# Utilizando Ontologias para apoiar a recomendação de revisores de código

Vinicius J. Schettino
Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais
vinicius.schettino@ice.ufjf.br

Regina Braga
Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais
regina.braga@ufjf.edu.br

Marco Antônio P. Araújo
Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais
marco.araujo@ufjf.edu.br

Victor Ströele
Universidade Federal de Juiz de Fora
Juiz de Fora, Minas Gerais
victor.stroele@ufjf.edu.br

## ABSTRACT

A prática de *code review* está amplamente associada a qualidade de código e detecção precoce de defeitos de software. Intrinsecamente colaborativo, a eficiência do processo depende também do perfil histórico do revisor e da natureza do código em avaliação. Em contexto de desenvolvimento distribuído, consolidar todas as informações que ajudam a encontrar o revisor adequado se torna ainda mais difícil, e a importância da recomendação automatizada ganha mais força. Dentre outros desafios, a arquitetura e habilidades necessárias entre projetos são distintas, atrapalhando a troca de experiências e auxílios entre diferentes grupos de trabalho. Para auxiliar no processo de escolha do revisor, uma ontologia de alinhamento é proposta, de forma a alinhar dos tópicos de conhecimento entre projetos distintos. Assim é possível encontrar especialistas que possam ser revisores adequados em diferentes grupos de tabalho. Este trabalho apresenta a arquitetura responsável pela extração dos dados e aplicação da ontologia proposta, exemplificado por um caso de uso envolvendo projetos OpenSource conhecidos de uma mesma organização.

## CCS CONCEPTS

•**Computer systems organization** →**Embedded systems;** *Redundancy;* Robotics; •**Networks** →Network reliability;

## KEYWORDS

ACM proceedings, LaTeX, text tagging

**ACM Reference format:**
Vinicius J. Schettino, Regina Braga, Marco Antônio P. Araújo, and Victor Ströele. 2019. Utilizando Ontologias para apoiar a recomendação de revisores de código. In *Proceedings of Simpósio Brasileiro de Sistemas de Informação, Aracaju/SE - Brasil, Maio 2019 (SBSI'19),* 13 pages.
DOI: xxxxxxx


*SBSI'19, Aracaju/SE - Brasil*

DOI: xxxxxxx

## 1 INTRODUCTION

O *code review* é considerado uma das principais técnicas para diminuição de defeitos de software [12]. Nela, o autor de uma alteração na base de código de um projeto submete tal conteúdo ao crivo de um conjunto de pares técnicos, que irão revisar sua estrutura com base em um lista de regras e convenções previamente definida. Diferentes aspectos relacionados ao autor, ao revisor e ao processo de revisão em si estão diretamente relacionados à eficiência da prática. Autores relatam a diminuição da incidência de *anti-patterns* [30] de acordo com o nível de participação dos envolvidos e cobertura do código revisado [6, 33, 36]. Reputação [7, 14] e experiência [31] 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 [3], principalmente no desenvolvimento distribuído. Dentro de workflows de trabalho descentralizados [23], a prática funciona como um *gateway* de qualidade que busca garantir que apenas alterações aderentes aos padrões de qualidade do projeto serão incorporados à codebase principal. Esta etapa do desenvolvimento se torna uma oportunidade para disseminação de conhecimento, embate de ideias e discussão de melhores práticas entre profissionais de experiência e visões diferentes. Para tanto, percebe-se a necessidade de suporte computacional para essas atividades colaborativas.

Tais aspectos configuram o Desenvolvimento Distribuído de Software (DDS), onde equipes de desenvolvimento se encontram espalhadas por organizações e espaços geográficos distintos. Este novo ramo da Engenharia de Software vem modificando a relação entre empresas e sistemas, principalmente em relação às estratégias de negócios [2]. 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.

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

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

Existem alguns trabalhos congêneres que demonstram métodos de recomendação de revisores [29, 48, 52]. 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 possivelmente potencializam a colaboração [7, 11, 30].

Neste contexto, porém, os os desafios à colaboração co-localizada são potencializados e as soluções tradicionais não são suficientes para fomentar esta aspecto das atividades distribuídas [37]. Casey [16] 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. 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 [21].

Outro ponto observado diz respeito à avaliação dos modelos de avaliação. Os trabalhos encontrados se limitam a comparar seus resultados com métricas relacionadas à proximidade dos mesmos com a indicação manual do revisor. Ou seja, a eficência é tida de acordo com a interseção entre o recomendado automaticamente e por decisão de um especialista, geralmente um desenvolvedor. Este modelo assume que o responsável pela indicação manual tem os subsídios naturais para fazer uma boa escolha. Em DDS isso pode não ser verdade, uma vez que fatores como diferenças culturais, de horário, geográficas e de maturidade podem diminuir a compreensão do indicador e propiciar a escolha inadequada do revisor. Por isso, no contexto apresentado, outras formas de avaliação podem ser mais apropriadas. Trabalhos recentes em sistemas de recomendação atentam para a importância de outras métricas de avaliação, como transparência, desempenho e feedback dos usuários [26].

Os métodos anteriormente propostos utilizam dados de expertise e reputação dos desenvolvedores para apontar os mais adequados para determinado conhecimento. São utilizados dados de proximidade de código ou de semântica com as revisões anteriores. Abordagens mais recentes de recomendação de revisores envolvem informações de colaboração, como por exemplo impacto do trabalho entre um revisor/autor no passado. Estas relações são geralmente modeladas como redes de colaboração e construídas através de análises semânticas de similaridade e frequência de colaboração entre determinados atores. Fu et al. [20] descreve uma recomendação baseada num grafo de relacionamento entre desenvolvedores e seu status de trabalho no projeto. Xia et al. [49] agrupa desenvolvedores utilizando métodos de proximidade para capturar relações implícitas e explícitas. Yu et al. [51, 52] estendem essa abordagem com métodos de aprendizado de máquina minerando os comentários para extrair metadados e enriquecer o modelo da rede colaborativa. Yang et al. [50] utilizam social network analysis (SNA) baseadas em informações de atividade recente dos potenciais revisores no projeto.

Como trabalhos recentes na área utilizam aspectos de redes colaborativas, buscamos aprofundar análises destas abordagens para propor novos métodos de recomendação que possam aumentar a eficiência da recomendação de revisores dentro do contexto escolhido.

Para aprimorar as abordagens colaborativas citadas, a simples rede colaborativa pode ser incrementada com informações relevantes, aumentando a eficiência de revisão e dos métodos de avaliação. Uma das formas reunir estas informações relevantes é através do uso de ontologias [34, 35]. A informações descobertas através das máquinas de inferência podem aumentar a eficiência do modelo de recomendação.

O uso de ferramentas computacionais para o processo de revisão de código se tornou prática comum nos últimos anos [3]. O GitHub é uma plataforma rica em repositórios de projetos de software. Muitos são de código aberto, disponíveis para mineração. 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*" [22].

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.

Com as informações contidas nos repositórios cada projeto, é possível aplicar diversas técnicas para detecção de especialistas em colaboração. Por exemplo, técnicas de clusterização podem detectar comunidades de trabalho com indivíduos centrais responsáveis por grande parte da colaboração. É possível também buscar indivíduos com os maiores índices de contribuição em cada tópico de conhecimento, seja em revisão, desenvolvimento ou documentação.

Contudo, as técnicas supracitadas encontram especialistas em projetos específicos, sem apontar quais são as capacidades de contribuição destes em projetos distintos. Por exemplo, como descobrir a capacidade do desenvolvedor A para auxiliar uma revisão no projeto B?

Este tipo de auxílio se faz importante dentro de organizações maiores. É o caso da Apache Foundation, com dezenas de projetos com diferentes linguagens, arquiteturas e propósitos. É possível que desenvolvedores que trabalham em projetos específicos tenham conheciementos aplicáveis em projetos distintos, apesar de sempre ter contribuído em apenas um deles. A quantidade de projetos, desenvolvedores e de características intrínsecas de cada um destes é uma barreira para detecção e recomendação de especialistas entre repositórios distintos.

Assim, este trabalho propõe uma ontologia para alinhar os tópicos de conhecimento entre projetos distintos. A contextualização destes tópicos de forma homogênea possibilita a avaliação de um determinado especialista em um projeto dentro das necessidades de um

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

**Tabela 1: Tabela do Template (para comparar)**

| Non-English or Math | Frequency | Comments |
|---|---|---|
| uai | 1 in 1,000 | At oprette |
| $\pi$ | 1 in 5 | Common in math |
| $ | 4 in 5 | Used in business |
| $\Psi_1^2$ | 1 in 40,000 | Unexplained usage |

outro. Através do *matching* das labels de diferentes projetos, é possível inferir a importância que um determinado desenvolvedor teria em um projeto que ele ainda não contribuiu.

Labels são *tags* atribuídas aos pullrequests, com diversos objetivos. Existem labels referentes a módulos de um projeto ()$mod_a$), conhecimentos (*crypto, performance*), linguagens (*python, php*) e *workflow* (*ready, in-progress*). A utilização desta e outras ferramentas do GitHub são detalhadas na seção 2.2

Além deste capítulo, este trabalho é composto de outras 4 partes. A seção 2 traz as fundamentações e conceitos necessários para a compreensão dos objetivos, motivação e métodos aplicados. O capítulo 3 elucida os principais trabalhos congêneres, tanto no tocante da recomendação como das ontologias. O *worflow* de extração, a ontologia proposta e as possibilidades de extensão dos trabalhos anteriores são evidenciados no capítulo 4. Por fim, a conclusão da pesquisa é descrita no capítulo 5

# 2 REFERENCIAL TEÓRICO

Este capítulo descreve os diferentes pressupostos teóricos necessários para entendimento do processo de recomendação de revisores. O contexto de desenvolvimento distribuído possui diversas variações quanto a processo e ferramentas, que são descritas nas seções seguintes.

## 2.1 *Code review*

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

*2.1.1 Relevância.* O *code review* está associada diretamente à detecção precoce de defeitos em produtos de software [30, 42], sendo reconhecida como uma das principais técnicas com este fim [12]. Mais especificamente, é relatada maior eficiência quanto aos defeitos não-funcionais, enquanto os defeitos funcionais são menos afetados no processo [9]. Outros autores reportam a diminuição de defeitos através de estudos de caso [6, 32, 36].

*2.1.2 Histórico.* A atividade de revisão remonta da década de 80 [17], 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 [3].

## 2.2 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 [5].

Estas tecnologias permitiram o surgimento de um paradigma de desenvolvimento baseado em pulls, ou *pull-based method* [22]. 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 [24]. 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 [8], 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*.

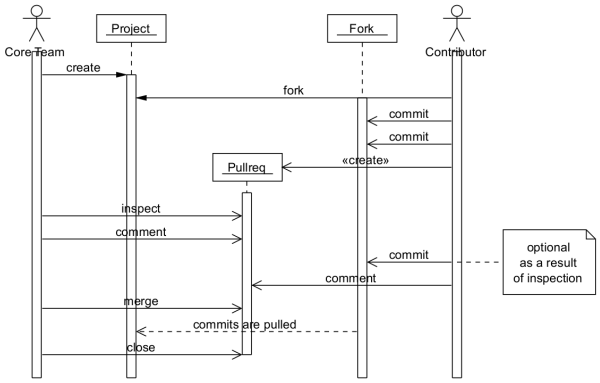


**Figura 1: Pull Request Process [22]**

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 [23]. Ao terminar, ele solicita a integração da branch do *fork* de volta ao projeto original. Essa solitação é chamada de *pull-request*, que será analisada por um desenvolvedor com as devidas permissões. Durante esta revisão, o autor pode gerar novas modificações, geralmente atreladas aos pedidos do revisor. Ao final, a mudança é rejeitada (*closed*) ou aceita *merged*.

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

Os membros do core também têm suas *branches* revisadas por um processo análogo [8, 14]. 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.

### 2.3 Ontologia

Ontologia é a definição de um vocabulário ou linguagem comum para representar conhecimentos [25]. Para computação, pode ser vista como uma especificação formal, explícita e não ambígua de um conceito compartilhado por diferentes agentes.

As ontologias são utilizadas em diversos contextos e aplicações, geralmente com o objetivo de:

- Compartilhar um conceito entre componentes de software.
- Reutilizar conceitos de um domínio em outro.
- Alinhar conceitos distintos propondo uma unificação explícita das diferentes visões.

Uma ontologia é composta por: Definições de classes; Relações e Funções. Se tornaram comuns para representação de conhecimento na Web Semântica [10]. As especificações da ontologia permitem a utilização de máquinas de inferências para descoberta de novos conhecimentos, antes não explícitos, nas informações disponíveis. Estas máquinas inferem propriedades e relacionamentos baseadas em conhecimento explícito [10]

## 3 TRABALHOS RELACIONADOS

Existem diversos trabalhos que visam recomendar revisores, alguns já citados em seções anteriores deste texto. Para esta seção foram escolhidos trabalhos fundamentais do campo, com atenção a diversidade de abordagens, especialmente no tocante da avaliação e métodos computacionais. O objetivo desta metodologia é oferecer uma visão ampla da área para identificar desafios e boas práticas que devem ser avaliados para proposta de novos métodos.

Balachandran [4] foi um dos primeiros autores a discutir a relevância da recomendação de revisores de código para o *code review*. Ele propõe uma ferramenta não apenas para realizar uma série de análises para indicar modificações no código de maneira automatizada, mas também para encontrar o melhor revisor baseada em informações do *changeset* em análise. A proposta também reconhece que em projetos muito grandes, com muitos contribuidores, responsabilidades e código, é difícil que um ser humano possa, sem suporte, consolidar todas as informações necessárias para escolher um bom revisor.

É proposto então o ReviewBot, que utiliza as linhas alteradas no código e busca revisores que já atuaram no mesmo lugar. Estas informações são utilizadas para o revisor adequado, baseado em experiências passadas com aquele componente.

CoRReCT [41] é uma ferramenta que também utiliza o histórico dos potenciais revisores para predição. Contudo, as informações em questão tangem a história dele como desenvolvedor. O especialista é definido pelas tecnologias com as quais ele trabalhou no passado, definidas como tópicos de conhecimento. A principal inovação da proposta é utilizar o histórico de todas contribuições OpenSource feitas pelo desenvolvedor.

Para apoiar esta hipótese, foi conduzida uma análise exploratória. Os resultados indicam que várias bibliotecas e tecnologias são utilizadas em diversos projetos simultaneamente, e que por isso um especialista de um projeto pode ser de grande ajuda em outros também. É criada uma rede baseada em tópicos de conhecimento, para descoberta da expertise dos desenvolvedores em cada assunto.

cHRev [53] é outra abordagem que utiliza o histórico dos potenciais revisores. Contudo, os autores substituem o papel da similaridade do código por observar a utilizade de suas participação em revissões passadas similares.

Esta análise é feita com base em três fatores: quantos comentários foram feitos, qual foi a velocidade de resposta e quão recente foi a revisão. Estas métricas servem para indicar o quão ativo foi o revisor anteriormente, fazendo dele alguém adequado para revisão em questão.

RevRec [38] foca na expertise do desenvolvedor e analisa também as ocasiões de colaboração passada entre o desenvolvedor e o autor. A premissa é que uma relação passada positiva facilita a colaboração e pode evitar revisões ineficientes e tarefas manuais e custosas no processo.

Além disso, eles hipotetizam que é comum que autores busquem pares de revisão com os quais já trabalharam anteriormente para evitar conflitos. Por exemplo, evitam que criticas ou rejeições sejam interpretados como ofensas. Além de utilizar informações de atividade e utilidade do revisor em iterações passadas, esta abordagem busca analisar revisões semanticamente parecidas para recomendar.

Alguns dos trabalhos mais recentes da área levam em consideração informações tidas como relacionadas à colaboração para recomendar os revisores adequados[38, 53]. Por ser uma atividade fundamentalmente colaborativa, a revisão de código depende da capacidade de interação entre os envolvidos, estando vulnerável aos desafios de colaboração que se intensificam no desenvolvimento global de software.

A principal inovação da proposta aqui descrita é o uso de ontologias para recomendar especialistas entre projetos distintos. A utilização de ontolgias para recomendação é uma técnica já explorada em trabalhos anteriores [34, 35] e indicada como sugestão para trabalhos futuros [1], e por isso é um potencial caminho para otimização dos resultados da recomendação.

## 4 PROPOSTA

Este capítulo descreve *workflow* da proposta de rede colaborativa para recomendação, partindo da e extração, transformação e carga dos dados dos dados até a disponibilização destes através de serviços para execução dos métodos para encontrar os revisores adequados.

### 4.1 Extração dos Dados

Os dados da análise são extraídos do GitHub através da API[5] disponibilizada para desenvolvedores do mundo todo estenderem as funcionalidades da plataforma e integrar novos produtos que agregem valor ao processo de desenvolvimento. A versão estável para integração foi a v3, que implementa o padrão arquitetural RESTful[18]. Existe a API em recém lançada v4[6], que utiliza da especificação GraphQL para exposição dos serviços. Esta não foi utilizada devido a documentação ainda recente e pela preferência ao padrão REST. A figura 2 mostra o pipeline de extração e estruturação dos dados.

[5] https://developer.github.com/v3/
[6] https://developer.github.com/v4/

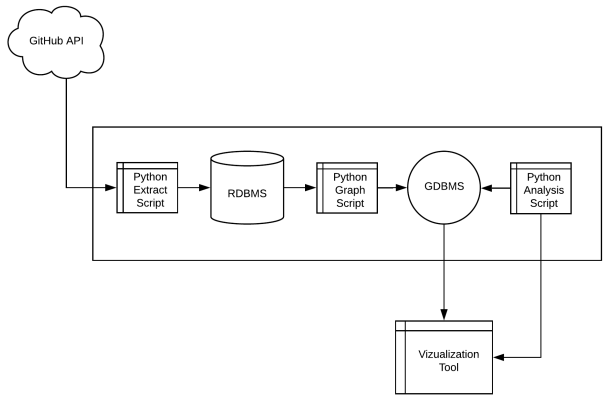


**Figura 2: Processo de extração dos dados**

Os dados extraídos são estruturados em um esquema relacional, mais especificamente em uma instância PostgreSQL 10.5. Este passo intermediário ocorre para que os dados fiquem disponíveis para outras análises, pesquisadores e ferramentas familiarizadas com o modelo relacional. Além disso, esta abordagem permite a extração de diversos metadados que não serão utilizados nas análises preliminares executadas na estrutura baseadas em grafo, ficando disponíveis para consultas posteriores sem degradar a performance do modelo em análise. A figura 4 mostra o modelo relacional proposto. As entidades *user* são normalizadas entre os diferentes projetos. Ou seja, o usuário é uma entidade única que se relaciona a diferentes projetos, sem duplicação de seu conteúdo. Esta caracterísitca facilita futuras análises de colaboração entre projetos distintos. As *labels* ligadas aos *pullrequests* são utilizadas como descritores semânticos do conteúdo da contribuição, como por exemplo *documentação*, *python* ou *performance*. As contribuições podem ser feitas através de comentários thread de discussão do *pullrequests*, representados pelos *discussioncomments* ou através de comentários no código em revisão, nos *reviewcomments*. As *reactions* são expressões de sentimento, ou *emojis* utilizados para representar a opinião de maneira mais lúdica, como mostra a tabela 3.

| content | emoji |
|---|---|
| +1 | 👍 |
| -1 | 👎 |
| laugh | 😄 |
| confused | 😕 |
| heart | ❤️ |
| hooray | 🎉 |

**Figura 3: Tabela de Reactions Disponíveis**

Os dados são carregados para o banco de dados baseado em grafo (GDBMS), com as informações relevantes nos nós e arestas. Os dados de interesse são os autores e revisores, bem como o relacionamento entre eles. Detalhes da modelagem da rede colaborativa serão cobertos em uma seção específica. A tecnologia escolhida aqui é o Neo4j, banco de dados orientado a grafo mais popular de sua família[7]. É a partir dele que as análises são executadas e vizualizadas utilizando o Gephi[8].

O algortimo de extração dos dados foi projetado com uma série de cuidados para garantir a integridade e disponibilidade das informações, diante das nuances do repostiório e do domínio de dados. Os principais aspectos observados são:

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

*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.

*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.

*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.

## 4.2 Aspectos de Reprodutibilidade

Cada uma das tecnologias apresentadas é encapsulada através da tecnologia de virtualização Docker. esta abordagem é uma resposta às iniciativas de reprodutibilidade na ciência, buscando maior transparência, confiabiliade e possibilidade de extensão nos experimentos [19]. 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 5 Estes são orquestrados através do docker-compose, 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 minúncias que estão envolvidas neste tipo de experimento, o código fonte e a descrição ainda que detalhada dos dados não é suficiente para

[7] https://db-engines.com/en/ranking
[8] https://gephi.org/

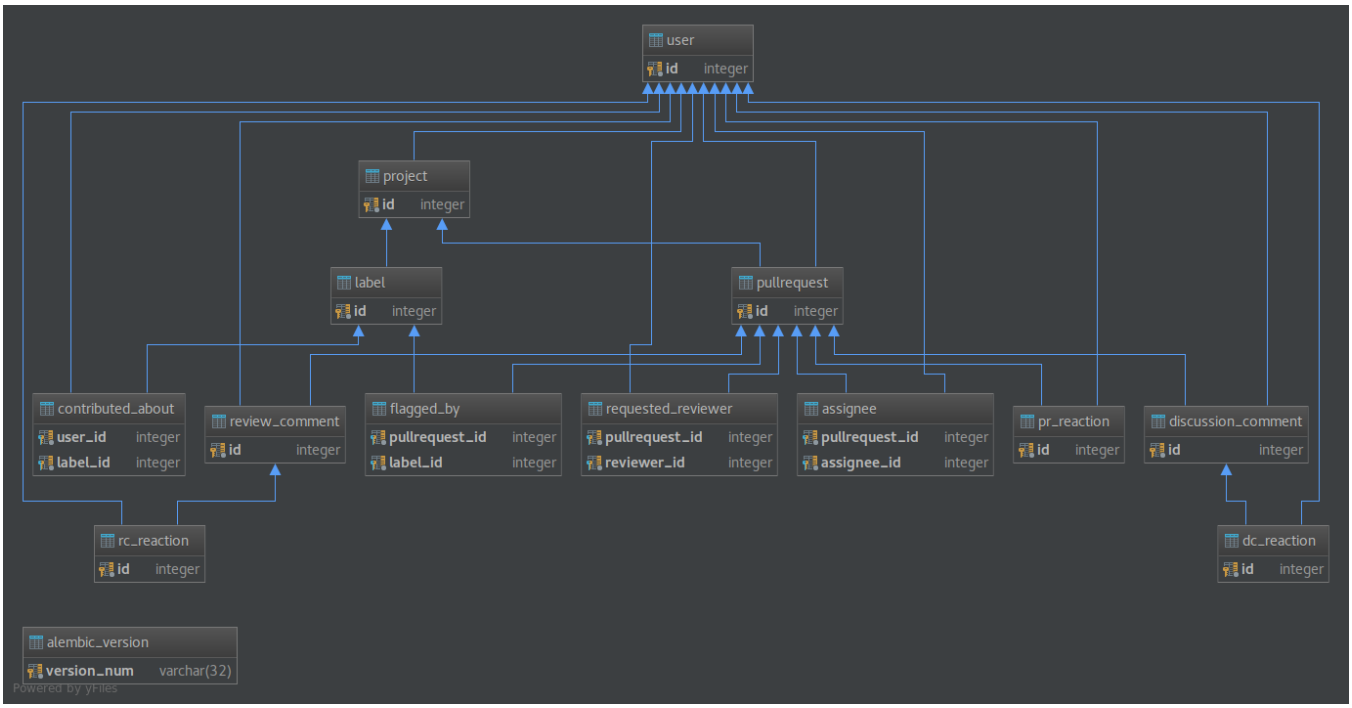


**Figura 4: Modelo Entidade Relacionamento**

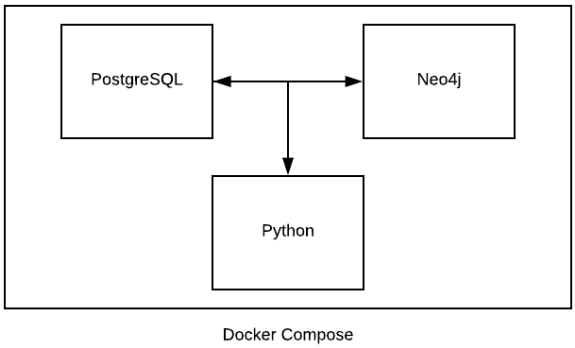


**Figura 5: Modelo de contâiners**

alcançar níveis adequados de reprodutibilidade [28]. 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 [13].

De acordo com Sinha et al. [46], 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 6 mostra o espectro de reprodutibilidade, que encara esta característica como um constante trabalho de evolução não binário, podendo uma pesquisa se tornar mais ou menos reprodutível ao se utilizar determinadas técnicas. A figura 7 mostra o detalhamento de cada aspecto que contribui para que esta característica seja evidenciada:

*Dados - Evidências Primárias/Secundárias:* Análise da disponibilidade dos dados utilizados para futuros pesquisadores. Pode variar de simplesmente não dispobilizado até a disponibilização íntegra das informações, valendo-se de meios para uma oferta não repudiável e com garantias de integridade e veracidade. Neste trabalho, todos os dados (em suas diferentes agregações) são disponibilizados para uso futuro. Além disso os mecanismos de extração são automatizados, permitindo buscar outros projetos ou atualizar os dados dos já utilizados.

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

*Código Fonte:* Disponibilidade do código fonte utilizado nas análises. Pode variar de não disponível (proprietário) até open source com direito de extensão e modificação. Neste projeto todo o código fonte é opensource e disponibilizado sob a permissiva licença MIT.

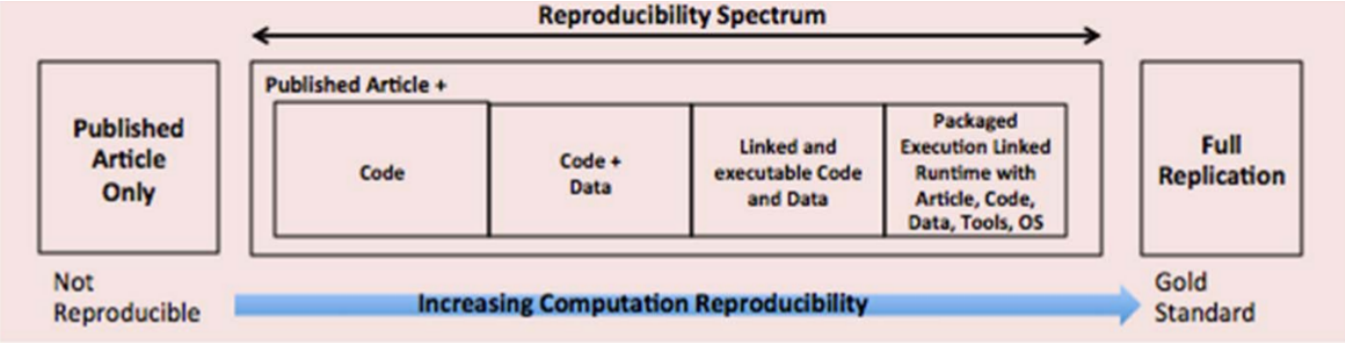


**Figura 6: Espectro de Reprodutibilidade**

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

**Figura 7: Escala de Maturidade de Reprodutibilidade**

*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. Informações do hardware utilizado nos experimentos serão disponibilizadas nas próximas seções.

## 4.3 Modelo de Rede Colaborativa

O modelo proposto se baseia nos dados de relacionamento entre autores e revisores, mais especificamente nas interações entre comentários de revisão e pullrequests. Quanto existe uma interação do tipo, cria-se ou atualiza-se uma aresta entre dois indivíduos. A figura 8 ilustra o modelo proposto.

O peso de cada aresta varia entre (0,1] e representa a importância que um revisor tem sobre determinado ator em determinado tópico de conhecimento. O tópico de conhecimento é definido através das labels de cada pullrequest. O cálculo é feito através da fórmula 1.

$$W(A, B, label) = \frac{countRevs(A, B, tag)}{countTotRevs(B, tag)}. \quad (1)$$

Para calcular o peso total do revisor A no autor B em determinada label, basta dividir a quantidade de revisões que A fez para B pelo total de revisões que B recebeu naquela label. Caso A tenha sido o único revisor naquele assunto, terá o peso máximo sobre ele neste contexto. Assim, este peso pode ser caracterizado como o nível influência que A têm sobre B.

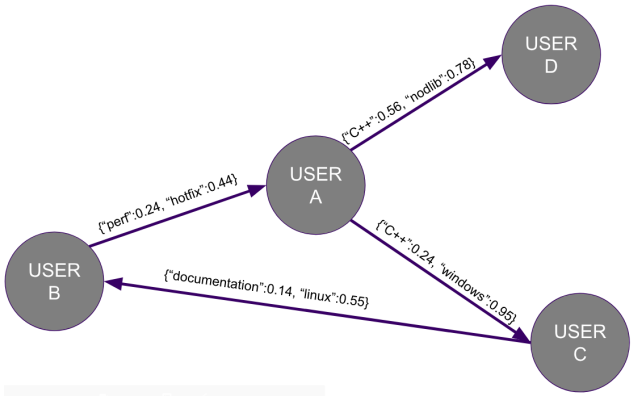


**Figura 8: Modelo da Rede Colaborativa**

O fórmula 1 representa o nível de influência entre os diversos indivíduos que colaboraram no projeto. Assim, revisores que foram responsáveis pela maior parte das revisões de um determinado autor terão pesos mais altos nestas relações. Contribuidores que foram revisados por muitos especialistas durante suas contribuições terão influencias individuais diluídas, e por isso apresentarão pesos de entrada mais baixos. O objetivo é utilizar os pesos para representar a influência e a reputação de cada revisor.

Para facilitar a análise de métricas gerais de redes complexas, é introduzido ainda o peso médio de A sobre B, independente das tags. Ele é dado pela fórmula 2.

$$avgW(A, B) = \frac{countRevs(A, B)}{countTotRevs(B)}. \quad (2)$$

Este valor é utilizado para análises preliminares, por englobar um cenário geral sobre a relação entre os desenvolvedores.

## 4.4 Escolha dos repositórios de dados

Para avaliar as diferentes propriedades das redes colaborativas propostas, é necessário definir um conjunto de dados inseridos no contexto de estudo. As principais características buscadas foram:

*Repositórios de GSD:.* Como a recomendação de revisores têm maior relevância em desenvolvimento global de software, os repositórios escolhidos devem ser representantes deste contexto de desenvolvimento.

*Acesso aberto:* Os repositórios devem estar disponíveis para uso em pesquisa e para futuras reproduções e extensões do estudo. Além disso há a restrição de estarem no GitHub para acesso via API.

*Popularidade e Escala:* Os repostórios devem ser razoavelmente conhecidos e possuir um número de contribuidores que justifique a necessidade de recomendação de revisores. Para isso foram escolhidos projetos integrantes do top-10 "projects with the most reviews"eleitos pelo próprio GitHub[9]

A tabela 2 ilustra os projetos escolhidos e informações de contexto e tamanho.

## 4.5 Integração de repositórios de dados

Apesar das informações encontradas no GitHub, é possível refinar métodos de agrupamento e recomendação utilizando dados oriundos de outras fontes. Esta seção descreve a proposta de integração entre diferentes banco de dados para extrair informações relevantes para o modelo de rede colaborativa utilizado.

O StackOverflow é o maior site de QA (question and answer) aberto da web. A plataforma é anfitrã de milhões de perguntas e respostas, organizadas por tags administradas pela comunidade e por moderadores, sendo assim uma organização confiável quanto a relevância desta distribuição. Além disso, existem informações da pontuação dos envolvidos, bem como das respostas e perguntas realizadas. Este tipo de informação serve de proxy para descrever reputação dos participantes das discussões, especialmente inseridos no contexto de um conhecimento específico, como uma linguagem de programação ou framework.

Estas informações podem estender o poder de recomendação das abordagens baseadas na rede colaborativa apresentada, especialmente:

- Refinando os resultados em caso de empates, através da Reputação.
- Refinando os agrupamentos baseado em mais informações de conhecimento, facilitando a colaboração.
- Mitigando problemas de *cold start*, angariando mais informações sobre novos autores.

Contudo, tal integração não é trivial. Em bancos de dados distribuidos, diversos desafios são referenciados na literatura e impactam diretamente na eficiência e desempenho da integração [39], notóriamente:

*Meios de acesso heterogêneos:* Mesmo quando os dados estão semanticamente alinhados, os meios de acesso geralmente são distitos: APIs (RESTful, SOAP, etc), arquivos (XML/JSON) e SGBDs (relacional ou não, tecnologias distintas) raramente são facilmente compatíveis. Soluções de integração envolvem *wrappers* que traduzem as nuances de cada repositório para uma interface homogênea que será acessada para as consultas globais.

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

*Dados semanticamente conflitantes:* Muitas vezes entidades análogas são representadas com nomenclaturas diferentes, e o inverso também pode acontecer. Existem diferentes tecnicas para alinhamento destes domínios desde uma verificação manual até utilizando estruturas de mediação, como as baseadas em ontologias.

*matching dos dados:* Mesmo quando há plena ciência, acesso e entendimento às diferentes bases, entidades análogas não necessariamente possuem dados idênticos. Pode haver falta de alguns dados ou até mesmo dados conflitantes. Para endereçar este problema, é necessário um processo de matching que pode utilizar diferentes abordagens, que varia desde da comparação de atributos chave até a utilização de lógica *fuzzy* para indicar tuplas correspondentes.

Outra discussão relevante é a forma de integração dos dados, que geralmente é divida em duas formas : física ou lógica[39]. O formato lógico consiste em disponibilizar uma interface homogênea virtual, onde as consultas globais são executadas. Estas são traduzidas e enviadas em tempo real para cada fonte de dados, e as respostas são integradas e apresentadas. Assim não existe união física das bases, e sim uma interface de consulta que permite consultas espalhadas nas diferentes fontes. Os principais problemas são as restrições de desempenho e a complexidade de distribuição das queries.

Outra abordagem consiste na consolidação dos dados distribuidos em uma única base, através de um processo conhecido como ETL (Extraction-Transform-Load). Os dados não são integrados em tempo real, mas sim em intervalos pré definidos. Esta característica faz com que o desempenho desta abordagem geralmente seja superior à integração lógica, e por isso é a base da maior parte dos datawarehouses e aplicações OLTP.

Buscando evitar a complexidade da distribuição de queries e visando flexibilidade nas transformações e no volume de dados em questão, optou-se pela abordagem física. Outros pesquisadores realizaram este tipo de integração com esta abordagem anteriormente, como descrito a seguir.

Para entender o impacto do StackOverflow no desempenho dos desenvolvedores, Vasilescu et al. [47] integram as bases para buscar correlação entre atividades nas plataformas e importância dos usuários. Para integrar as bases, sob a premissa de evitar falsos positivos, foi utilizada a técnica de correspondência completa entre emails das entidades "usuários"em ambas as bases de dados. A técnica deixa de fora usuários com mais de um email, reduzindo bastante o universo de dados resultante. Os repositórios dos dados são arquivos extraídos de forma offline de ambas as plataformas, criando um experimento que não pode ser facilmente atualizado ou a confecção de ferramenta que pode ser utilizada em tempo real.

De forma mais rebuscada, Silvestri et al. [45] propõe uma integração entre três redes sociais: Twitter, StackOverflow e GitHub. Apesar de utilizar arquivos offline semelhantes à trabalhos anteriores (com adição da API REST do Twitter), o modelo de *matching* é mais completo e tem potencial de encontrar mais correspondências corretas. Além de verificar o email, o perfil dos usuários é vasculhado para buscar referências aos perfis de outras redes. Além disso, informações como o login são utilizados tanto em correspondência direta quanto em correspondência parcial. A foto dos usuários também passa por um processo *fuzzy* para indicar a probabilidade dos usuários serem os mesmos.

| Nome | Linguagem Principal | Watchers | Stars | Contribuidores | Pull Requests |
|---|---|---|---|---|---|
| Node.js | JavaScript | 2.868 | 52.709 | 2.088 | 8.440 |
| TensorFlow | C++/Python | 8.293 | 108.249 | 1.624 | 8.562 |
| Kubernetes | Go | 2.675 | 40.610 | 1.776 | 41.059 |
| Symfony | PHP | 1.297 | 18.381 | 1.692 | 17.475 |

**Tabela 2: Descrição dos Respositórios Selecionados**

A abordagem aqui proposta une os principais aspectos positivos dos casos apresentados, adaptando aos objetivos da pesquisa e mitigando os principais defeitos. A primeira diferença é que no caso aqui descrito, existe uma base hierarquicamente mais relevante, de onde toda a informação estrutural da rede é extraída. No GitHub reside a maior parte da informação necessária, e por isso a extração é feita a partir dele. A integração com o StackOverflow é feita de maneira posterior e secundária, visando enriquecer as informações disponíveis para as análises.

Outro desafio que esta proposta precisa lidar de maneira mais eficiente é a atualização dos dados. Os modelos anteriores se baseiam em repositórios offlines que contém todas as informações de todos os projetos do GitHub. O objetivo aqui é propor uma ferramenta de recomendação viável, que possa ser recarregada com dados atualizados em tempo razoável para que recomendações precisas sejam feitas.

Assim, o modelo proposto extende a abordagem apresentada na seção 2, adicionando na extração dos dados do GitHub a busca no repositório de dados do StackOverflow, através da API[10] que implementa a interface SQL BigQuery[11]. As informações de cada usuários são então aprimoradas, como mostra a figura 9.

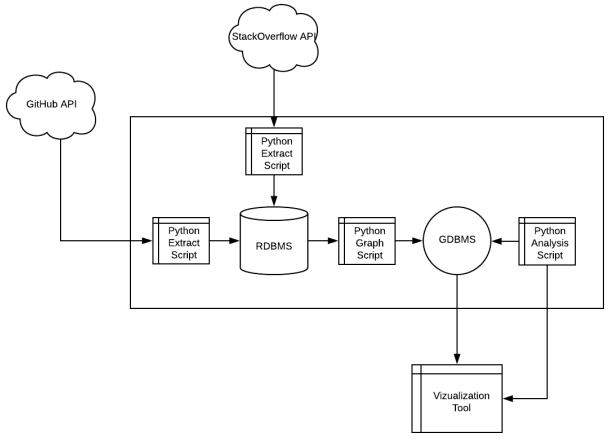


**Figura 9: Processo de Extração com StackOverflow**

Para o processo de matching, além das informações de email utilizadas anteriormente, pode-se utilizar a busca por referências ao perfil do StackOverflow no perfil do GitHub. Além disso o nome pode ser utilizado para *matching* parcial e comparação com o login. Por exemplo, inferindo que *John Smith* pode ser *jsmith*. Para aumentar a precisão destes algoritmos, é possível também restringir as buscas aos usuários que façam menção ao projeto em processo de extração em seu perfil.

Dentre os principais dados relevantes para o processo de recomendação, estão aqueles relacionados a conhecimento e reputação dos usuários que estão também no StackOverflow. Os principais são:

*informações de reputação:* É possível entender a presença do usuário no StackOverflow através da sua pontuação. Este *score* é obtido através de diferentes formas de colaboração, como perguntas, respostas, reações, comentários e outras formas de interação. Usuários mais importantes para a rede tendem a ter pontuações mais altas. Este ranking pode ser utilizado para desempatar recomendações.

*informações de conhecimento:* Analisar a contribuição dos potenciais revisores em determinadas áreas do conhecimento. Este rankeamento pode ser feito pelas tags do StackOverflow e serem comparadas com as informações semânticas de seu histórico no GitHub, como as labels ou tópicos extraídos do conteúdo dos pull-requests.

Outra forma de integrar as informações presentes nos repositórios, garantindo o alinhamento semântico e a completude destas é através do uso de ontologias de integração [40]. Assim é possível modelar consultas e extrair conhecimentos de forma abstrata, onde os conceitos são representações não ambíguas das entidades de cada um dos respositórios [15]

## 4.6 Recomendando Especialistas Entre Projetos

Especialistas de um projeto podem ser detectados de diversas formas. Analisando a atividade recente, por exemplo, é possível encontrar os desenvolvedores responsáveis pelas principais contribuições. Estas contribuições podem ser revisões, modificações no código, documentação e tradução dos artefatos.

Contudo, o intercâmbio destes especialistas entre projetos distintos é um desafio. Isso porque desenvolvedores podem ser especialistas em determinadas atividades ou tecnologias, e indentificar semelhanças de demandas entre projetos não é trivial. No GitHub, as labels dos pull requests tem como objetivo esta caracterização da modificação que está sendo feita, e por isso ajuda a identificar o tipo de conhecimento que o autor e o revisor demonstram ter influência. Ainda assim, as labels são particulares de cada projeto e podem ser editadas por seus mantenedores, o que reforça o desafio de buscar especialsitas que podem ser úteis em outros projetos.

[10] http://data.stackexchange.com/stackoverflow/queries
[11] https://cloud.google.com/bigquery/public-data/stackoverflow

Para atacar esta barreira, este trabalho propõe o uso de ontologias para alinhar as tags de diferentes projetos. O objetivo é que desenvolvedores conceituados dos projetos alvos do alinhamento tenham uma infraestrutura para documentar e expressar a relação entre as diversas labels dos projetos. Assim, é possível que especialistas encontrados em determinado projeto possam ser recomendados como especialsitas em outros projetos, de acordo com a equivalência das labels envolvidas.

*4.6.1 Ontologia Proposta.* A ontologia proposta pode ser visualizada na figura 10. Apenas duas entidades são criadas, *Project* e *Label*. Entre elas uma Object Property *owns* que denota que determinada label originalmente é originária de determinado projeto.

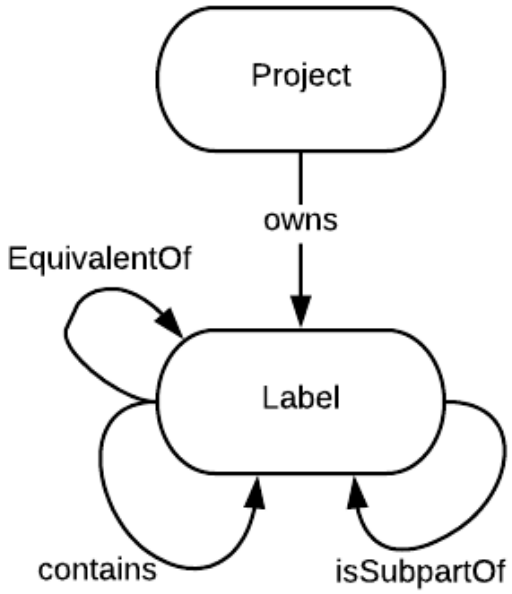


**Figura 10: Visualização da Ontologia Proposta**

As diversas Object Properties que relacionam as Labels denotam relacionamento semântico entre elas. Enquanto *contains* siginifica que a *Label* de destino representa parte do conteúdo denotado pela entenidade de origem, *isSubPartOf* traduz o sentido inverso e pode ser inferida pelos *reasoners*. Já *equivalentOf* denota uma correspondência completa entre duas entidades.

Através desta ontologia, é possível expressar os diferentes tipos de equivalência entre labels de projetos distintos. Assim, é possível inferir especialsitas de um projeto que podem contribuir em outro, dentro de suas especialidades já identificadas. Este alinhamento é o principal diferencial da abordagem baseada em ontologias, pois:

- Oferece uma interface direta para consulta das *Labels* já alinhadas, sem necessidade de processamento custoso em tempo real.
- Permite a utilização de Object Properties para descrever os relacionamentos de forma completa, onde o SGBD se limitaria a relacionamentos triviais como *"de..para"*.
- A estrutura é facilmente extensível para criar novos tipos de relacionamento entre as *labels* sem mudanças na estrutura relacional.

As próximas subseções mostram como a ontologia proposta pode ser populada e um estudo de caso para averiguar a utilização prática dos métodos descritos.

*4.6.2 Populando a Ontologia.* Como descrito em seções anteriores, os dados são extraídos dos repositórios distribuidos e consolidados em uma base de dados relacional. Este é o ponto central de onde outras transformações são executadas, como a criação da rede colaborativa. Para popular a ontolgia proposta, é necessário acessar este repositório central. A figura 11 mostra o processo de extração para disponibilizar uma interface SPARQL para prover as informações necessárias para os serviços de recomendação.

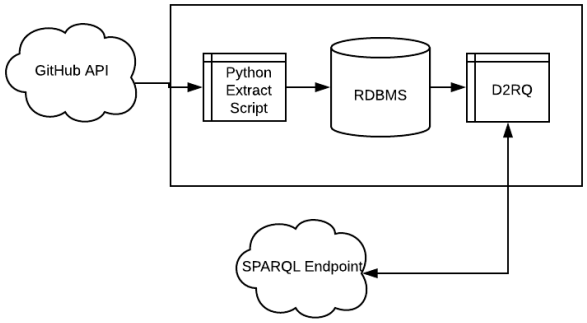


**Figura 11: Workflow de Extração**

O script de extração em Python fornece mecanismos para o alinhamento proposto das Labels, para o então fornecimento da interface SPARQL que será utilizada pelos metodos de detecção e recomendação de especialistas entre projetos. Assim como descrito na seção 4.5 que trata da integração de diversas bases, existem diversos métodos para buscar equivalência entre entidades oriundas de repositórios distintos. Algumas são baseadas na semântica ou na sintaxe dos descritores das entidades.

Todos os métodos apresentados servem para facilitar o processo de *matching* e evitar o trabalho operacional humano, mas para evitar a existência de falsos positivos, deve ser supervisionados por um especialista, que deve interferir quando achar necessário. Existem outros métodos que podem ser testados no futuro, extendendo o poder de alinhamento automático e auxiliando o processo com mais eficiência. Os principais métodos são:

- Direto, buscando *Labels* com descritores idênticos e fornencendo uma relação de *equivalentOf*.
- Parcial, buscando proximidade em detrimento de caracteres equivalentes
- Radicais, identificando abreviações de labels equivalentes
- Relação "todo-parte", onde *Labels* são identificadas como "sublabels" de outras

A tabela 3 mostra exemplos de alinhamentos que podem ser identificados automaticamente. Todas as saídas são sugestões de alinhamento para o especialista, que deve aprová-las para utilização posterior dos métodos de detecção e recomendação.

| Método | Exemplos de Labels Alinhadas | Tipo de Relação |
|---|---|---|
| Direto | "python"e "python" | *equivalentOf* |
| Parcial | "lib_src"e "lib-src" | *equivalentOf* |
| Radicais | "perf"e "performance" | *equivalentOf* |
| "Todo-Parte" | "python-crypto"e "crypto" | *contains / isSubPartOf* |

**Tabela 3: Alinhamento automatizado**

O D2RQ[12] é uma plataforma que provê uma interface de acesso às bases de dados relacionais como triplas RDF, acessíveis através de uma interface que interpreta consultas SPARQL. A conversão entre os modelos é feita nativamente para tipos mais comuns e baseada na ontologia proposta, e pode ser extendida através do desenvolvimento de classes que implementem o comportamento esperado. A conversão pode ser feita em tempo real ou criando um arquivo TTL (Turtle Triple Language) que solidifica todas as informações necessárias para posterior consulta.

Apesar do poder que as máquinas de inferência possuem, o desempenho desta abordagem ainda é um fator limitante de sua adoção geral na web semântica [44]. Quando se tratam de grande volumes de dados, os bancos de dados relacionais proveem mais escalabilidade, e por isso abordagens híbridas são aconselhadas [27]. Como os projetos opensource encontrados no GitHub possuem centenas de milhares de registros, testes preliminares apontaram pela inviabilidade da inferência através destas tecnologias. Assim, o D2RQ se mostrou como alternativa plausível que permite a especificação de serviços e modelos SPARQL aproveitando o poder da ontologia como linguagem de especificação, permitindo a utilização da infraestrutura relacional que dispõe de maior escalabilidade e desempenho.

*4.6.3 Avaliação da Ontologia Proposta.* Para uma avaliação preliminar da estrutura proposta, foi realizado um estudo de caso com dados reais de um projeto de relevância no GitHub. Buscando por repositórios com quantidades razoáveis de dados, utilizamos o top-5 oficial da plataforma para projetos mais revisados[13], selecionado aquele com mais estrelas. Esta funcionalidade do GitHub pode ser utilizada como medida para o tamanho da comunidade passiva de um projeto[43]. Assim, este critério nos levou a escolher o **Node.js**[14], uma popular engine JavaScript especialmente utilizada para aplicações *server-side* de alta vazão (I/O).

A organização responsável pelo projeto conta com outros 148 repositórios, a maioria deles bem menores e de alguma forma relacionados ao projeto principal. Para este estudo de caso foram escolhidos repositórios: o **llnode**[15] que mantém um plugin para debug do Node.js e o **build**[16] que administra o pipeline de deploy e integração contínua da tecnologia.

[12]http://d2rq.org/
[13]https://octoverse.github.com/
[14]https://github.com/nodejs/node
[15]https://github.com/nodejs/llnode
[16]https://github.com/nodejs/build

Os dados dos três projetos foram extraídos através do processo descrito anteriormente, e disponibilizados na na instância relacional. O script de alinhamento é executado, fornencendo sugestão de equivalência entre as diversas labels. Os resultados são descritos abaixo. Juntos, estes projetos reúnem mais de 200 labels, o que torna o alinhamento completamente manual lento e oneroso. Este aspecto potencializa a importância das técnicas automatizadas para suportar o processo.

Entre o projeto Node.js e o nodejs/llnode, diversas labels foram alinhadas de maneira direta: "V8 changes" e "feature request" por exemplo. A segunda é comum a diversos projetos e está mais relacionada ao workflow de desenvolvimento do que efetivamente um tópico de conhecimento, e pode ser negada pelo operador caso ele considere que este não seja um bom fator para recomendação de especialistas para projetos distintos. Utilizando o alinhamento parcial, as tags "flasky tests" e "flaky test" foram alinhadas. Esta equivalência é relevante pois identificar testes instáveis pode ser uma atribuição compartilhada dos projetos, pois pode estar relacionada à escrita dos testes ou à infraestrutura de execução.

Já entre Node.js e nodejs/build, houve alinhamento direto entre "tsc-review", que indica que o pull request deve ser avaliado pelo Technical Steering Committee do projeto. Outros alinhamentos como "enhancement" e "bug" foram identificados, e provavelmente não servem para detecção de especialistas entre projetos pois são comuns às atividades de software e não necessariamente a um tópico de conhecimento. Já a comparação por radicais apontou equivalência entre as labels "install" e "installer", que parecem ter funções parecidas nos projetos mas foram grafadas de forma distinta. Neste aspecto o alinhamento parece relevante, pois os principais revisores em cada uma destas labels são os mesmos em cada um dos projetos.

Com este estudo de caso foi possível observar que algumas equivalências automatizadas competem a tópicos muito generalistas, onde um especialista de determinado projeto não parece ter competência explícita para o ser em outro. Contudo, diversas equivalências parecem ser relevantes para os projetos selecionados, e por isso podem expressar relacionamentos que apoiam o processo de alinhamento manual. Estudos de caso posteriores podem ajudar a investigar se os especialistas têm performance equiparável quando contribuindo em outros projetos.

## 5 CONCLUSÃO

O *code review* é uma prática estabelecida com efeitos conhecidos e confirmados por vários autores através dos anos. Desde o processo tradicional que nasceu nos anos 70"baseado em um workflow rígido, a técnica se desenvolveu constantemente para o modelo leve e ágil que valoriza mais a troca de conhecimento e a relação entre pessoas.

Estudos recentes mostram o impacto da escolha do revisor correto na eficiência e impacto do processo na qualidade do código e na detecção precoce de defeitos. Isso fez com que diversos autores propossem métodos para recomendar revisores adequados, dado o contexto e o conteúdo de umaa revisão. Os desafios do desenvolvimento global de software potencializam a necessidade desta recomendação, visto a quantidade de informações que influenciam na capacidade do revisor e a dificuldade de agregá-las de forma manual, levando a escolhas tardias ou fundamentalmente ruins.

Trabalhos mais recentes apresentam propostas e métodos de avaliação baseados em informações de colaboração, como atividade, eficiência de revisões passadas e capacidade de interação. Estes resultados motivaram a proposta de novos métodos de recomendação baseados em redes colaborativas e voltados para o desenvolvimento global de software, com suas peculiaridades.

Para expansão dos métodos já avaliados, especialistas podem ser recomendados para projetos distintos de suas contribuições originais. Para isso, este trabalho propõe a utilização de ontologias para alinhamento dos tópicos de semânticos internos dos repositórios, de forma a inferir que se um desenvolvedor domina determinado conhecimento de um projeto, poderá auxiliar em outro projeto com demandas equivalentes. Para testar esta hipótese, foi realizado um estudo de caso com o alinhamento de projetos relevantes da comunidade opensource. Foram identificadas diversas equivalências, algumas relevantes e outras que podem ser negadas pelo o operador do processo, um especialista realiza a curadoria do alinhamento e julga as sugestões geradas automaticamente.

Como trabalhos futuros, existem diversas aplicações de ontologia nos métodos propostos. É possível sua utilização para integração e alinhamento dos dados de diferentes bases que contém informações sobre desenvolvedores e sua reputação, como o StackOverflow (já explicitado neste trabalho) e ferramentas como o Twitter. Além disso, testes aprofundados e avaliação dos métodos de recomendação conhecidos podem apontar novas aplicações de ontologia com o objetivo de melhorar a qualidade das saídas e a predição de revisores cada vez mais adequados.

## REFERÊNCIAS


[1] Gediminas Adomavicius and Alexander Tuzhilin. 2005. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering* 6 (2005), 734–749.

[2] Jorge Luis Nicolas Audy and Rafael Prikladnicki. 2007. *Desenvolvimento distribuído de software.* Elsevier.

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13).* IEEE Press, 712–721.

[4] Vipin Balachandran. 2013. Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation. In *Software Engineering (ICSE), 2013 35th International Conference on.* IEEE, 931–940.

[5] Earl Barr, Christian Bird, Peter Rigby, Abram Hindle, Daniel German, and Premkumar Devanbu. 2012. Cohesive and isolated development with branches. *Fundamental Approaches to Software Engineering* (2012), 316–331.

[6] G. Bavota and B. Russo. 2015. Four eyes are better than two: On the impact of code reviews on software quality. *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings* (2015), 81–90. https://doi.org/10.1109/ICSM.2015.7332454

[7] O. Baysal, O. Kononenko, R. Holmes, and M.W. Godfrey. 2013. The influence of non-technical factors on code review. *Proceedings - Working Conference on Reverse Engineering, WCRE* (2013), 122–131. https://doi.org/10.1109/WCRE.2013.6671287

[8] O. Baysal, O. Kononenko, R. Holmes, and M. W. Godfrey. 2012. The Secret Life of Patches: A Firefox Case Study. In *2012 19th Working Conference on Reverse Engineering.* 447–455. https://doi.org/10.1109/WCRE.2012.54

[9] M.a Beller, A.a Bacchelli, A.a Zaidman, and E.b Juergens. 2014. Modern code reviews in open-source projects: Which problems do they fix? *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings* (2014), 202–211. https://doi.org/10.1145/2597073.2597082

[10] Tim Berners-Lee, James Hendler, and Ora Lassila. 2001. The semantic web. *Scientific american* 284, 5 (2001), 34–43.

[11] C. Bird, T. Carnahan, and M. Greiler. 2015. Lessons learned from building and deploying a code review analytics platform. *IEEE International Working Conference on Mining Software Repositories* 2015 (2015), 191–201. https://doi.org/10.1109/MSR.2015.25

[12] Barry Boehm and Victor R. Basili. 2001. Software Defect Reduction Top 10 List. *Computer* 34, 1 (12 2001), 135–137. https://doi.org/10.1109/2.962984

[13] Carl Boettiger. 2015. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49, 1 (2015), 71–79.

[14] A. Bosu and J.C. Carver. 2014. Impact of developer reputation on code review outcomes in OSS projects: An empirical investigation. *International Symposium on Empirical Software Engineering and Measurement* (2014). https://doi.org/10.1145/2652524.2652544

[15] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. 2002. A framework for ontology integration. In *The Emerging Semantic Web—Selected Papers from the First Semantic Web Working Symposium.* 201–214.

[16] Valentine Casey. 2010. Virtual software team project management. *Journal of the Brazilian Computer Society* 16, 2 (2010), 83–96.

[17] M. E. Fagan. 1976. Design and Code Inspections to Reduce Errors in Program Development. *IBM Syst. J.* 15, 3 (09 1976), 182–211. https://doi.org/10.1147/sj.153.0182

[18] Roy T Fielding and Richard N Taylor. 2002. Principled design of the modern Web architecture. *ACM Transactions on Internet Technology (TOIT)* 2, 2 (2002), 115–150.

[19] Juliana Freire, Philippe Bonnet, and Dennis Shasha. 2012. Computational reproducibility: state-of-the-art, challenges, and database research opportunities. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data.* ACM, 593–596.

[20] Chenbo Fu, Mingming Zhou, Qi Xuan, and Hong-Xiang Hu. 2017. Expert recommendation in oss projects based on knowledge embedding. *International Workshop on Complex Systems and Networks (IWCSN)*, 149–155.

[21] Hugo Fuks, Alberto Barbosa Raposo, Marco Aurélio Gerosa, and Carlos J Pereira Lucena. 2003. Do modelo de colaboração 3c à engenharia de groupware. *Simpósio Brasileiro de Sistemas Multimídia e Web–Webmidia* (2003), 0–8.

[22] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering.* ACM, 345–355.

[23] Georgios Gousios, Margaret-Anne Storey, and Alberto Bacchelli. 2016. Work practices and challenges in pull-based development: The contributor's perspective. In *Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on.* IEEE, 285–296.

[24] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: the integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1.* IEEE Press, 358–368.

[25] Nicola Guarino. 1998. *Formal ontology in information systems: Proceedings of the first international conference (FOIS'98), June 6-8, Trento, Italy.* Vol. 46. IOS press.

[26] Chen He, Denis Parra, and Katrien Verbert. 2016. Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. *Expert Systems with Applications* 56 (2016), 9–27.

[27] Martin Hepp. 2008. Ontologies: State of the art, business potential, and grand challenges. In *Ontology Management.* Springer, 3–22.

[28] Darrel C Ince, Leslie Hatton, and John Graham-Cumming. 2012. The case for open computer programs. *Nature* 482, 7386 (2012), 485–488.

[29] Jing Jiang, Yun Yang, Jiahuan He, Xavier Blanc, and Li Zhang. 2017. Who should comment on this pull request? Analyzing attributes for more accurate commenter recommendation in pull-based development. *Information and Software Technology* 84 (2017), 48–62.

[30] C. F. Kemerer and M. C. Paulk. 2009. The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data. *IEEE Transactions on Software Engineering* 35, 4 (07 2009), 534–550. https://doi.org/10.1109/TSE.2009.27

[31] O.a Kononenko, O.b Baysal, L.c Guerrouj, Y.b Cao, and M.W.a Godfrey. 2015. Investigating code review quality: Do people and participation matter? *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings* (2015), 111–120. https://doi.org/10.1109/ICSM.2015.7332457

[32] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. The impact of code review coverage and code review participation on Software quality: A case study of the Qt, VTK, and ITK projects. *11th Working Conference on Mining Software Repositories, MSR 2014 - Proceedings* (2014), 192–201. https://doi.org/10.1145/2597073.2597076

[33] A. Meneely, A.C.R. Tejeda, B. Spates, S. Trudeau, D. Neuberger, K. Whitlock, C. Ketant, and K. Davis. 2014. An empirical investigation of socio-technical code review metrics and security vulnerabilities. *6th International Workshop on Social Software Engineering, SSE 2014 - Proceedings* (2014), 37–44. https://doi.org/10.1145/2661685.2661687

[34] Stuart E Middleton, David C De Roure, and Nigel R Shadbolt. 2001. Capturing knowledge of user preferences: ontologies in recommender systems. In *Proceedings of the 1st international conference on Knowledge capture.* ACM, 100–107.

[35] Stuart E Middleton, Nigel R Shadbolt, and David C De Roure. 2004. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)* 22, 1 (2004), 54–88.

[36] Rodrigo Morales, Shane McIntosh, and Foutse Khomh. 2015. 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)* 00 (2015), 171–180.

[37] Ana Maria Nicolaci-da Costa and Mariano Pimentel. 2011. Sistemas colaborativos para uma nova sociedade e um novo ser humano. *Sistemas colaborativos. PIMENTEL, M.; FUKS, H.(Orgs.). Rio de Janeiro: Elsevier* (2011).

[38] A. Ouni, R. G. Kula, and K. Inoue. 2016. Search-Based Peer Reviewers Recommendation in Modern Code Review. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 367–377. https://doi.org/10.1109/ICSME.2016.65

[39] M Tamer Özsu and Patrick Valduriez. 2011. *Principles of distributed database systems*. Springer Science & Business Media.

[40] Helena Sofia Pinto and João P Martins. 2001. A methodology for ontology integration. In *Proceedings of the 1st international conference on Knowledge capture*. ACM, 131–138.

[41] Mohammad Masudur Rahman, Chanchal K Roy, and Jason A Collins. 2016. CoRReCT: code reviewer recommendation in GitHub based on cross-project and technology experience. In *Software Engineering Companion (ICSE-C), IEEE/ACM International Conference on*. IEEE, 222–231.

[42] Vinicius Junqueira Schettino and Marco Antônio Pereira Araújo. 2017. Implantação da prática de code review em um modelo de desenvolvimento de software: um estudo de caso. (2017).

[43] Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. 2014. Understanding watchers on GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 336–339.

[44] Pavel Shvaiko and Jérôme Euzenat. 2013. Ontology matching: state of the art and future challenges. *IEEE Transactions on knowledge and data engineering* 25, 1 (2013), 158–176.

[45] Giuseppe Silvestri, Jie Yang, Alessandro Bozzon, and Andrea Tagarelli. 2015. Linking Accounts across Social Networks: the Case of StackOverflow, Github and Twitter.. In *KDWeb*. 41–52.

[46] Rajesh Sinha and Prem Sewak Sudhish. 2016. A principled approach to reproducible research: a comparative review towards scientific integrity in computational research. In *Ethics in Engineering, Science and Technology (ETHICS), 2016 IEEE International Symposium on*. IEEE, 1–9.

[47] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *Social computing (SocialCom), 2013 international conference on*. IEEE, 188–195.

[48] X.a Xia, D.b Lo, X.a b Wang, and X.a Yang. 2015. 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). *2015 IEEE 31st International Conference on Software Maintenance and Evolution, ICSME 2015 - Proceedings* (2015), 261–270. https://doi.org/10.1109/ICSM.2015.7332472

[49] Z. Xia, H. Sun, J. Jiang, X. Wang, and X. Liu. 2017. A hybrid approach to code reviewer recommendation with collaborative filtering. In *2017 6th International Workshop on Software Mining (SoftwareMining)*. 24–31. https://doi.org/10.1109/SOFTWAREMINING.2017.8100850

[50] Xin Yang, Norihiro Yoshida, Raula Gaikovina Kula, and Hajimu Iida. 2016. Peer review social network (PeRSoN) in open source projects. *IEICE TRANSACTIONS on Information and Systems* 99, 3 (2016), 661–670.

[51] Y. Yu, H. Wang, G. Yin, and C. X. Ling. 2014. Reviewer Recommender of Pull-Requests in GitHub. In *2014 IEEE International Conference on Software Maintenance and Evolution*. 609–612. https://doi.org/10.1109/ICSME.2014.107

[52] Yue Yu, Huaimin Wang, Gang Yin, and Charles X Ling. 2014. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, Vol. 1. IEEE, 335–342.

[53] Motahareh Bahrami Zanjani, Huzefa Kagdi, and Christian Bird. 2016. Automatically recommending peer reviewers in modern code review. *IEEE Transactions on Software Engineering* 42, 6 (2016), 530–543.