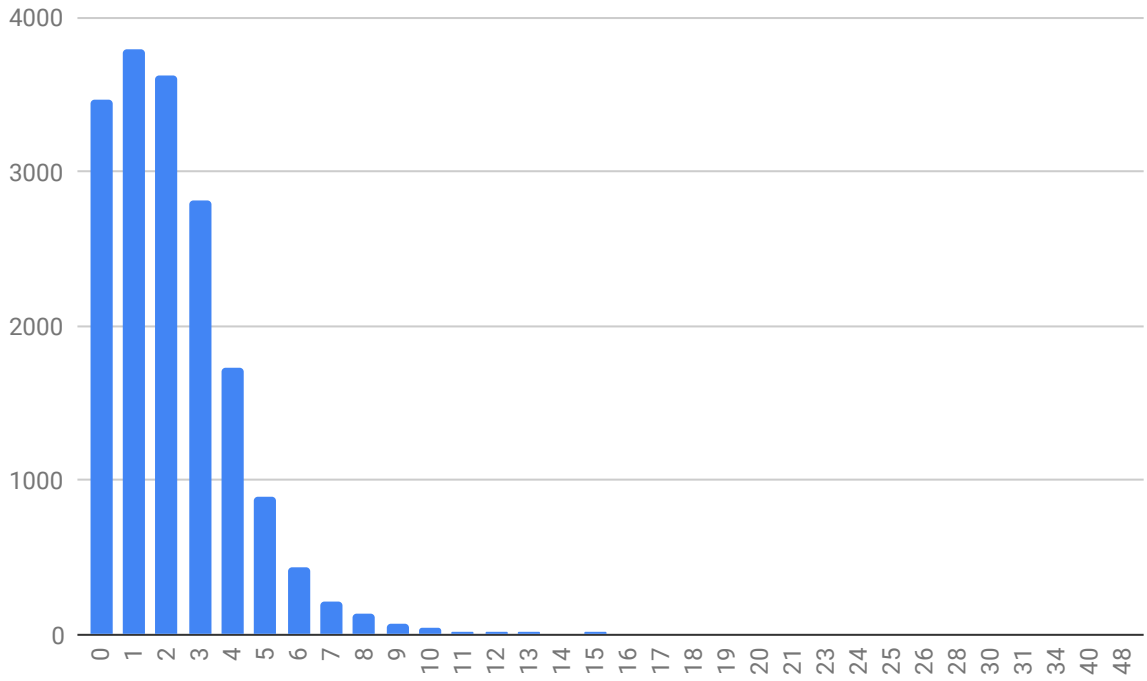
Para proporcionar a melhor escolha de parâmetros, foram testadas diferentes combinações em busca de otimizar a silhueta dos grupos (TAN; STEINBACH; KUMAR, 2005). Esta métrica usa a distância entre os nós contidos em um *cluster* para comparar a similaridade entre eles. A ideia é que a proximidade deles seja maior que em relação a membros externos dos grupos. O índice varia entre -1 e 1 onde valores mais altos indicam grupos mais coesos. (ALMEIDA et al., 2011). Na Figura 12 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.

Figura 11 – Distribuição autores de comentários de discussão em dois dos projetos selecionados



(a) Node.js



(b) Kubernetes

Figura 12 – Silhuetas do Node.js

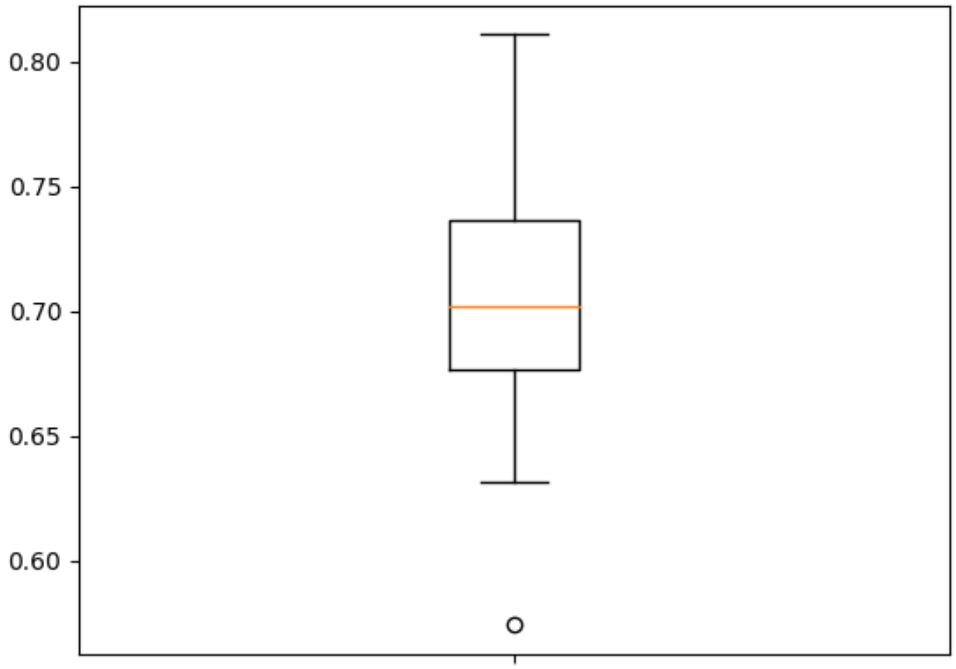


Tabela 3 – Valores otimizados pela silhueta de cada projeto

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

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.

3.4.2 **Execução**

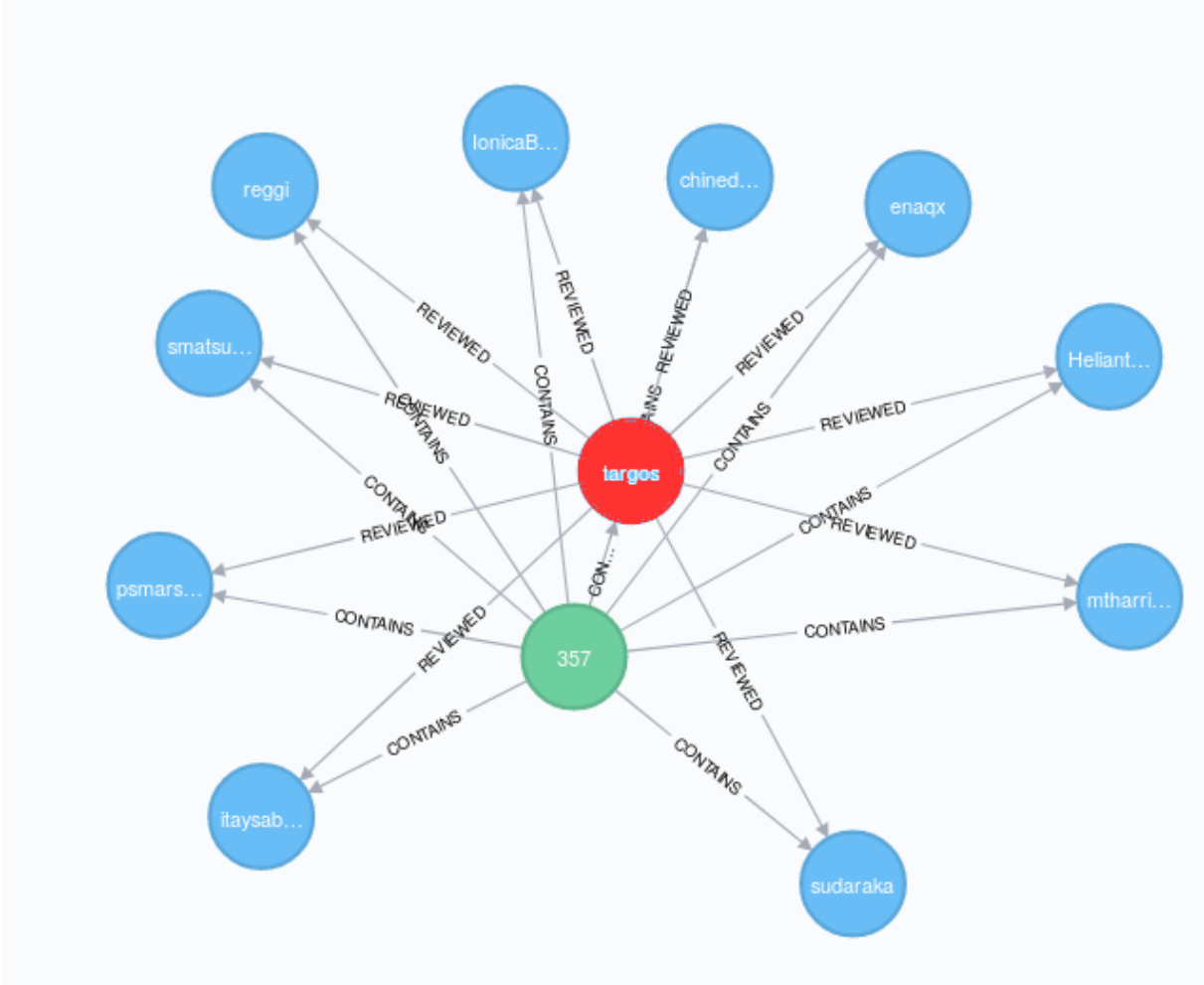
Os dados são automaticamente carregados para uma instância Neo4j e o algoritmo haje com os parâmetros debatidos nas seções anteriores. A Figura 13 ilustra um dos *clusters* encontrados.

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

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

Figura 13 – Representação de um dos *clusters* do Kubernetes



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 características que podem indicar maior efetividade da clusterização como base para algoritmos de detecção de especialistas e recomendação para revisão.

3.4.3 **Avaliação da clusterização**

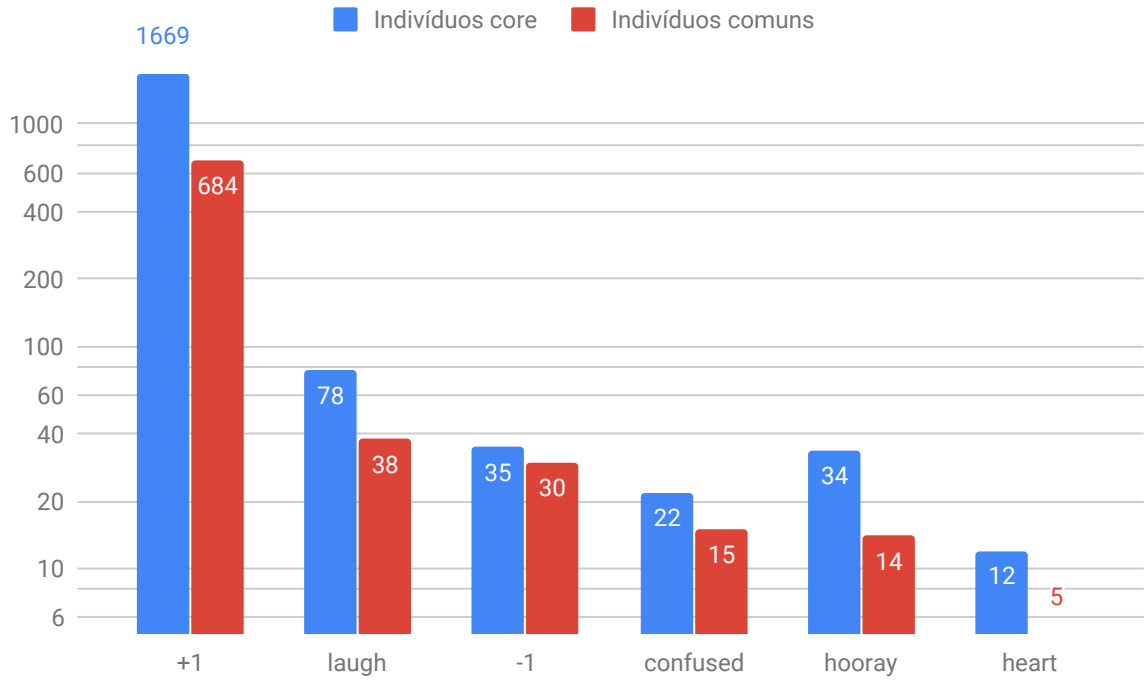
Duas formas de avaliação semântica foram aplicadas neste trabalho. A primeira diz respeito à pertinência dos *cores* encontrados enquanto desenvolvedores influentes e responsáveis pelo projeto. A segunda verifica a diferença entre as atividades e os principais focos dos grupos destacados.

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 14. 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 14 – Comparação das *reactions* recebidas por usuários *cores* e comuns no Symfony



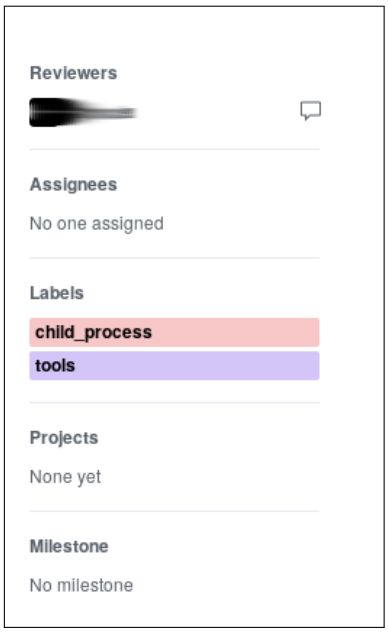
Quanto à responsabilidade dos desenvolvedores *core* no projeto, é possível analisar se eles possuem permissões de aprovar *"pull requests"* e de outras tarefas administrativas no projeto. A Tabela 4 mostra os resultados da proporção de *cores* detectados que efetivamente são dotados desse nível de permissão. 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.

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 15.

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

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

Figura 15 – *Labels* associadas a um *"pull request"* no Node.js

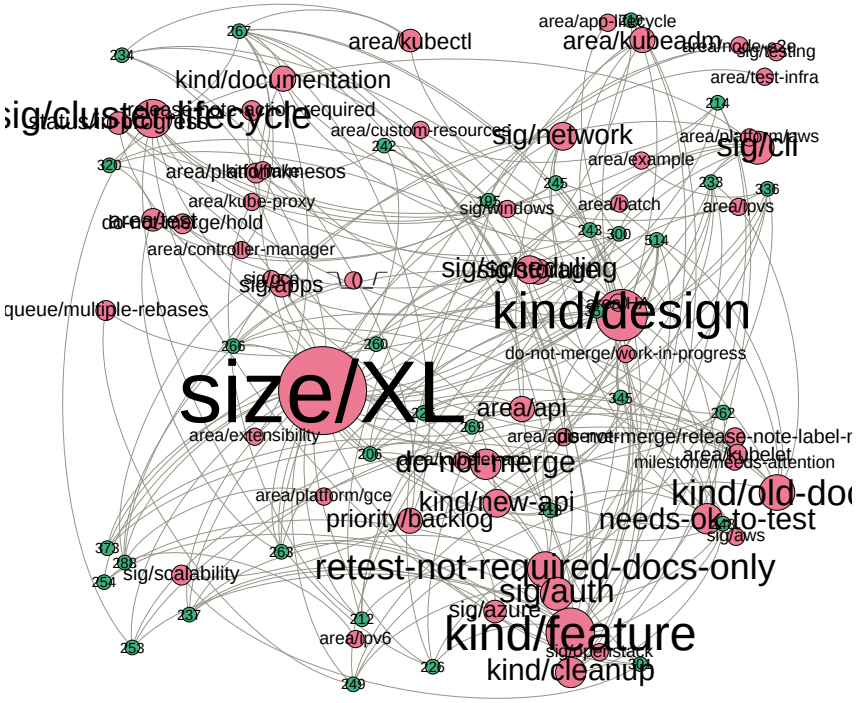


Para avaliar tal distinção, foram comparados quais *clusters* trabalham com quais *Labels*. Foram selecionadas as 20 *Labels* mais utilizadas de cada projeto e quais clusters trabalham com tais categorias entre suas 20 mais utilizadas. O resultado pode ser representado como u grafo na Figura 16. Em todos 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 16, 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 áreas 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

Figura 16 – *Clusters* associados a cada *"Label"* no Kubernetes



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 (BOSU; CARVER, 2014; KOVALENKO; BACCHELLI, 2018).

3.5 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.

3.5.1 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.

3.5.2 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
```

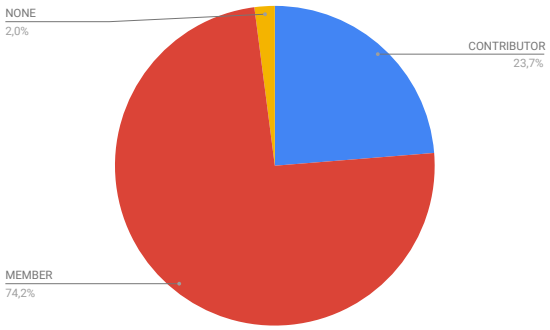
O principal objetivo deste algoritmo é retornar k indivíduos para revisão, que sejam *cores* de clusters que o autor do *"pull request"*. Como muitas vezes o universo destes nós é pequeno, para dar suporte à valores de k mais altos o algoritmo complementa a lista com cores aleatórios, assim como o método *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 interseção 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.

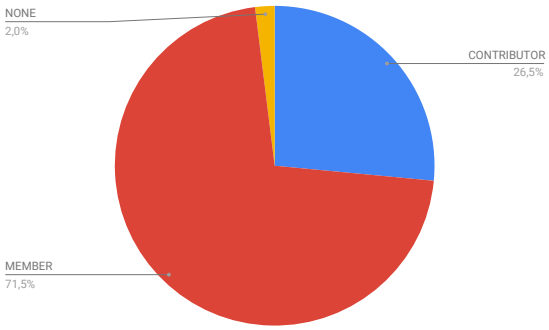
3.5.3 **LabelPartners**

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. Isso porque é possível observar que apesar de membros com acessos de administrador e centrais para os projetos são responsáveis por grande parte interações nas revisões, mas não sua composição absoluta. A Figura 17 mostra os *contributors* chegam a ser responsáveis por 38% dos comentários, e por isso podem ter plena capacidade técnica de serem revisores adequados, e por isso podem também ser recomendados. Segundo a documentação oficial[15], os *contributors* já fizeram alguma contribuição ao projeto enquanto os classificados como *none* nunca interagiram antes com o repositório. Os *members* e *collaborators* são pessoas convidadas ou participantes das organizações responsáveis pelo projeto, contando com direitos administrativos, como por exemplo aprovar os *"pull requests"*.

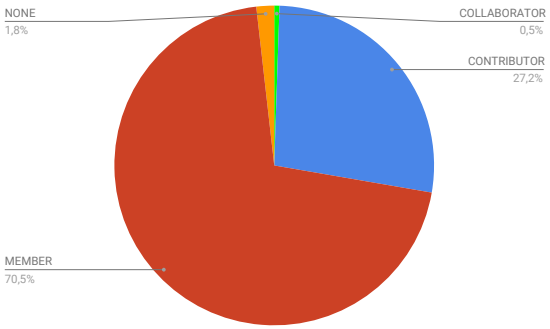
Figura 17 – Categorias de autores responsáveis pelas interações nos projetos observados.



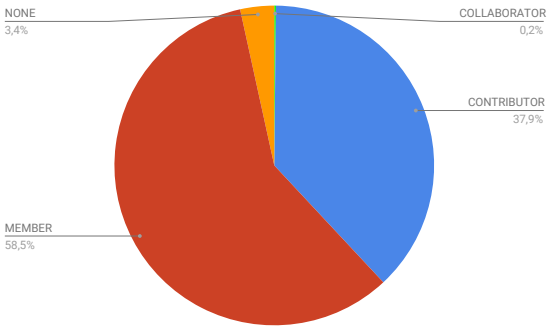
(a) Node.js



(b) Symfony



(c) Kubernetes



(d) Tensorflow

Assim, ao invés de indicar apenas os *cores*, este método indica indivíduos na seguinte ordem de prioridade:

[15] https://developer.github.com/v4/enum/commentauthorassociation/

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 3.1 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 interseção entre os grupos já indicados e os *cores*, de maneira análoga ao executado no algoritmo 2.

3.5.4 **Discussão dos Resultados**

Os métodos aqui propostos foram desenvolvidos de acordo com recomendações de trabalhos pregressos em relação ao uso de dados de colaboração e análises de redes sociais (SNA). Como base, tivemos os avanços e resultados promissores obtidos nesta área em trabalhos anteriores (FU et al., 2017; XIA et al., 2017; SCHETTINO et al., 2019a). Tais diretrizes condizem com os objetivos do trabalho de indicar revisores que aumentem a colaboração no contexto alvo. As abordagens foram concebidas como complementares, com o objetivo de cobrir particularidades dos projetos, mesmo que inseridos no mesmo contexto. Para avaliação dos métodos, foi desenvolvida uma plataforma para aferir as métricas clássicas da área e especialmente os aspectos que tratam sobre a qualidade da revisão de código, cobertos na seção 2.6.

4 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. O objetivo é deixar a avaliação dos métodos propostos neste trabalho mais diretos e reprodutíveis, de forma a facilitar a inclusão de novas abordagens e inserção dos modelos em contextos distintos. Levando em consideração o workflow atual 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 características técnicas da ferramenta que suportam os requisitos funcionais, como por exemplo disponibilidade, portabilidade e manutenibilidade. Dentre tais aspectos, pode-se destacar como mais relevantes:

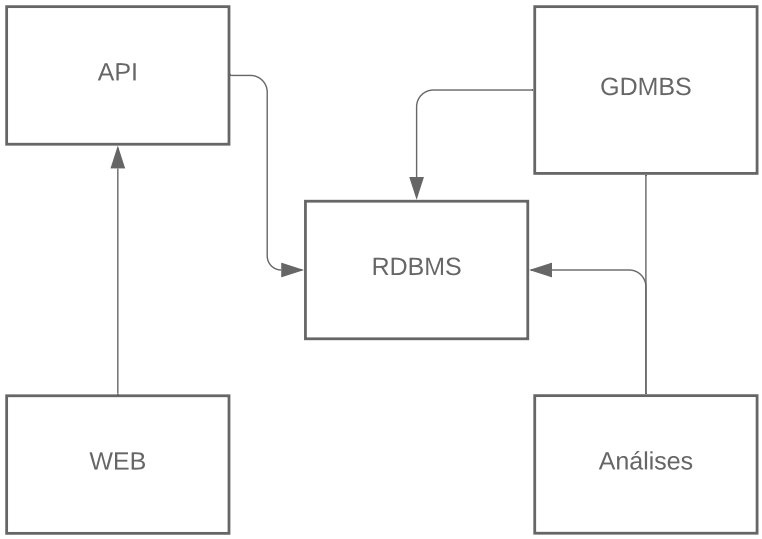
- Interoperabilidade
  - A ferramenta proposta deve-se comunicar o repositório de código fonte e ser compatível diferentes IDEs .
- Manutenibilidade
  - Existem diversas ferramentas de mercado para revisão, além de ferramentas de repositório de código fonte. Por isso a ferramenta deve ser modelada para 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 estendido por pesquisas futuras. A próxima seção apresenta como tais aspectos foram levados em consideração.

4.1 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, confiabilidade e possibilidade de extensão nos experimentos (FREIRE; BONNET; SHASHA, 2012). 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 18.

Figura 18 – 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 4.2 e 4.3. Estes componentes são orquestrados através do docker-

[1] https://www.docker.com/

compose[2], que configura os parâmetros e acessos necessários para o funcionamento do sistema sem que o usuário precise realizar nenhuma ação extra.

Com um conjunto grande de dependências, tecnologias e minúcias que estão envolvidas neste tipo de experimento, o código e artefatos descritivos não são suficientes para alcançar níveis adequados de reprodutibilidade (INCE; HATTON; GRAHAM-CUMMING, 2012). 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 (BOETTIGER, 2015).

De acordo com Sinha et al. (SINHA; SUDHISH, 2016), 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 19 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 Tabela 5 mostra o detalhamento de cada aspecto que contribui para que esta característica seja evidenciada.

Figura 19 – Espectro de Reprodutibilidade (SINHA; SUDHISH, 2016)

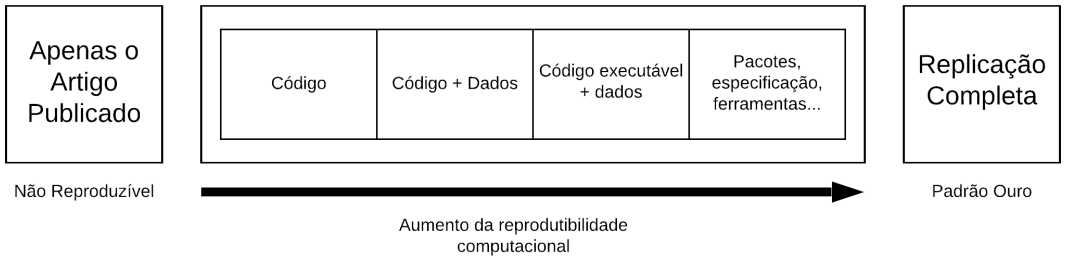


Tabela 5 – Escala de Reprodutibilidade (SINHA; SUDHISH, 2016)

	Level 0	Level 1	Level 2	Level 3
Dados / Evidências	Não Disponibilizado	Estatísticas Descritivas	Dados Compartilhados	Compartilhamento não repudiável
Modelos e Parâmetros	Não Disponibilizado	Referência aos mecanismos	Descrição de parâmetros e opções	Documentação, robustez, tratamento de entradas
Código Fonte	Não Disponibilizado	Proprietário	Open Source	Livre e de licença não restritiva
Sistema Requerido	Não Disponibilizado	Divulgado	Virtualizado	Pacotes disponibilizados em nuvem
Artefatos de Apresentação	Não Disponibilizado	Texto	Programação Estática (Latex)	Escrita colaborativa e artefatos automatizados height

4.1.1 **Dados - Evidências Primárias/Secundárias**

Esta categoria analisa a disponibilidade dos dados utilizados para futuros pesquisadores. Pode variar de simplesmente não disponibilizado 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)

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

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.

4.1.2 **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 encapsuladas através de comandos executáveis dentro de um contâiner Docker. Por conseguinte, o trabalho pode ser classificado como nível 3 nesta categoria

4.1.3 **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 aberto sob a licença MIT (OPEN SOURCE INITIATIVE AND OTHERS, 2000), categorizando assim o nível 3.

4.1.4 **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 infraestrutura 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.

4.1.5 **Artefatos de apresentação**

Todos os artefatos de apresentação, tais como artigos, dissertação e posters oriundos deste trabalho foram construídos com o LaTeX(LAMPORT, 1994) e o código fonte

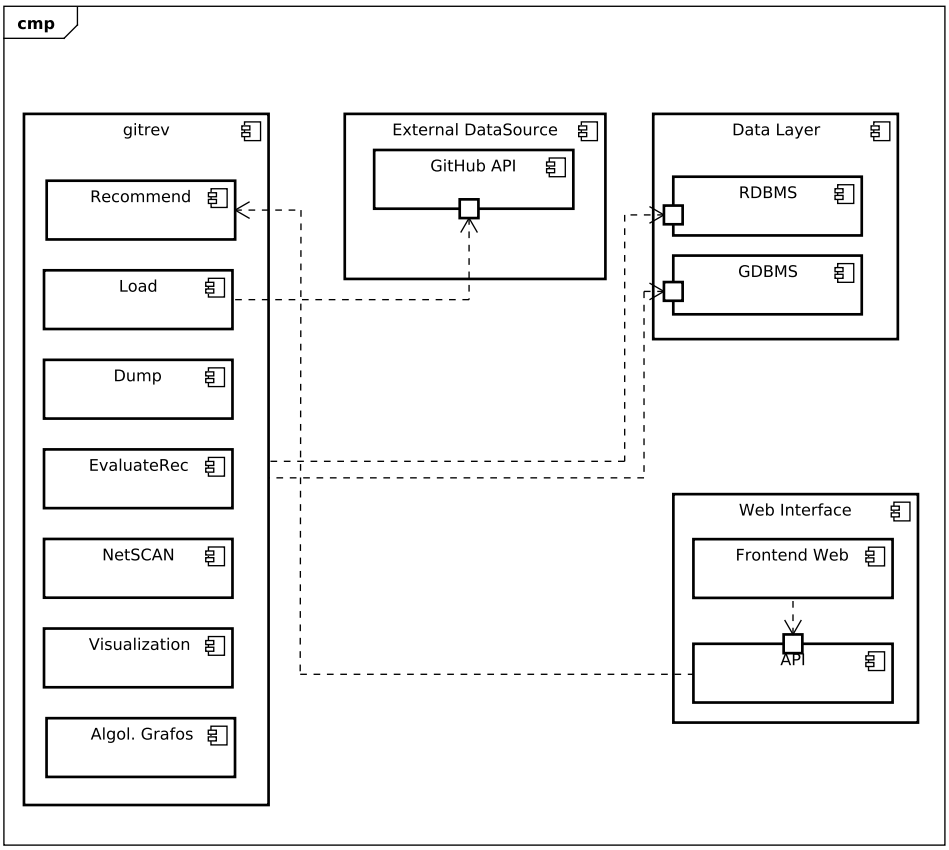
disponibilizado. Gráficos e tabelas foram produzidos diretamente através de scripts e/ou com ferramentas *Open Source*, como o Gephi[3]. Nesta categoria o estudo também se caracteriza como nível 3.

De acordo com a escala proposta (SINHA; SUDHISH, 2016) 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.

4.2 ARQUITETURA

A ferramenta proposta é estruturada em diferentes componentes, de acordo com sua função e tecnologias escolhidas. A Figura 20 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 20 – Diagrama de Componentes da Ferramenta



[3] https://gephi.org/

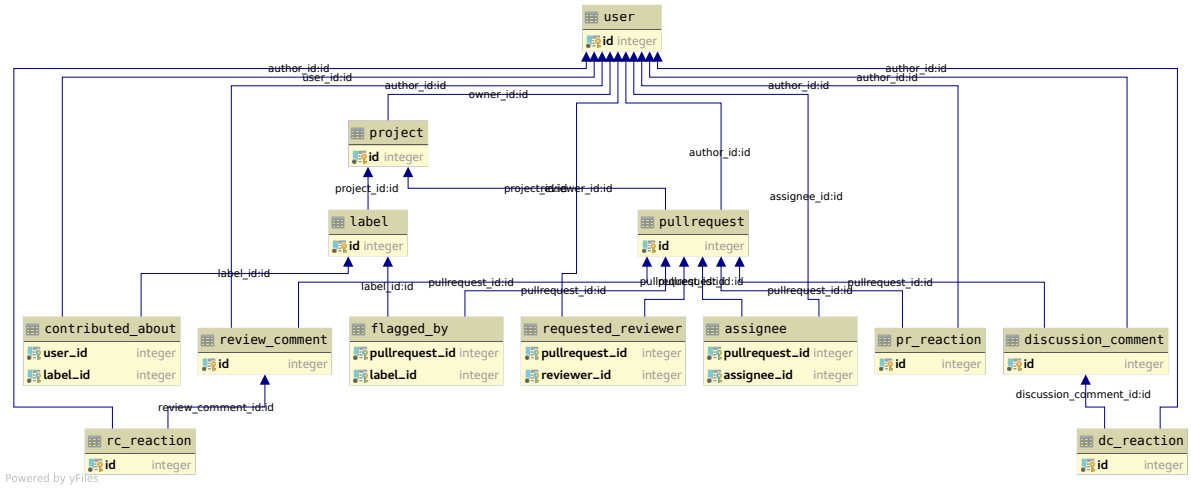
4.2.1 External DataSource

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 para integração foi a v3, que implementa o padrão RESTful (FIELDING; TAYLOR, 2002). 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.

4.2.2 Data Layer

Neste componentes os dados normalizados oriundos do GitHub são persistidos e tidos como base para as outras funcionalidades existentes. O Diagrama de Entidade Relacionamento (DER) na Figura 21 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 (GILBERT; LYNCH, 2002) de atomicidade, consistência, isolamento e durabilidade.

Figura 21 – Diagrama de Entidade Relacionamento (DER)



Na Figura 21 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

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

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 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 instanciada 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.

4.2.3 **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, ecossistema e flexibilidade da linguagem. 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 algoritmo 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:

4.2.3.1 *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

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

para evitar que informações inconsistentes sejam armazenadas (como por exemplo um comentário relacionado a um *"pull request"* que não existe).

4.2.3.2 *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.

4.2.3.3 *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.

4.2.3.4 *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 3.2, 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 3.4.1. 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 *Algoritmos de Grafos* onde consta um conjunto de métodos para avaliar as redes e os relacionamentos encontrados.

Na camada *Visualization* é possível encontrar os algoritmos que se apoiam nas

ferramentas matplotlib[9] e numpy[10] para gerar gráficos de distribuição de grau, *boxplot* das silhuetas entre outros.

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].

4.2.4 **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 22. 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.

4.3 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 23 descreve o processo.

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 algoritmos de recomendação 1 e 2. Os parâmetros utilizados podem ser obtidos através da otimização,

[9] https://matplotlib.org/index.html
[10] https://www.numpy.org/
[11] https://www.scipy.org/
[12] http://flask.pocoo.org/
[13] https://www.nginx.com/
[14] https://vuejs.org/

Figura 22 – Funcionalidade de recomendação no projeto Kubernetes

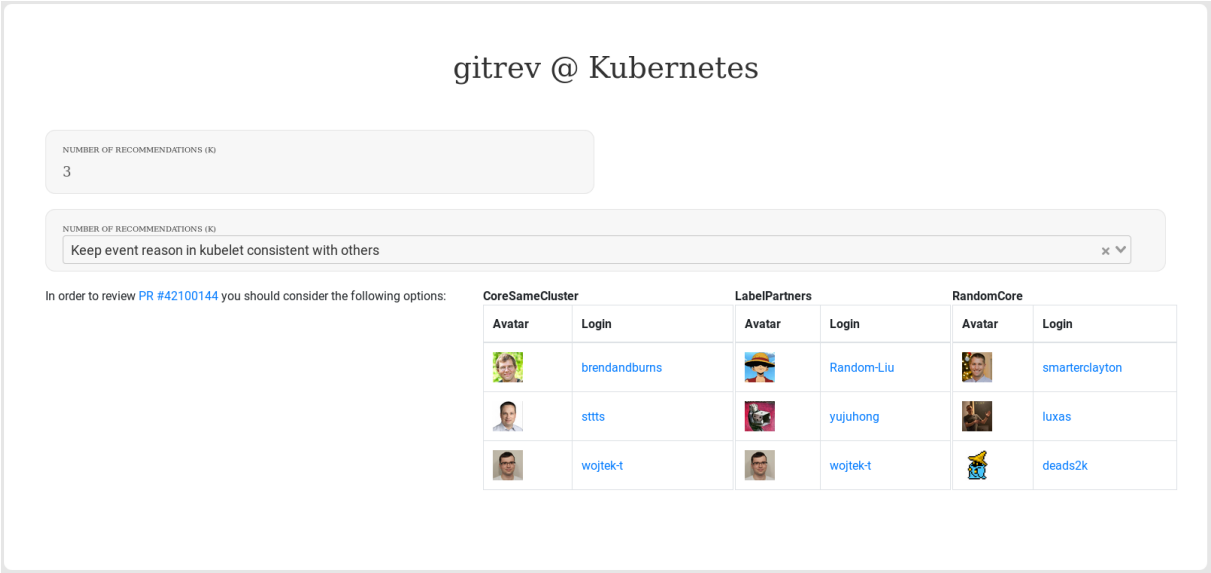
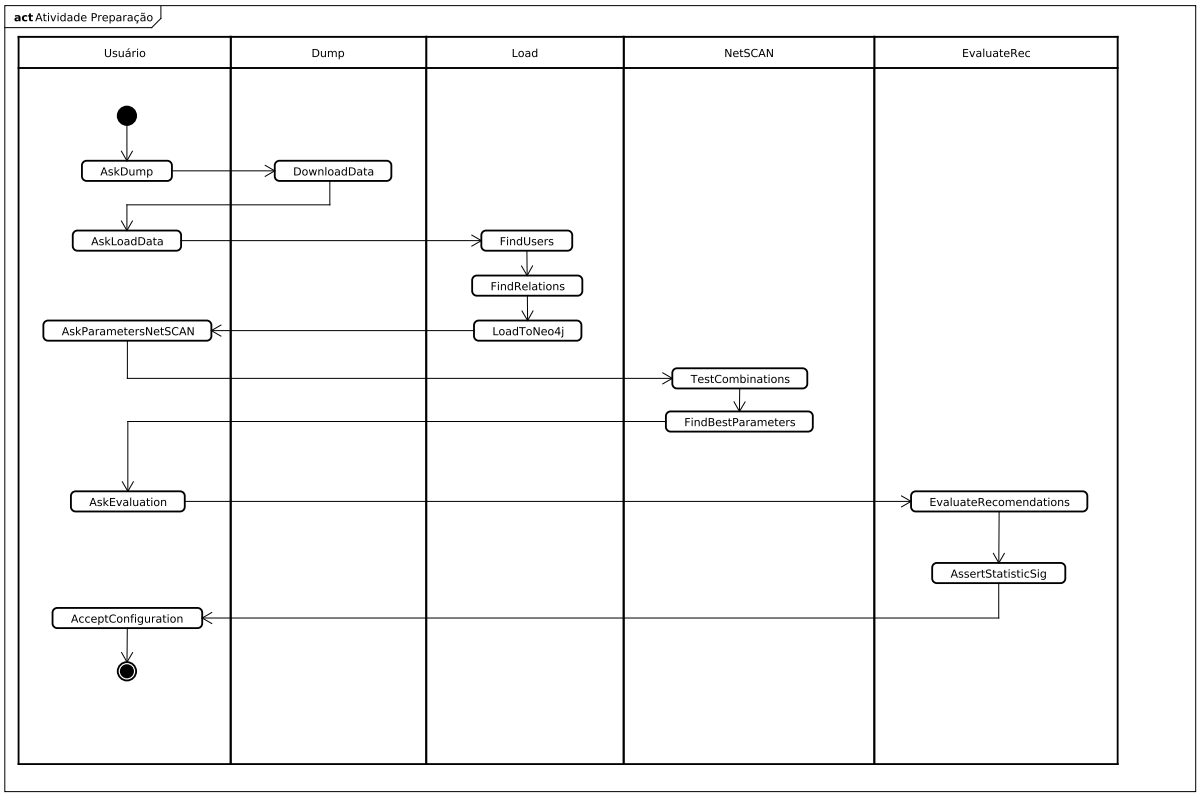


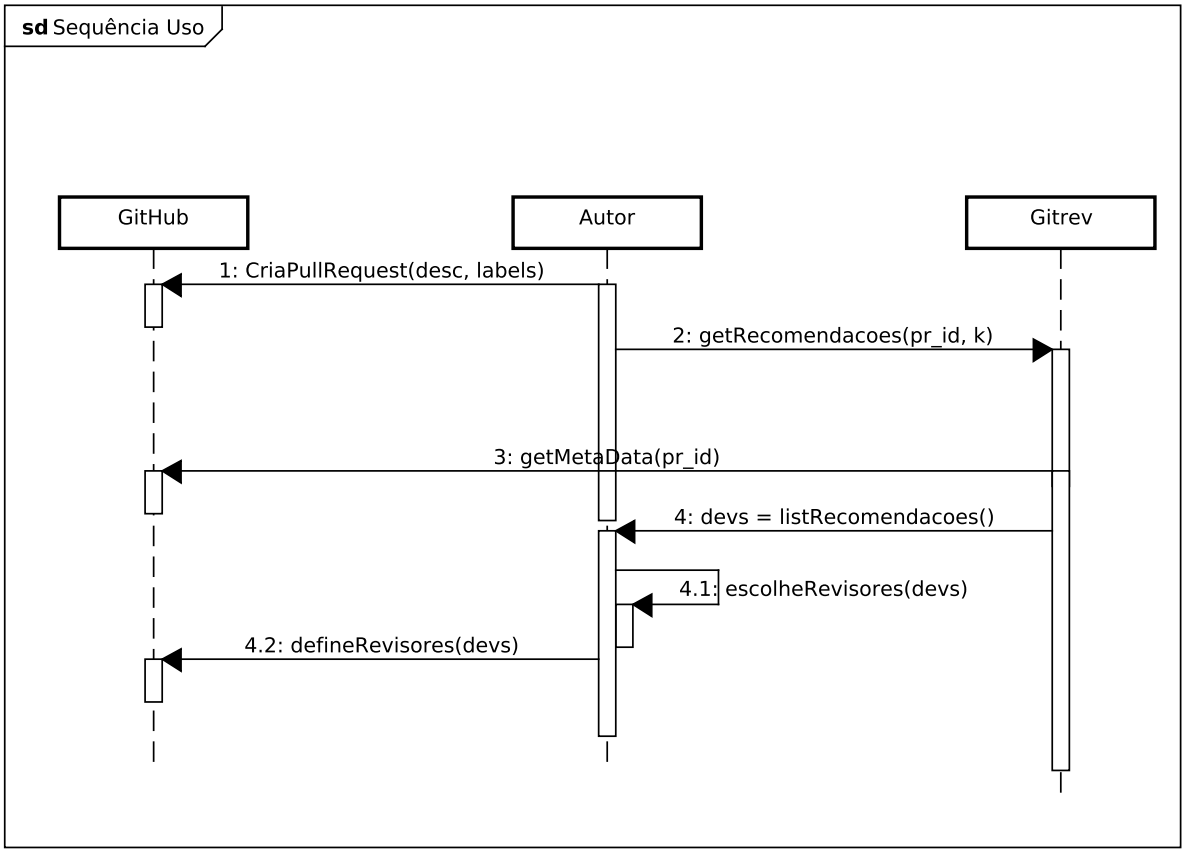
Figura 23 – Diagrama Atividades - Preparação



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 *pull based* (GOUSIOS; PINZGER; DEURSEN, 2014) introduzido na seção 2.1.1. A Figura 24 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 24 – 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.

5 AVALIAÇÃO DA SOLUÇÃO

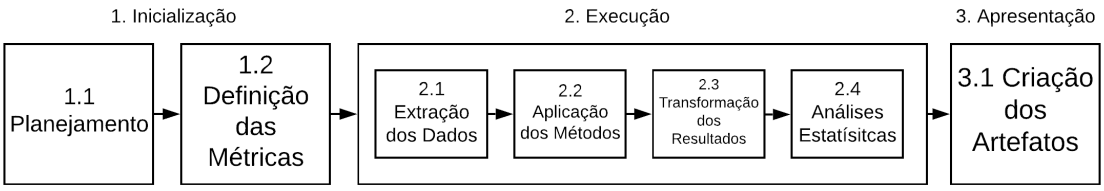
Avaliar métodos de recomendação pode ser um desafio em projetos onde não se há o controle do fluxo de trabalho ou influência para solicitar a utilização e um retorno sobre o impacto da ferramenta. Nos casos onde não há esse tipo de gerência sob o comportamento dos objetos de estudo, categoria a qual o presente estudo se inclui, a recomendação é utilizar os princípios da pesquisa histórica para avaliar a proposta (ROBERT, 1994). Nessa abordagem, utiliza-se a série histórica para procurar características que parecem estar relacionadas a comportamentos específicos dos usuários.

O foco da avaliação se encontra nos métodos de recomendação, sem dependência da ferramenta apresentada. Esta abordagem desvinculada permite a utilização de *datasets* maiores e maior abrangência entre os projetos, uma vez que não obriga a utilização da ferramenta por parte dos indivíduos envolvidos. Além disso, a avaliação dos métodos fica mais direta, não se confundindo com as métricas de avaliação do software.

5.1 PROTOCOLO

Para avaliar o desempenho dos métodos propostos, definiu-se um protocolo inspirado em boas práticas de engenharia experimental de software (WOHLIN et al., 2012) e na pesquisa histórica (ROBERT, 1994). Assim é possível mitigar riscos relacionados à validade dos resultados e fomentar melhorias nas métricas utilizadas. A Figura 25 ilustra cada um dos passos.

Figura 25 – Protocolo de Avaliação



Os passos são divididos em três categorias: Inicialização, Execução e Apresentação. Cada uma das categorias é apresenta das seções subsequentes.

5.1.1 **Inicialização**

O protocolo de avaliação se inicia com a elucidação da motivação de objetivos da abordagem. A motivação se alia à hipótese do presente trabalho, ao investigar se existe ganho na colaboração ao se utilizar os métodos propostos. O objetivo pode ser formalizado através do GQM (BASILI; WEISS, 1984) na seguinte forma:

> "Analisar os métodos de recomendação propostos para avaliar os respectivos desempenhos sobre a colaboração no desenvolvimento distribuído de software do ponto de vista dos participantes dos projetos selecionados."

O GQM personifica o objetivo da avaliação e ajuda a definir os próximos passos, como o formato e as métricas utilizadas.

Na avaliação em questão leva-se em consideração os casos em que os métodos propostos indicaram os mesmos desenvolvedores que efetivamente fizeram a revisão no passado. Assim, apesar de não existir contato direto do desenvolvedor com a ferramenta, é possível observar qual comportamento dos revisores que seriam indicados em relação aos que não seriam.

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(SCHETTINO et al., 2019a). São a porcentagem de precisão e acerto, como explicado na seção 2.5, e doravante referenciadas como “métricas de proximidade”. Já as métricas do segundo grupo são baseadas nos estudos de Bosu et al. (BOSU; GREILER; BIRD, 2015) e Rahman et al. (RAHMAN; ROY; KULA, 2017) detalhadas na seção 2.6 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.

5.1.2 **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 algoritmos de recomendação. É referenciado pela variável *num_comments*.

5.1.3 **Número de Reações**

Mede o número médio de reações (os populares *emojis*) recebidas 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 (BOSU; GREILER; BIRD, 2015). Não são computados comentários do próprio autor do *“pull request”*, já que esse também não é considerado para os algoritmos de recomendação. É referenciado pela variável *num_reactions*.

5.1.4 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 (BOSU; GREILER; BIRD, 2015). Neste caso as respostas do autor do *"pull request"* são contabilizadas. É referenciado pela variável *num_replies*.

5.1.5 Tamanho dos comentários

O tamanho dos comentários tem relação a relevância (RAHMAN; ROY; KULA, 2017). Não são computados comentários do próprio autor do *"pull request"*, já que esse também não é considerado para os algoritmos de recomendação. É referenciado pela variável *size_comments*, e equivale ao número de caracteres do corpo do comentário.

5.1.6 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 (WILBUR; SIROTKIN, 1992). A alta proporção de palavras que não se encaixam nesta descrição podem ser um sinal de um texto com mais substância e significativo, estando associado a relevância do comentário ao processo de revisão (RAHMAN; ROY; KULA, 2017). É referenciado pela variável *rate_non_stop_words*. O cálculo é feito através da biblioteca de buscas textuais do PostgreSQL[1], especialmente com a função *to_tsvector*() que converte o corpo do comentário para um array de palavras relevantes.

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.

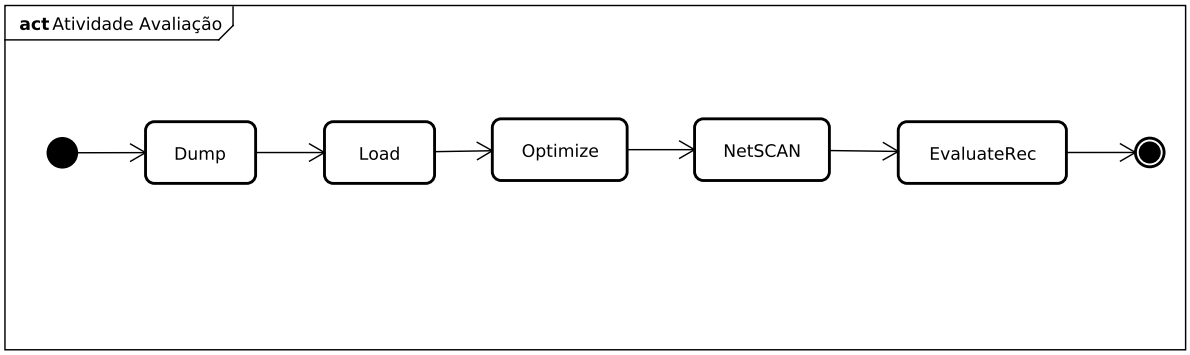
5.1.7 Execução

A execução dos métodos do ponto de vista do GitRev passa por três etapas distintas: a extração dos dados, a preparação das estruturas e a apreciação das métricas escolhidas e

[1] https://www.postgresql.org/docs/11/functions-textsearch.html

a significância estatística dos resultados. A Figura 26 mostra o procedimento.

Figura 26 – Execução GitRev para avaliação



As atividades estão todas contidas na ferramenta, com uma ordem de execução similar ao uso exemplificado na seção 4.3. A partir dos dados extraídos relativos aos 4 projetos selecionados na subseção 3.3.1, 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 discorrem das métricas utilizadas para avaliação e do processo de apreciação da significância estatística dos resultados obtidos.

5.1.8 **Significância estatística**

Para estabelecer juízo de valor sobre o experimento é necessário, primariamente, avaliar a relevância estatística sobre os dados coletados durante sua execução. A literatura acerca deste assunto indica possível subutilização de ferramentas adequadas na Engenharia de Software (MILLER et al., 1997), o que pode acarretar em trabalhos com resultados questionáveis e sem fundamentação sólida (DYBÅ; KAMPENES; SJØBERG, 2006). O objetivo é observar se existem diferenças significativas entre o desempenho de cada abordagem, ou se possíveis dissimilitudes são apenas frutos do acaso.

As análises dispostas nesta seção foram conduzidas de acordo com o trabalho de Araújo et al. (ARAúJO et al., 2006) e aplicado em Schettino et al. (SCHETTINO; ARAúJO, 2017), com objetivo de observar se são significantes as diferenças percebidas nas métricas avaliadas. O nível de significância (valor p ou *p-value*) escolhido foi 0,05, muito utilizado na Engenharia de Software e outras ciências naturais. Isso significa que os resultados serão aceitos como diferentes apenas a probabilidade disso ser verdade for

maior que 95%. O universo de avaliação aqui são as recomendações de cada método para o conjunto de *"pull requests"* escolhido. Nas métricas de proximidade, primeiro é feita a comparação entre os conjuntos recomendados e os efetivamente escolhidos, e depois com os resultados de cada método. Nas métricas de eficiência a comparação é feita entre as interações dos integrantes dos grupos recomendados e as interações dos que não foram. Depois os resultados totais podem ser apreciados frente às saídas dos outros métodos.

Levando em consideração o grande número de amostras analisadas neste 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 5.3 os resultados são apresentados em forma de tabela.

O primeiro passo é verificar a normalidade dos dados, ou seja, se eles se distribuem de forma normal, em torno de um valor médio e com a maior parte das amostras a poucos desvios-padrão de distância. Este teste é necessário para definir qual abordagem mais adequada para descobrir se há diferença significativa entre as amostras. Como pode-se observar que todas métricas contém mais de 30 amostras, aplica-se o teste de *Kolmogorov-Smirnov*. 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 27 mostra o resultado deste tipo de teste.

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

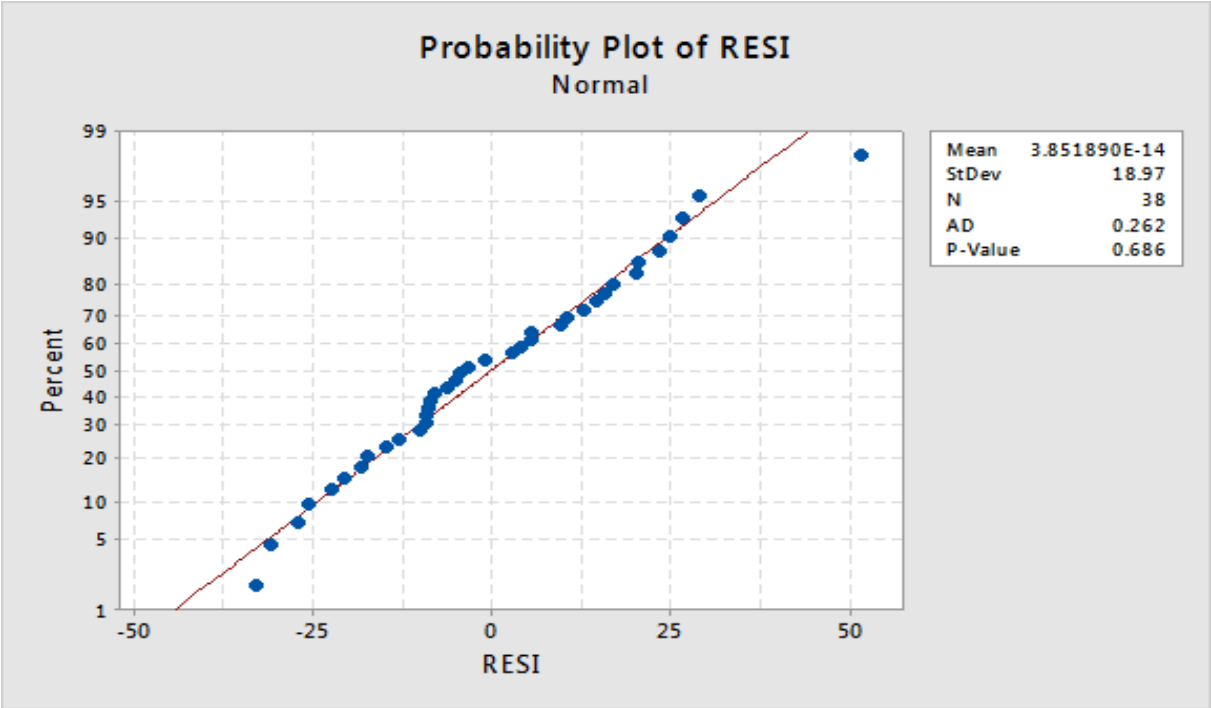
- H0 (hipótese nula): as amostras são homocedásticas.
- H1 (hipótese alternativa): as amostras não são homocedásticas;

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 28.

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;

Figura 27 – Gráfico representando amostra normal segundo o teste de *Kolmogorov-Smirnov* (PSU, 2017b)



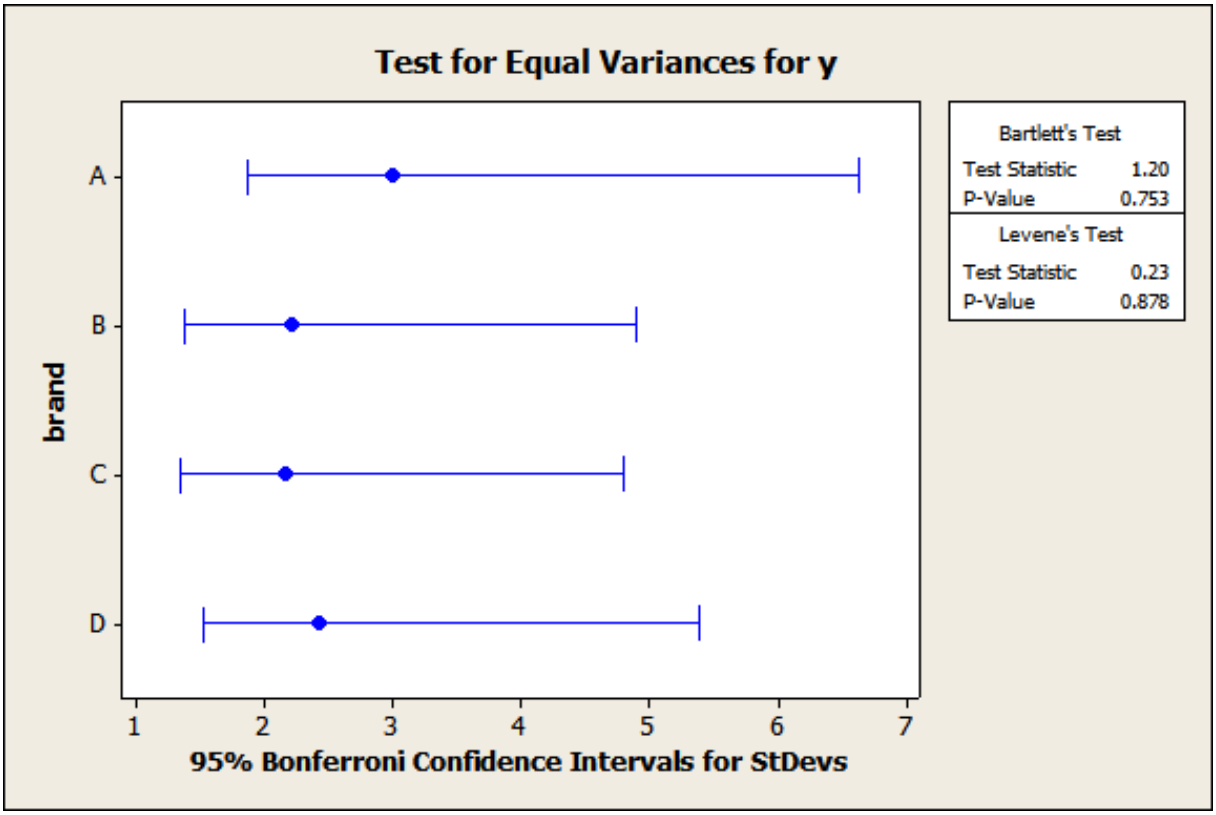


Figura 28 – Teste de *Levene* apontando amostra homocedástica (WANG, 2009)

- 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 29. 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 30.

5.1.9 **Apresentação**

Os resultados da execução dos métodos são filtrados pela sua significância estatística, uma vez que o número total de resultados dificultaria a leitura. O filtro permite o destaque dos resultados mais relevantes, sejam positivos ou negativos. A criação das tabelas LaTeX é automatizada de acordo com as saídas do GitRev. Os gráficos de linhas, barras e caixas são gerados com o auxílio do pelo próprio GitRev.

5.2 EXECUÇÃO

Para evitar enviesamento das análises, os conjuntos de teste e treinamento da rede são diferentes. Os dados utilizados para montagem da rede variam entre 01/01/2010 e 31/12/2017. Já o conjunto para avaliação está contido entre 01/01/2018 e 31/12/2018. A funcionalidade de avaliação dos resultados salva todas as métricas para cada revisão em cada projeto em arquivos .csv, facilitando análises exploratórias. Dois tipos de avaliações distintas são executadas: para as métricas de proximidade, a comparação é feita entre as três abordagens, agrupados dois a dois; para as métricas de eficiência a comparação é feita entre os grupos recomendados (r) e não recomendados (w), para cada método. Em ambos os casos, o valor de $k \in \{1, 3, 5, 10\}$. As figuras 31 e 32 mostram, respectivamente, exemplos das saídas para avaliações do primeiro e do segundo grupo.

5.3 APRESENTAÇÃO DOS RESULTADOS

A apresentação dos resultados é dividia em duas partes: nas métricas de proximidade e nas métricas de eficiência. Em cada uma delas há a apresentação dos dados brutos e a comparação entre os grupos de análise levando em consideração a significância estatísticas das saídas.

5.3.1 **Métricas de Proximidade**

As métricas de proximidade foram extraídas para cada projeto separadamente. Os resultados para o projeto Node.js estão nas tabelas 6 e 7. A precisão para $k \in \{1, 3\}$ mostram o método *LabelPartners* ligeiramente melhor, mas com a perda relativa para

Figura 29 – *Teste T* para asserção da significância estatística da amostra (PSU, 2017a)

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 30 – Teste de *Mann-Whitney* para asserção da significância estatística da amostra (FROST, 2013)

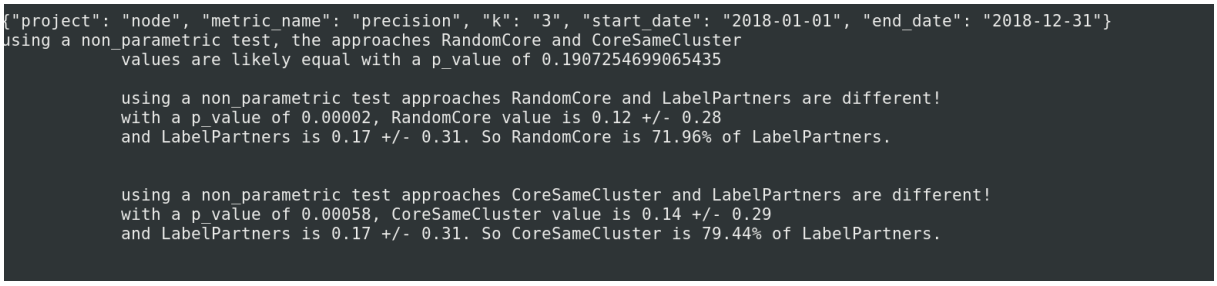
Mann-Whitney Test and CI: Female_Multitask, Male_Multitask

```
                    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 31 – Avaliação de Precision no Node.js

```
{"project": "symfony", "method_name": "LabelPartners", "k": "3", "start_date": "2018-01-01", "end_date": "2018-12-31"}

            using a non_parametric test approaches r_num_comments and w_num_comments are different!
            with a p_value of 0.01991, r_num_comments value is 2.67 +/- 2.61
            and w_num_comments is 2.23 +/- 1.95. So r_num_comments is 119.79% of w_num_comments.


            using a non_parametric test approaches r_num_reactions and w_num_reactions are different!
            with a p_value of 0.04138, r_num_reactions value is 0.28 +/- 0.70
            and w_num_reactions is 0.30 +/- 0.61. So r_num_reactions is 96.09% of w_num_reactions.

using a non_parametric test, the approaches r_num_replies and w_num_replies
            values are likely equal with a p_value of 0.15751060060622457
using a non_parametric test, the approaches r_size_comments and w_size_comments
            values are likely equal with a p_value of 0.1813919834948759

            using a non_parametric test approaches r_rate_non_stop_words and w_rate_non_stop_words are different!
            with a p_value of 0.01863, r_rate_non_stop_words value is 1.48 +/- 1.53
            and w_rate_non_stop_words is 1.20 +/- 1.11. So r_rate_non_stop_words is 123.41% of w_rate_non_stop_words.
```

Figura 32 – Avaliação das métricas de eficiência do método LabelPartners no Symfony



conjuntos de revisõs maiores. O mesmo acontece com o *hit*, apesar das diferenças para todos valores de k terem sido maiores.

Tabela 6 – Resultados da Precisão para o Node.js

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	4%	4%	6%
3	12%	14%	17%
5	21%	21%	23%
10	42%	41%	32%

Tabela 7 – Resultados do *hit* para o Node.js

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	9%	9%	13%
3	22%	23%	30%
5	36%	37%	37%
10	61%	59%	46%

Os resultados das métricas de proximidade para o Symfony foram melhores, para todos os métodos. Os resultados de *hit* apresentados na Tabela 9 mostram uma eficiência em até 81% dos casos. Os resultados de precisão na Tabela 8 mostram maior discrepância entre os valores mais baixos de k com o *LabelPartners* em vantagem substancial. O desempenho dos métodos *RandomCore* e *CoreSameCluster* foi similar em todos os casos estudados, superiores ou próximos ao terceiro método para ks mais altos.

No projeto Kubernetes é possível ver maior diferença entre os métodos *RandomCore* e *CoreSameCluster* para as médias da precisão (Tabela 10) e *hit*(Tabela 11). É também o único caso da superioridade do *LabelPartners* para $k \in \{5, 10\}$

Os resultados da precisão (Tabela 12) e *hit* (Tabela 13) do projeto Tensorflow mostram superioridade leve do *LabelPartners*, além de uma diferença maior entre os métodos *RandomCore* e *CoreSameCluster*, com o segundo sendo mais eficiente para valores de k mais baixos.

Tabela 8 – Resultados da Precisão para o Symfony

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	8%	9%	23%
3	21%	21%	40%
5	33%	35%	50%
10	65%	59%	62%

Tabela 9 – Resultados da *hit* para o Symfony

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	14%	17%	39%
3	35%	37%	59%
5	53%	55%	67%
10	81%	78%	76%

Tabela 10 – Resultados da Precisão para o Kubernetes

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	2%	5%	11%
3	6%	8%	21%
5	9%	13%	26%
10	18%	21%	32%

Tabela 11 – Resultados da *hit* para o Kubernetes

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	3%	7%	18%
3	10%	14%	21%
5	14%	20%	37%
10	28%	31%	42%

Tabela 12 – Resultados da Precisão para o Tensorflow

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	2%	4%	9%
3	11%	12%	15%
5	16%	18%	27%
10	35%	35%	42%

Tabela 13 – Resultados da *hit* para o Tensorflow

	RandomCore	**CoreSameCluster**	**LabelPartners**
1	2%	4%	10%
3	13%	13%	16%
5	18%	21%	29%
10	38%	39%	46%

Apesar de um domínio aparente do *LabelPartners* para valores de k mais baixos, outro comportamento pode ser observado nas figuras 33 e 34, que comparam a precisão e o *hit* dos métodos. Em dois casos a eficiência para k mais altos fica próxima para todos os métodos, enquanto em dois outros casos se observa uma escalabilidade maior dos métodos baseados em *cores*, linear, enquanto o *LabelPartners* não acompanha a mesma tendência. A aproximação dos métodos se dá em conjuntos de recomendações maiores pois quando não há mais *cores* do mesmo grupo ou parceiros de revisão suficientes para indicação, todos os métodos recorrem a lista completa de *cores* do repositório. Assim a tendência é que para ks mais altos o grupo recomendado seja muito parecido, levando ao resultado próximo. Além disso, como grande parte das revisões possui apenas um revisor (como mostrado na Figura 10), o valor das duas métricas tende a ficar mais próximo.

O maior desempenho dos algoritmos 1 e 2 para ks elevados pode ser justificada pela quantidade indicação de indivíduos centrais no projeto, que tem a responsabilidade (e nível de permissão) para aceitar o *"pull request"*. Apesar de potencializar os resultados nas métricas de proximidade, ainda é preciso avaliar se estes indivíduos tem a disponibilidade e as habilidades interpessoais para realizar boas revisões, segundo as métricas de eficiência escolhidas.

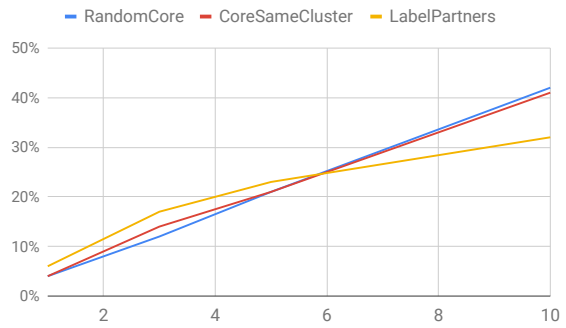
Apesar da análise visual dos comportamentos dos métodos perante a variação do conjunto recomendado, é necessário avaliar em quais casos houve diferença significativa entre as performances observadas. Como explicado na seção 5.1.8, os métodos são comparados dois a dois. São definidas as seguintes hipóteses para a precisão:

- H0 (hipótese nula): os métodos possuem precisão igual;
- H1 (hipótese alternativa): os métodos possuem precisões diferentes.

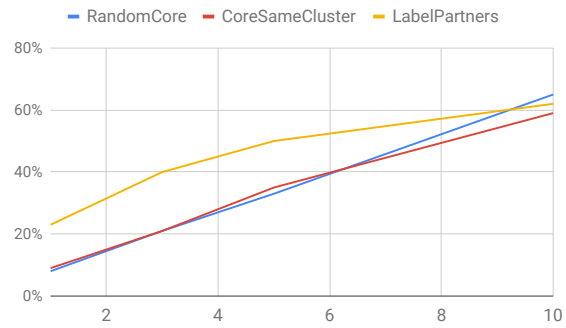
Já para o *hit* as seguintes hipóteses são levantadas:

- H0 (hipótese nula): os métodos possuem desempenho igual;
- H1 (hipótese alternativa): os métodos possuem desempenho distinto.

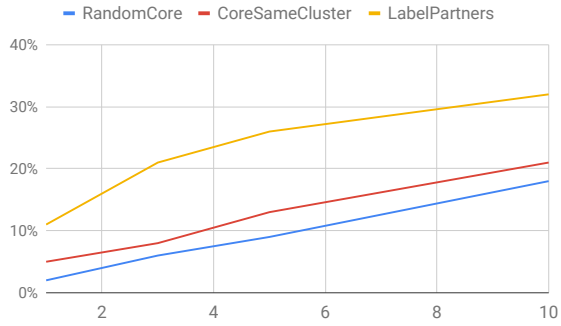
Figura 33 – Comparação da precisão entre os diferentes métodos.



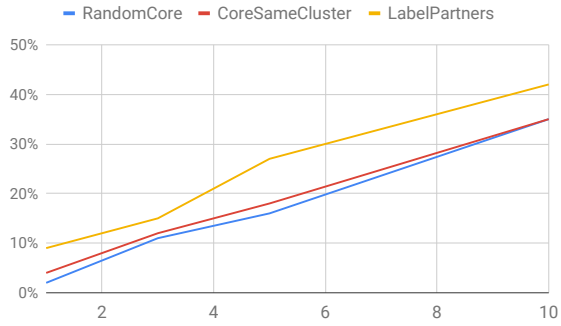
(a) Node.js



(b) Symfony



(c) Kubernetes



(d) Tensorflow

As amostras com resultados dos testes paramétricos ou não paramétricos executados (de acordo com o protocolo apresentado na seção 5.1.8) onde $p < 0.05$ são considerados estatisticamente relevantes, e apresentados na tabelas 14 e 15.

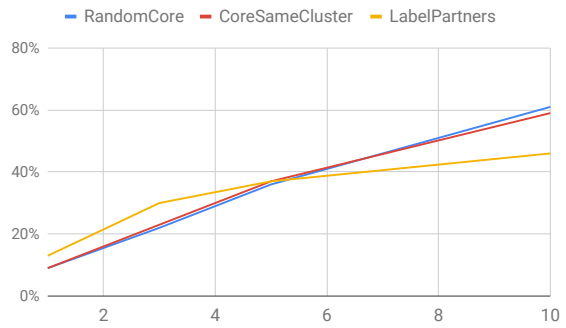
É possível observar que em todos os casos as amostras não eram homocedásticas e normais, e por isso foram aplicados testes não paramétricos. Foram poucos os casos onde existiu diferença significativa entre os métodos *RandomCore* e *CoreSameCluster*, notoriamente no projeto Kubernetes. A maior parte das comparações significativas mostra o método *LabelPartners* como mais preciso, salvo em casos de $k = 10$ no Node.js.

O comportamento das medições significativas do *hit* é muito próximo da precisão. Em apenas um dos casos de comparação com $p < 0.05$ (Symfony, $k = 10$) não houve diferença relevante no mesmo caso para ambas as métricas de proximidade.

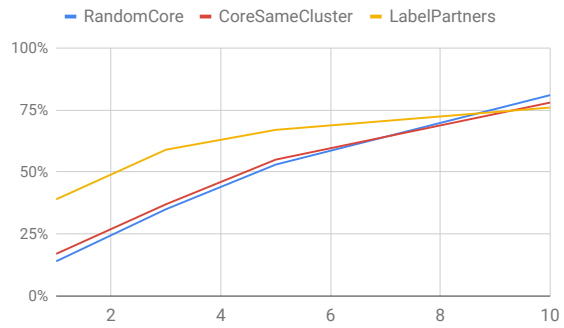
5.3.2 **Métricas de eficiência**

As métricas de eficiência se aproximam mais dos objetivos do presente trabalho, uma vez que através delas busca-se traduzir se os métodos apresentados potencializam a colaboração no contexto estudado. Para isso é necessário comparar dois grupos distintos de revisores: aqueles que foram recomendados pelos métodos e aqueles que não foram. A

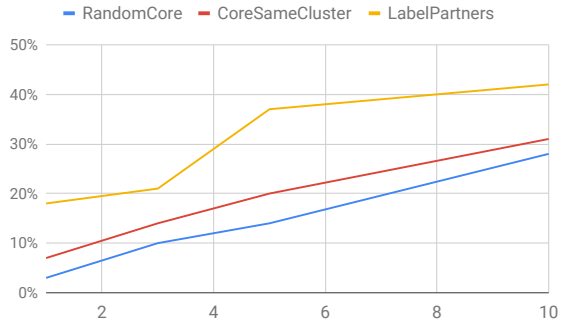
Figura 34 – Comparação da precisão entre os diferentes métodos.



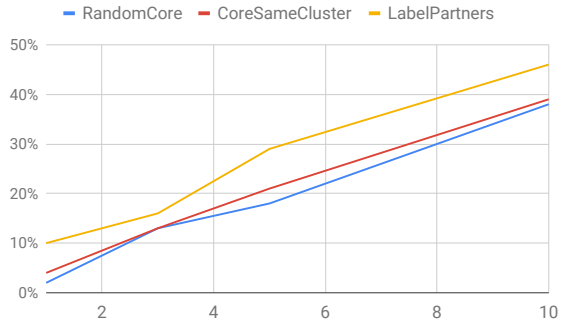
(a) Node.js



(b) Symfony



(c) Kubernetes



(d) Tensorflow

diferença no desempenho destes conjuntos pode indicar em quais casos os métodos são responsáveis por indicar indivíduos aptos à um grau maior de colaboração, interação e convergência com os objetivos da revisão de código no desenvolvimento distribuído, amplamente apresentados e discutidos neste trabalho. Diferente das métricas de proximidade, apenas os resultados significativos serão apresentados aqui, em decorrência do espaço necessário para análise de um grupo maior de métricas. Como descrito anteriormente, é necessário definir a hipótese nula e alternativa para cada métrica. São apresentados os casos onde $p < 0.05$. A Tabela 16 descreve as métricas, sua simbologia e as hipóteses referentes a cada uma delas.

A análise é feita para cada um dos projetos, com o tamanho dos grupos recomendados $k \in \{1, 3, 5, 10\}$ já aplicados anteriormente, de forma a facilitar posteriores comparações. Para cada método são executados 80 combinações diferentes de projetos, ks e métricas de avaliação. Os resultados são apresentados em tabelas distintas. Para o *RandomCore*, como apresenta a Tabela 17, foram poucos os casos testados onde pode ser observada diferença significativa entre os grupos de recomendados e não recomendados. Neles, a maioria expressa uma diferença pequena entre os grupos ou ainda situações onde os indivíduos recomendados se saem pior do que os não recomendados.

É possível observar que não há um comportamento diretamente relacionado com

Tabela 14 – Comparação da precisão apontando para diferenças estatisticamente relevantes

Projeto	k	Método A	Método B	Média A	Média B	Tipo de Teste	p-value	Proporção
Node.js	1	RandomCore	LabelPartners	0.04	0.06	Não Paramétrico	0.006	72%
Node.js	1	CoreSameCluster	LabelPartners	0.04	0.06	Não Paramétrico	0.001	72%
Node.js	3	RandomCore	LabelPartners	0.12	0.17	Não Paramétrico	0.00002	71%
Node.js	3	CoreSameCluster	LabelPartners	0.14	0.17	Não Paramétrico	0.00058	79%
Node.js	10	RandomCore	LabelPartners	0.42	0.32	Não Paramétrico	0.00000	130%
Node.js	10	CoreSameCluster	LabelPartners	0.40	0.32	Não Paramétrico	0.00000	127%
Symfony	1	RandomCore	LabelPartners	0.08	0.23	Não Paramétrico	0.00000	33%
Symfony	1	CoreSameCluster	LabelPartners	0.09	0.23	Não Paramétrico	0.00000	38%
Symfony	3	RandomCore	LabelPartners	0.21	0.40	Não Paramétrico	0.00000	52%
Symfony	3	CoreSameCluster	LabelPartners	0.21	0.40	Não Paramétrico	0.00000	52%
Symfony	5	RandomCore	LabelPartners	0.33	0.50	Não Paramétrico	0.00000	65%
Symfony	5	CoreSameCluster	LabelPartners	0.35	0.50	Não Paramétrico	0.00000	68%
Symfony	10	RandomCore	CoreSameCluster	0.65	0.59	Não Paramétrico	0.00826	109%
Kubernetes	1	RandomCore	CoreSameCluster	0.02	0.05	Não Paramétrico	0.00000	37%
Kubernetes	1	RandomCore	LabelPartners	0.02	0.11	Não Paramétrico	0.00000	15%
Kubernetes	1	CoreSameCluster	LabelPartners	0.05	0.11	Não Paramétrico	0.00000	40%
Kubernetes	3	RandomCore	CoreSameCluster	0.06	0.08	Não Paramétrico	0.00004	69%
Kubernetes	3	RandomCore	LabelPartners	0.06	0.35	Não Paramétrico	0.00000	27%
Kubernetes	3	CoreSameCluster	LabelPartners	0.08	0.35	Não Paramétrico	0.00000	39%
Kubernetes	5	RandomCore	CoreSameCluster	0.09	0.13	Não Paramétrico	0.00000	70%
Kubernetes	5	RandomCore	LabelPartners	0.09	0.26	Não Paramétrico	0.00000	34%
Kubernetes	5	CoreSameCluster	LabelPartners	0.13	0.26	Não Paramétrico	0.00000	48%
Kubernetes	10	RandomCore	CoreSameCluster	0.18	0.21	Não Paramétrico	0.00801	88%
Kubernetes	10	RandomCore	LabelPartners	0.18	0.32	Não Paramétrico	0.00000	56%
Kubernetes	10	CoreSameCluster	LabelPartners	0.21	0.32	Não Paramétrico	0.00000	64%
Tensorflow	1	RandomCore	CoreSameCluster	0.02	0.04	Não Paramétrico	0.02871	41%
Tensorflow	1	RandomCore	LabelPartners	0.02	0.09	Não Paramétrico	0.00000	18%
Tensorflow	1	CoreSameCluster	LabelPartners	0.04	0.09	Não Paramétrico	0.00286	44%
Tensorflow	5	RandomCore	LabelPartners	0.16	0.27	Não Paramétrico	0.00020	33%
Tensorflow	5	CoreSameCluster	LabelPartners	0.18	0.27	Não Paramétrico	0.00261	38%
Tensorflow	10	RandomCore	LabelPartners	0.35	0.42	Não Paramétrico	0.01961	83%
Tensorflow	10	CoreSameCluster	LabelPartners	0.35	0.42	Não Paramétrico	0.02402	83%

o tamanho do conjunto recomendado, o que desassocia os resultados das métricas de proximidade. De maneira geral o número de respostas e o número de comentários foram as métricas de pior desempenho deste método, enquanto a maior diferença ficou com a número de reações, que em todos os casos significativos, mostra uma vantagem do grupo recomendado. No projeto Node.js é possível observar que nos casos significativos houve apenas um caso com uma ligeira vantagem do grupo recomendado, e nos outros este grupo se sai pior. No Kubernetes há apenas um caso significativo, onde o tamanho dos comentários é maior dentre os autores indicados. A Figura 35 destaca alguns dos casos mais relevantes encontrados. É possível perceber performances distintas especialmente na quantidade de respostas, onde no Symfony poucos são os casos que os recomendados não obtiveram nenhuma interação do tipo, frente ao desempenho do grupo não recomendado.

Já no Tensorflow (Figura 35d), apesar dos *outliers* que receberam muitas respostas, a maioria dos recomendados teve um desempenho significativamente maior que os não recomendados. Já a média do tamanho dos comentários no Symfony (Figura 35a) é influenciada pela presença de uma amostra com mais de 5000 mil caracteres, muito maior que o normal.

Tabela 15 – Comparação do *hit* apontando para diferenças estatisticamente relevantes

Projeto	**k**	**Método A**	**Método B**	**Média A**	**Média B**	**Tipo de Teste**	**p-value**	**Proporção**
Node.js	1	RandomCore	LabelPartners	0.09	0.13	Não Paramétrico	0.00572	72%
Node.js	1	CoreSameCluster	LabelPartners	0.9	0.13	Não Paramétrico	0.00175	68%
Node.js	3	RandomCore	LabelPartners	0.22	0.30	Não Paramétrico	0.0002	72%
Node.js	3	CoreSameCluster	LabelPartners	0.23	0.30	Não Paramétrico	0.0037	77%
Node.js	10	RandomCore	LabelPartners	0.61	0.46	Não Paramétrico	0.00000	133%
Node.js	10	CoreSameCluster	LabelPartners	0.59	0.46	Não Paramétrico	0.00000	129%
Symfony	1	RandomCore	LabelPartners	0.14	0.39	Não Paramétrico	0.00000	34%
Symfony	1	CoreSameCluster	LabelPartners	0.17	0.39	Não Paramétrico	0.00000	42%
Symfony	3	RandomCore	LabelPartners	0.35	0.49	Não Paramétrico	0.00000	59%
Symfony	3	CoreSameCluster	LabelPartners	0.37	0.49	Não Paramétrico	0.00000	62%
Symfony	5	RandomCore	LabelPartners	0.53	0.67	Não Paramétrico	0.00000	79%
Symfony	5	CoreSameCluster	LabelPartners	0.55	0.67	Não Paramétrico	0.00002	82%
Symfony	10	RandomCore	LabelPartners	0.81	0.79	Não Paramétrico	0.01013	107%
Kubernetes	1	RandomCore	CoreSameCluster	0.03	0.07	Não Paramétrico	0.00000	39%
Kubernetes	1	RandomCore	LabelPartners	0.03	0.18	Não Paramétrico	0.00000	15%
Kubernetes	1	CoreSameCluster	LabelPartners	0.07	0.18	Não Paramétrico	0.00000	39%
Kubernetes	3	RandomCore	CoreSameCluster	0.10	0.14	Não Paramétrico	0.00005	73%
Kubernetes	3	RandomCore	LabelPartners	0.10	0.31	Não Paramétrico	0.00000	32%
Kubernetes	3	CoreSameCluster	LabelPartners	0.14	0.31	Não Paramétrico	0.00000	43%
Kubernetes	5	RandomCore	CoreSameCluster	0.14	0.20	Não Paramétrico	0.00000	72%
Kubernetes	5	RandomCore	LabelPartners	0.14	0.37	Não Paramétrico	0.00000	39%
Kubernetes	5	CoreSameCluster	LabelPartners	0.20	0.37	Não Paramétrico	0.00000	54%
Kubernetes	10	RandomCore	CoreSameCluster	0.28	0.31	Não Paramétrico	0.01013	89%
Kubernetes	10	RandomCore	LabelPartners	0.28	0.42	Não Paramétrico	0.00000	65%
Kubernetes	10	CoreSameCluster	LabelPartners	0.31	0.42	Não Paramétrico	0.00000	72%
Tensorflow	1	RandomCore	CoreSameCluster	0.02	0.04	Não Paramétrico	0.02834	43%
Tensorflow	1	RandomCore	LabelPartners	0.02	0.10	Não Paramétrico	0.00000	20%
Tensorflow	1	CoreSameCluster	LabelPartners	0.04	0.10	Não Paramétrico	0.00290	45%
Tensorflow	5	RandomCore	LabelPartners	0.18	0.29	Não Paramétrico	0.00022	61%
Tensorflow	5	CoreSameCluster	LabelPartners	0.21	0.29	Não Paramétrico	0.00283	69%
Tensorflow	10	RandomCore	LabelPartners	0.38	0.46	Não Paramétrico	0.02034	83%
Tensorflow	10	CoreSameCluster	LabelPartners	0.39	0.46	Não Paramétrico	0.02915	84%

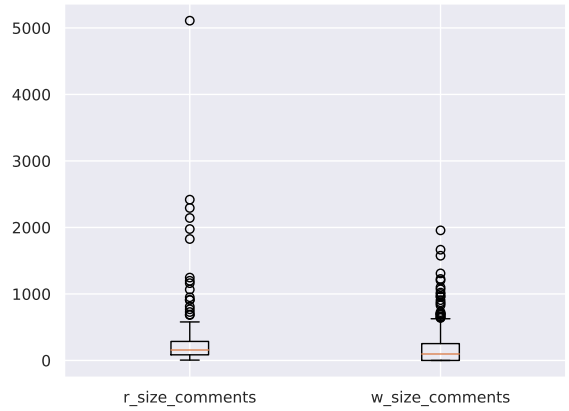
Tabela 16 – Métricas de proximidade e as respectivas hipóteses

Métrica	Variável	Descrição	H0 (hipótese nula)	H1 (hipótese alternativa)
Número de Comentários	*num_comments*	Média da quantidade de comentários que cada revisor fez por revisão	Os recomendados pelo método proposto fazem comentários do mesmo tamanho que os não recomendados	Os recomendados pelo método proposto fazem comentários de tamanho distinto em relação aos não recomendados
Número de Reações	*num_reactions*	Média de reações que cada revisor recebeu por revisão	Os recomendados pelo método proposto recebem a mesma quantidade de reações que os não recomendados	Os recomendados pelo método proposto recebem quantidades distintas de reações em relação aos não recomendados
Número de repostas	*num_replies*	Média de respostas que cada revisor recebeu por revisão	Os recomendados pelo método proposto recebem a mesma quantidade de respostas que os não recomendados	Os recomendados pelo método proposto recebem quantidades distintas de respostas em relação aos não recomendados
Tamanho dos comentários	*size_comments*	Tamanho médio dos comentários que cada revisor fez por revisão	Os recomendados pelo método proposto escrevem comentários do mesmo tamanho que os não recomendados	Os recomendados pelo método proposto escrevem comentários de tamanho distintinto em relação aos não recomendados
Proporção de *non stop words*	*rate_non_stop_words*	Média da proporção de *non stop words* dos comentários que cada revisor fez por revisão	Os comentários dos recomendados pelo método proposto apresentam a mesma proporção de *non stop words* que os não recomendados	Os comentários dos recomendados pelo método proposto apresentam proporção de *non stop words* distinta em relação aos não recomendados

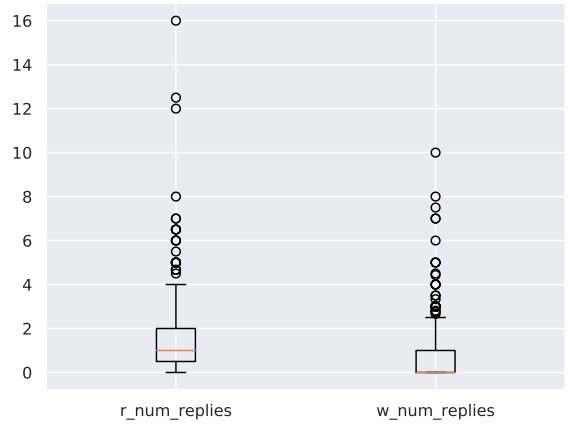
Tabela 17 – Casos com diferença significativa para o método *RandomCore*

Projeto	k	Métrica	Média Recomendados	Média Não Recomendados	Tipo de Teste	p-value	Proporção
Node.js	1	num_comments	2.39	2.36	Não Paramétrico	0.03016	101%
Node.js	1	num_replies	1.20	1.34	Não Paramétrico	0.00638	90%
Node.js	3	rate_non_stop_words	1.23	1.29	Não Paramétrico	0.01321	95%
Symfony	5	size_comments	258.90	297.70	Não Paramétrico	0.00384	87%
Symfony	10	num_reactions	0.31	0.29	Não Paramétrico	0.02873	107%
Symfony	10	num_replies	1.50	1.30	Não Paramétrico	0.00750	115%
Symfony	10	rate_non_stop_words	1.35	1.28	Não Paramétrico	0.03289	105%
Kubernetes	3	size_comments	472.64	418.93	Não Paramétrico	0.00150	113%
Tensorflow	3	num_replies	1.27	2.19	Não Paramétrico	0.04128	58%
Tensorflow	10	num_reactions	0.35	0.10	Não Paramétrico	0.04107	350%

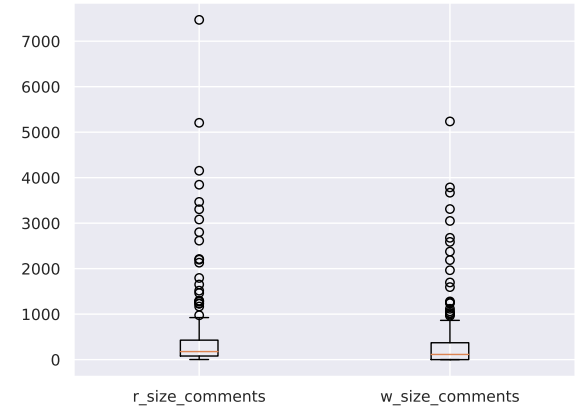
Figura 35 – Destaques para o método *RandomCore*



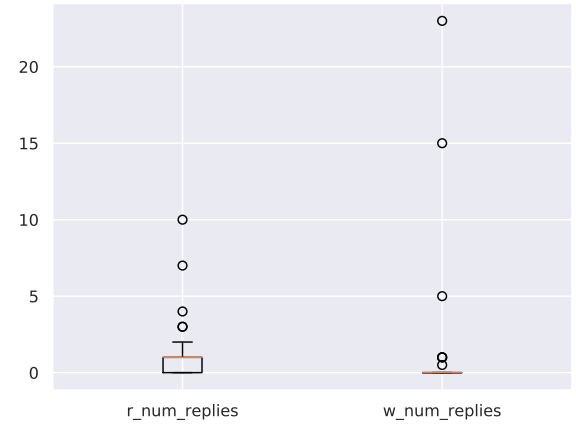
(a) Symfony ($size\_comments, k = 5$)



(b) Symfony ($num\_replies, k = 10$)



(c) Kubernetes ($size\_comments, k = 3$)



(d) Tensorflow ($num\_replies, k = 3$)

A Tabela 18 apresenta os casos para o método *CoreSameCluster* onde foram verificadas diferenças significativas entre os dois grupos. É um conjunto maior que o encontrado para o método *CoreSameCluster*, além de mais consistente em métricas como a proporção de *non stop words* e do número de respostas. O número de comentários também foi claramente maior para o conjunto dos recomendados. Diferente do primeiro método, apenas em um dos casos é possível observar um desempenho pior no grupo recomendado: Para o projeto Tensorflow, o número de respostas recebidas pelo grupo indicado quando $k = 3$ foi apenas 71% do total recebido pelo grupo não recomendado.

Tabela 18 – Casos com diferença significativa para o método *CoreSameCluster*

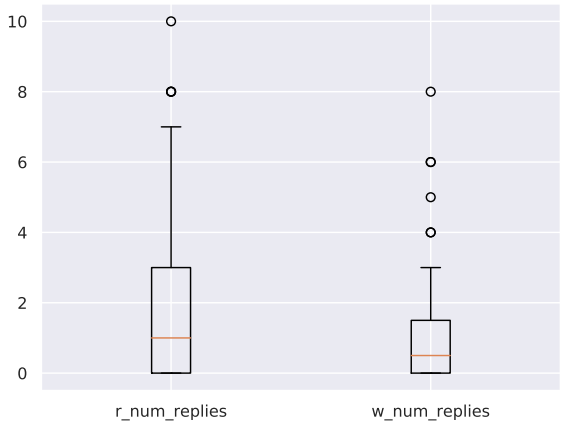
Projeto	k	Métrica	Média Recomendados	Média Não Recomendados	Tipo de Teste	p-value	Proporção
Node.js	1	num_reactions	0.5	0.47	Não Paramétrico	0.02897	106%
Symfony	1	num_replies	1.97	1.31	Não Paramétrico	0.01197	150%
Symfony	3	num_comments	2.95	2.25	Não Paramétrico	0.00922	131%
Symfony	3	num_replies	1.73	1.29	Não Paramétrico	0.01639	134%
Symfony	3	rate_non_stop_words	1.58	1.22	Não Paramétrico	0.01175	130%
Symfony	5	num_replies	1.58	1.24	Não Paramétrico	0.00283	127%
Symfony	10	num_comments	2.59	2.33	Não Paramétrico	0.02054	111%
Symfony	10	num_replies	1.57	1.27	Não Paramétrico	0.00279	124%
Symfony	10	rate_non_stop_words	1.4	1.27	Não Paramétrico	0.03665	110%
Tensorflow	3	num_replies	1.53	2.15	Não Paramétrico	0.04805	71%
Tensorflow	10	num_reactions	0.31	0.1	Não Paramétrico	0.00656	310%
Kubernetes	3	num_comments	4.15	3.1	Não Paramétrico	0.00338	134%
Kubernetes	3	rate_non_stop_words	2.29	1.72	Não Paramétrico	0.01102	133%
Kubernetes	10	num_comments	3.72	3.11	Não Paramétrico	0.00524	120%
Kubernetes	10	rate_non_stop_words	2.04	1.73	Não Paramétrico	0.01274	118%

É possível observar para o segundo método que o tamanho de k não teve impacto direto na comparação entre os grupos. Para o número de reações no Tensorflow ($k = 10$), foi registrada a maior discrepância entre os grupos: os recomendados em média receberam o triplo de reações do que seus pares não recomendados. O pior desempenho do método é associado ao projeto Node.js, no qual em apenas um caso o número de reações foi levemente maior para o grupo indicado (106%). A Figura 36 ilustra com mais detalhes a comparação entre os grupos. O destaque fica para a quantidade de respostas, que foi maior em diversos projetos e tamanhos de grupos indicados. A mediana dos grupos recomendados é significativamente maior nos casos apresentados.

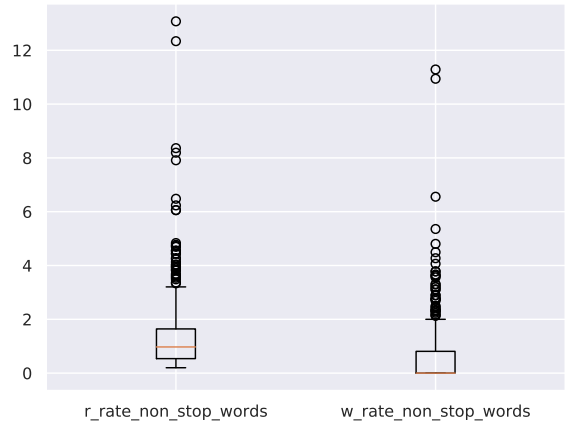
A proporção de *non stop words* é outro destaque no projeto Symfony, como mostra a Figura 36b. É possível observar que nenhum dos comentários do grupo recomendado contém apenas palavras de parada, realidade distinta a do grupo não recomendado: todo o primeiro quartil do conjunto de indicados está acima da distribuição análoga do grupo de comparação.

O *LabelPartners* apresentou o resultado mais consistente, como apresenta a Tabela 19. Isso porque os resultados foram constantes em todos os projetos, com diferentes valores de k. Todas as métricas foram significativas em pelo menos um projeto, e no geral a diferença prevaleceu para todos os tamanhos de conjuntos recomendados. O método é também o detentor de maior número de casos onde a diferença de desempenho dos dois grupos é estatisticamente significativa. Em 44 ocasiões o método apresentou grupos com

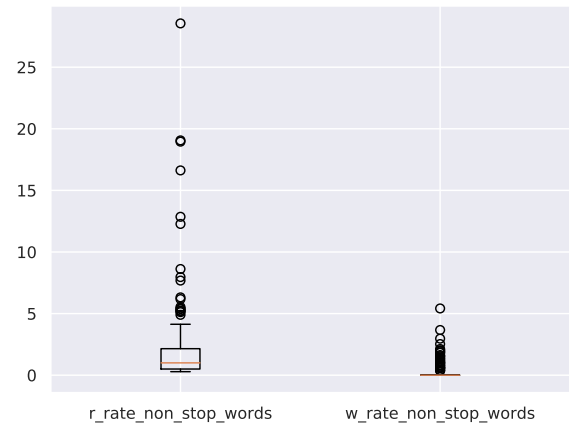
Figura 36 – Destaques para o método *CoreSameCluster*



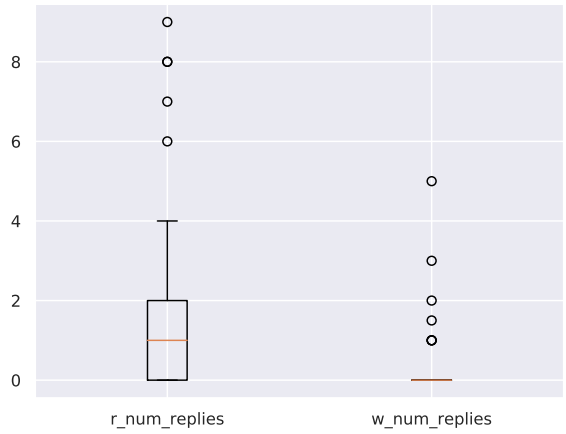
(a) Symfony ($num\_replies, k = 1$)



(b) Symfony ($non\_stop\_words, k = 10$)



(c) Tensorflow ($num\_replies, k = 10$)



(d) Tensorflow ($num\_replies, k = 3$)

desempenho significativamente distintos em relação aos indivíduos não recomendados, e apenas em uma delas o resultado do segundo conjunto foi ligeiramente melhor.

A métrica de *non stop words* foi o principal destaque em relação aos outros métodos, tendo sido significativamente melhor em vários casos. A quantidade de reações praticamente não foi relevante em nenhum caso, e quando foi mostrou diferenças pequenas. O método também foi o único a se destacar nos projetos Node.js e Kubernetes, sendo neste segundo responsável por indicar grupos mais eficientes em quatro das cinco métricas em todos os casos. Este comportamento também pode ser observado no Tensorflow, mas apenas para valores de k mais baixos. Com os destaques deste método (apresentados na Figura 37) é possível analisar o número de reações, métrica pouco significativa nos outros métodos mas que foi mais relevante nestas amostras. É um conjunto de avaliações esparso, visto que a maior parte dos comentários não parece receber nenhuma reação.

O tamanho dos comentários é outra métrica importante, podendo indicar que determinado grupo tem o hábito de dar respostas mais completas e relevantes às discussões.

Tabela 19 – Casos com diferença significativa para o método *LabelPartners*

Projeto	k	Métrica	Média Recomendados	Média Não Recomendados	Tipo de Teste	p-value	Proporção
Node.js	1	num_comments	3.42	2.35	Não Paramétrico	0.00964	146%
Node.js	1	num_replies	2.14	1.31	Não Paramétrico	0.00995	163%
Node.js	1	size_comments	483.62	309.8	Não Paramétrico	0.00463	156%
Node.js	1	rate_non_stop_words	1.89	1.27	Não Paramétrico	0.0308	149%
Node.js	3	size_comments	385.53	313.36	Não Paramétrico	0.00399	123%
Node.js	5	num_replies	1.69	1.33	Não Paramétrico	0.0358	127%
Node.js	5	size_comments	369.76	316.03	Não Paramétrico	0.00348	117%
Node.js	10	num_comments	2.55	2.38	Não Paramétrico	0.02944	107%
Node.js	10	num_replies	1.61	1.31	Não Paramétrico	0.02298	123%
Node.js	10	size_comments	369.43	300.3	Não Paramétrico	0.00008	123%
Symfony	1	num_comments	3.03	2.26	Não Paramétrico	0.00789	134%
Symfony	1	rate_non_stop_words	1.68	1.21	Não Paramétrico	0.01234	139%
Symfony	3	num_comments	2.67	2.23	Não Paramétrico	0.01991	120%
Symfony	3	num_reactions	0.28	0.3	Não Paramétrico	0.04138	96%
Symfony	3	rate_non_stop_words	1.48	1.2	Não Paramétrico	0.01863	123%
Symfony	5	num_comments	2.56	2.3	Não Paramétrico	0.02954	111%
Symfony	5	rate_non_stop_words	1.4	1.24	Não Paramétrico	0.03355	113%
Symfony	10	rate_non_stop_words	1.36	1.22	Não Paramétrico	0.04724	111%
Tensorflow	1	num_comments	5.74	3.71	Não Paramétrico	0.00202	155%
Tensorflow	1	num_reactions	0.8	0.11	Não Paramétrico	0.00139	727%
Tensorflow	1	num_replies	3.43	1.93	Não Paramétrico	0.0054	178%
Tensorflow	1	size_comments	1118.74	444.74	Não Paramétrico	0.00156	252%
Tensorflow	1	rate_non_stop_words	3.05	2.27	Não Paramétrico	0.00554	134%
Tensorflow	3	num_comments	4.47	3.79	Não Paramétrico	0.00219	118%
Tensorflow	3	num_reactions	0.55	0.1	Não Paramétrico	0.01798	550%
Tensorflow	3	num_replies	2.77	1.95	Não Paramétrico	0.01184	142%
Tensorflow	3	size_comments	784.84	456.97	Não Paramétrico	0.01579	172%
Tensorflow	3	rate_non_stop_words	2.41	2.32	Não Paramétrico	0.00379	104%
Kubernetes	1	num_comments	3.94	3.11	Não Paramétrico	0.00127	127%
Kubernetes	1	num_replies	2.65	1.89	Não Paramétrico	0.02029	140%
Kubernetes	1	size_comments	550.93	415.52	Não Paramétrico	0.02411	133%
Kubernetes	1	rate_non_stop_words	2.19	1.72	Não Paramétrico	0.00453	127%
Kubernetes	3	num_comments	3.81	3.06	Não Paramétrico	0.00014	125%
Kubernetes	3	num_replies	2.52	1.86	Não Paramétrico	0.00206	135%
Kubernetes	3	size_comments	538.81	410.01	Não Paramétrico	0.00889	131%
Kubernetes	3	rate_non_stop_words	2.06	1.7	Não Paramétrico	0.00164	121%
Kubernetes	5	num_comments	3.71	3.03	Não Paramétrico	0.00009	122%
Kubernetes	5	num_replies	2.44	1.84	Não Paramétrico	0.00013	133%
Kubernetes	5	size_comments	513.72	405.75	Não Paramétrico	0.00758	127%
Kubernetes	5	rate_non_stop_words	2.03	1.68	Não Paramétrico	0.00116	121%
Kubernetes	10	num_comments	3.59	3.02	Não Paramétrico	0.00003	119%
Kubernetes	10	num_replies	2.36	1.81	Não Paramétrico	0.00003	130%
Kubernetes	10	size_comments	485.31	406.54	Não Paramétrico	0.0354	119%
Kubernetes	10	rate_non_stop_words	1.96	1.68	Não Paramétrico	0.00022	117%

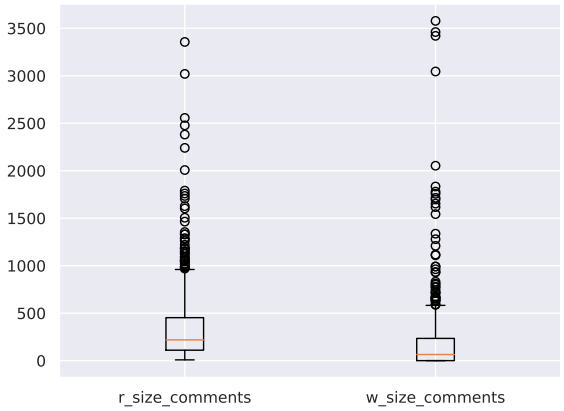
O método *LabelPartners* apresenta o maior número de casos onde a diferença no tamanho dos comentários foi significativa, tendo pelo menos um representante em cada um dos projetos avaliados.

5.4 DISCUSSÃO DOS RESULTADOS

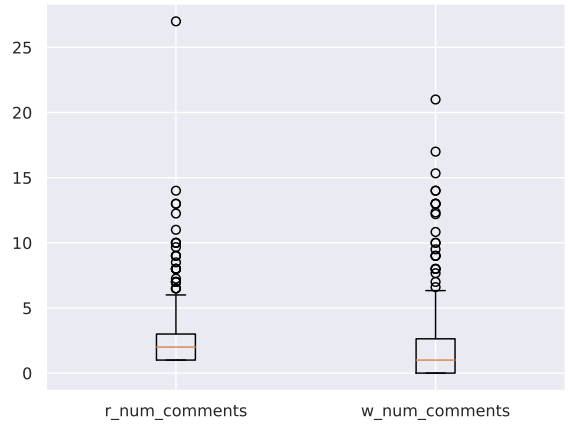
Os resultados da sessão anterior descrevem o desempenho dos métodos propostos de acordo com as métricas estabelecidas, condizentes com os objetivos do trabalho. Os projetos foram escolhidos para validar os métodos de acordo com a sua relevância no contexto estudado. A avaliação foi feita utilizando a base histórica dos repositórios, separando em um grupo para preparação das redes de desenvolvedores modeladas e outro para medir o desempenho dos métodos. Os indicadores foram divididos em duas categorias. As métricas de proximidade são condizentes com a literatura relacionada, especialmente de sistemas de recomendação.

Neste grupo é possível perceber que, especialmente em conjuntos de recomendação

Figura 37 – Destaques para o método *LabelPartners*

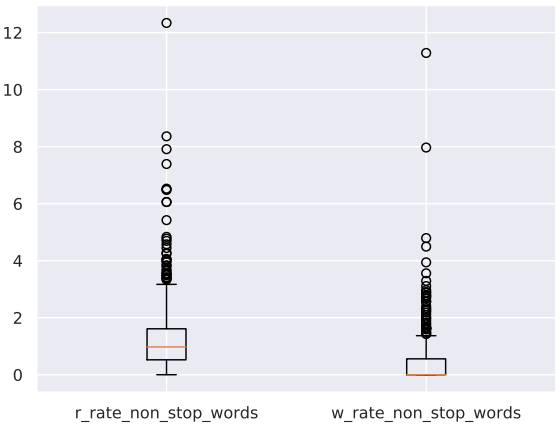


(a) Node.js ($size\_comment, k = 10$)



(b) Node.js ($num\_comments, k = 5$)

(c) Symfony ($num\_reactions, k = 3$)



(d) Symfony ($non\_stop\_words, k = 10$)

menores, o método *LabelPartners* foi mais eficiente que o *RandomCore* e *CoreSameCluster*. Estes dois últimos tiveram resultados próximos e mais lineares de acordo com a variação de k, fenômeno que não ocorreu no primeiro. A tendência de aproximação dos resultados para os três métodos em valore de k maiores se dá pelas condições próximas dos três para quando o grupo preferencial de revisores não é grande o suficiente para cobrir o número de recomendações solicitado: todos preenchem o restante com *cores* aleatórios.

Nos casos onde o autor do *"pull request"* nunca havia contribuído ou não possuía uma relação forte suficiente para ser classificado em um dos grupos, os algoritmos *RandomCore* e *CoreSameCluster* sofriam maior revés. Isso porque qualquer tipo de relacionamento em determinado tópico, fraco ou forte, é suficiente para as indicações do método *LabelPartners*.

Apesar de comuns na literatura, os indicadores de proximidade não são suficientes para avaliar os métodos de acordo com a proposta deste trabalho. Isso porque os grupos de comparação são formados pelos indivíduos escolhidos pelo autor do projeto e pelos moderadores envolvidos. Ou seja, podem não ser a representação do grupo ideal, dada a

dificuldade de obter e processar manualmente todos os fatores que influenciam na capacidade de um desenvolvedor ser um bom revisor: disponibilidade, habilidade, conhecimento do tema, entre outras, como apresentado nas seções preliminares deste estudo.

Por isso o grupo de métricas de eficiência foi elaborado, tendo como base a literatura relacionada. O objetivo é avaliar o desempenho dos indicados em relação aos que não foram, de acordo com resultados associados a qualidade e proveito das interações no âmbito da revisão. Neste contexto, os resultados apontam que o *LabelPartners* foi mais eficiente em todas as variáveis apresentadas na seção anterior, com mais casos de diferenças significativas entre o grupo de recomendados e os grupos recomendados e não recomendados. Além do número de casos significativamente melhores ter sido maior, estes também foram os mais dispersos entre os diferentes projetos e tamanhos dos grupos recomendados. Por exemplo, foi o único método a mostrar melhoras significativas no Node.js. Os casos onde os resultados dos métodos *RandomCore* e *CoreSameCluster* são significativos são minoria, com diversas ocorrências de resultados onde o grupo recomendado foi menos eficiente.

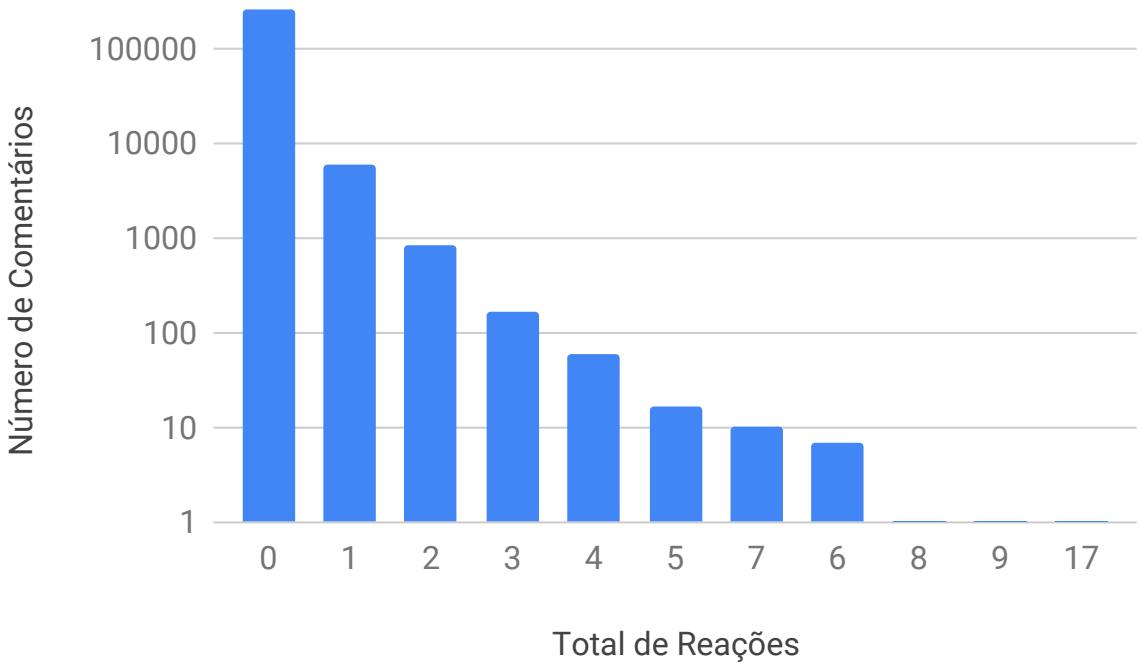
Especialmente em grupos recomendados menores, é possível observar que os indivíduos influentes são mais eficientes nas revisões quando os autores são provenientes do mesmo *cluster*. Isso indica a relevância do relacionamento prévio entre os pares na qualidade do processo. Estes resultados podem indicar que a influência exercida sobre alguns dos *cores* é compartimentalizada no projeto, sendo a proximidade com determinado tópico de conhecimento ou responsabilidade dentro do fluxo de trabalho fator determinante para a participação no processo de revisão.

Também é possível perceber que não há uma correlação direta entre as métricas de eficiência e proximidade. Não é possível observar maior desempenho das métricas de eficiência quando o valor de k aumenta, enquanto o crescimento da precisão e do *hit* aumentam consistentemente quanto maior é o grupo recomendado. Ao mesmo tempo, casos onde por exemplo o *CoreSameCluster* foi 27% mais preciso que o *LabelPartners* (Node.js, $k = 10$) apresentaram três métricas de eficiência positivas para o segundo método, enquanto o primeiro não conseguiu nenhuma.

Com o *LabelPartners*, indicadores como o tamanho e a quantidade de comentários foram significativamente maiores em todos os projetos. Em geral os valores para os grupos recomendados foram respectivamente 126% e 145% dos índices para os grupos não recomendados. O mesmo comportamento se repete para o índice de *non stop words*, que chega a ser 25% maior nos grupos recomendados pelo método.

Quanto às métricas escolhidas, apenas o número de reações está distribuído de forma que dificulta a análise dos seus resultados. Isso porque para os 265714 comentários analisados, existem apenas 8471 reações. Isso significa que em média cada interação do tipo recebe apenas 0,031 reações. E a distribuição da Figura 38 mostra que a mais de 80% dos comentários não recebe nenhuma reação.

Figura 38 – Distribuição de reações por comentário de revisão



A distribuição esparsa deste recurso faz com que as técnicas utilizadas para as outras variáveis não se apliquem tão bem em sua análise.

Como descrito no capítulo introdutório, a hipótese levantada pelo presente trabalho é:

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

Assim, a abordagem escolhida foi propor não um, mas três métodos baseados em linhas de pesquisa já conhecidas e indicadas por trabalhos anteriores, com foco na interação entre os indivíduos envolvidos no processo de revisão. Os métodos foram baseados em diferentes aspectos da relação entre os desenvolvedores e, de acordo com os dados disponíveis e objetivos, podem ser melhor aplicados em determinados casos. As métricas escolhidas para a avaliação também foram escolhidas de acordo com a hipótese central, já que traduzem comportamentos relevantes para a qualidade da revisão.

Os resultados indicam que o método *LabelPartners* foi capaz de indicar revisores mais propensos à colaboração e a contribuir com comentários convergentes com os objetivos do processo de revisão de código.

6 CONSIDERAÇÕES FINAIS

O *code review* é uma das principais práticas da engenharia de software que visam aumentar a qualidade de código e a detecção precoce de defeitos. Diversos autores mostraram a capacidade desta técnica em gerar interações valiosas entre autores e a implantação de soluções melhores.

Intrinsecamente colaborativa, os resultados do processo são penalizados diretamente pelas diferentes ameaças inerentes ao desenvolvimento distribuído e seus desdobramentos. A prática ganha contornos ainda mais importantes no contexto *open source* quando se posiciona como uma espécie de *gateway*, onde os membros responsáveis pelos direcionamentos de um projeto buscam garantir os requisitos de qualidade e segurança necessários e aderência aos objetivos traçados.

Apesar de ganhar importância no contexto distribuído, a prática de revisão sofre com particularidades inerentes a esse paradigma. O processo de revisão pode se tornar lento e ineficiente quando a colaboração é afetada, devido aos baixos níveis de participação e cobertura. Este cenário se consolida com os desafios de comunicação, horários, disponibilidade, diferenças culturais e idiomáticas, barreiras tecnológicas e outros aspectos que a distância geográfica projeta no desenvolvimento de software. O mesmo vale para a disseminação do conhecimento, que fica prejudicada.

A prática de revisão evoluiu junto com as tecnologias e abordagens do desenvolvimento de software. Hoje é geralmente aplicada através de ferramentas computacionais e na presença de idiossincrasias que influenciam no seu desempenho. Um dos contextos particulares com forte presença da técnica é o de desenvolvimento "*pull based*", presente em ferramentas como o GitHub e o GitLab e centrais para o desenvolvimento *Open Source*.

Tendo tais aspectos estabelecidos, o presente trabalho direciona os esforços numa particularidade da revisão de código em contextos distribuídos: a escolha do revisor adequado. É uma fase crucial do processo, uma vez que vários autores já evidenciaram a influência do perfil do revisor no impacto da abordagem. No desenvolvimento distribuído, tal fase é ainda mais complexa, pois os autores dos *changesets* e os mantenedores dos projetos precisam escolher em uma longa lista de potenciais revisores quais seriam os mais adequados àquele caso. Com um vasto número de opções e pouca informação disponível sobre aspectos técnicos e gerenciais (e.g. tempo disponível) dos pares (já que não há contato co-localizado entre eles) a natureza distribuída deste tipo de desenvolvimento dificulta a escolha do revisor.

Com a hipótese traçada, foram desenvolvidos métodos de recomendação de revisores para encontrar indivíduos capazes de influenciar positivamente o processo de revisão de código em ambientes distribuídos. Tais métodos foram implementados no Gitrev, uma ferramenta de recomendação de revisores baseada nos dados históricos dos projetos

hospedados no GitHub. Através de uma interface web, é possível informar uma determinada revisão em andamento e o sistema informa quais revisores são indicados, de acordo com cada método.

As abordagens se baseiam em características particulares deste tipo de desenvolvimento e direcionamentos de trabalhos anteriores na área. Foram empregadas técnicas de análise de redes sociais e clusterização para explorar o relacionamento entre os indivíduos e gerar saídas que condizem com os objetivos da revisão de código.

Dois dos métodos consistem em indicar membros centrais do projeto para o processo de revisão. O primeiro (*RandomCore*) indica estes membros sem distinção da proveniência do autor. Já o segundo, *CoreSameCluster*, busca *cores* no contexto do círculo de relacionamento do criador *"pull request"*. É possível notar que o segundo é mais eficiente sob os parâmetros que norteiam este trabalho, indicando a relevância do relacionamento pregresso entre os indivíduos na qualidade da revisão. O terceiro método é pautado nessa asserção, e utiliza das relações diretas entre os desenvolvedores com o discernimento dos tópicos de conhecimento para encontrar os potenciais revisores. Os resultados mostram que esta abordagem é ainda mais convergente com os objetivos do estudo, ao indicar grupos de revisores que em mais casos se mostram significativamente mais propensos a fomentar um ambiente colaborativo no processo de revisão.

Além da presente dissertação, três outros trabalhos foram publicados como desdobramentos ou fundamentação para os resultados aqui apresentados. No CSCWD 2019 (*International Conference on Computer Supported Cooperative Work in Design*) foram duas publicações sobre análise das redes (SCHETTINO et al., 2019b) e aplicação dos modelos de clusterização no contexto dos sites de pergunta e resposta (QA) (HORTA et al., 2019). Já no CibSE 2019 (22nd Iberoamerican Conference on Software Engineering) foi apresentado o mapeamento e a revisão sistemática que sustentam a seção de fundamentação teórica deste texto (SCHETTINO et al., 2019a).

6.1 AMEAÇAS À VALIDADE

Existem ameaças à validade dos resultados aqui obtidos, que devem ser cuidadosamente observadas e ponderadas para utilização destes dados em aplicações industriais ou como base para trabalhos futuros. A maior parte destas ressalvas está relacionada ao ambiente real no qual foi conduzido o experimento, como qualquer trabalho de Engenharia de Software experimental (SJOBERG et al., 2002). Apesar dos esforços contrários relatados neste trabalho, os resultados podem ter sido influenciados, positiva ou negativamente, por variáveis alheias ao experimento, não mapeadas e relacionadas ao contexto da organização e dos projetos, como escopo, planejamento e distribuição de recursos. Nesta seção iremos discutir algumas delas.

As ameaças à generalização dos resultados aqui obtidos são caracterizadas como ameaças externas (WOHLIN et al., 2012). Nesta categoria podemos elencar o contexto distribuído do desenvolvimento, a categorização *open source* dos projetos e da plataforma única de estudo (GitHub). Apesar dos esforços para contemplar diferentes abordagens e mapear as particularidades destes componentes, estes fatores são ameaças à generalidade dos resultados obtidos.

Wholin et al. (WOHLIN et al., 2012) descrevem os riscos das influência de variáveis não controladas e desconhecidas como ameaças internas a validade dos resultados. Neste aspecto podemos destacar o pequeno conjunto de projetos, o que deixa os resultados mais sensíveis a particularidades de cada repositório, como formas de utilização das ferramentas e políticas de desenvolvimento e processo de trabalho, alheios ao escopo do trabalho. A transformação do processo durante o tempo de observação e coleta dos dados também podem impactar os resultados apresentados. Ao avaliar diretamente os métodos de recomendação, sem vínculo com a ferramenta desenvolvida, despreza-se a influência do software e fatores como usabilidade e portabilidade na avaliação dos métodos.

6.2 TRABALHOS FUTUROS

Como trabalhos futuros, os resultados obtidos podem ser comparados e analisados sob diferentes óticas com objetivo de aumentar o escopo e contexto de aplicação dos métodos apresentados. É possível englobar outros projetos, métricas, análises e hipóteses à estrutura desenvolvida ao longo da elaboração do presente documento. Além disso novas avaliações podem levar em consideração os impactos da utilização da ferramenta desenvolvida (GitRev), não só dos métodos de recomendação. Os métodos propostos podem ser melhorados levando em consideração a natureza das interações entre desenvolvedores, e comportamento histórico dessas ligações. Dados provenientes dos repositórios, alheios ao escopo deste trabalho, podem ser utilizados para enriquecer os métodos e as formas de avaliação propostas.

Além deste aprofundamento, outros objetivos da revisão de código, como por exemplo aprendizado e treinamento de novos profissionais, também podem ser levados em consideração na avaliação e aprimoramento das abordagens desenvolvidas. Neste sentido, estudos mais aprofundados em formação de grupos e recomendação de equipes podem potencializar os indícios encontrados. Para revisão com finalidades específicas, como auditoria e segurança, podem ser avaliadas outras métricas e particularidades para potencializar os efeitos dos métodos propostos. Tais avanços podem ser utilizados para recomendação de revisores entre projetos distintos, de acordo com os ecossistemas formados por tecnologias, tendências e abordagens distintas.

Por fim, recomenda-se a utilização das estruturas e ferramentas confeccionadas

para extração, execução e análise dos experimentos para outros estudos que utilizam dados de repositórios de software e contexto de desenvolvimento distribuído. Tais componentes foram modelados e implementados visando a extensão e reprodução dos experimentos propostos.

REFERÊNCIAS

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.

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.

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

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.

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.

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

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.

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.

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.

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

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.

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.

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.

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.

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

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

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>.

CAPURUÇO, R. A.; CAPRETZ, L. F. Integrating recommender information in social ecosystems decisions. In: ACM. *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*. [S.l.], 2010. p. 143–150.

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

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

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.

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.

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.

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.

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.

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.

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.

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>.

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.

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

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.

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>.

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

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.

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.

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.

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.

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.

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>.

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

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.

HORTA, V. et al. Collaboration analysis in global software development. In: IEEE. *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. [S.l.], 2019. p. 464–469.

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.

JANSEN, S.; CAPELLEVEEN, G. van. 10. quality review and approval methods for extensions in software ecosystems. *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry*, Edward Elgar Publishing, p. 187, 2013.

JANSEN, S.; FINKELSTEIN, A.; BRINKKEMPER, S. A sense of community: A research agenda for software ecosystems. In: IEEE. *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*. [S.l.], 2009. p. 187–190.

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.

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.

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.

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

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

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>.

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

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.

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>.

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

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.

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.

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.

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.

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>.

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.

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.

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.

OPEN SOURCE INITIATIVE AND OTHERS. *The MIT license*. 2000. Disponível em: <https://opensource.org/licenses/MIT>.

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.

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

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

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.

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

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

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.

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>.

ROBERT, K. Y. Case study research: Design and methods. *sage publications*, 1994.

SCHETTINO, V. et al. Towards code reviewer recommendation: a systematic review and mapping of the literature. In: *22th Ibero-American Conference on Software Engineering (CIbSE 2019)*. [S.l.: s.n.], 2019.

SCHETTINO, V. et al. Towards community and expert detection in open source global development. In: IEEE. *2019 IEEE 23rd International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. [S.l.], 2019. p. 350–355.

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.

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.

SJOBERG, D. I. K. et al. Conducting realistic experiments in software engineering. In: *Proceedings International Symposium on Empirical Software Engineering*. [S.l.: s.n.], 2002. p. 17–26.

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.

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

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.

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

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

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

WILBUR, W. J.; SIROTKIN, K. The automatic identification of stop words. *Journal of information science*, Sage Publications Sage CA: Thousand Oaks, CA, v. 18, n. 1, p. 45–55, 1992.

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

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.

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.

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.

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>.

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.

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.

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.

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.

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.

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.], 2012.