Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Pós-graduação em Engenharia de Software

# Utilizando Ontologias para apoiar a recomendação de revisores de código

**Vinicius Schettino**

JUIZ DE FORA
SETEMBRO, 2018

# Utilizando Ontologias para apoiar a recomendação de revisores de código

Vinicius Schettino

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Pós Graduação em Ciência da Computação

Mestrado em Engenharia de Software

Orientador: Regina Braga

JUIZ DE FORA

SETEMBRO, 2018

# Resumo

As instruções aqui contidas objetivam auxiliar os autores na preparação de documentos para impressão de monografias do Departamento de Ciência da Computação. Os estilos encontram-se definidos em um modelo denominado Monografia.cls. O resumo deve ser escrito na mesma língua do texto (Português, Inglês ou Espanhol) e descreve o conteúdo do texto em cerca de 150-200 palavras. Esta é a primeira versão das instruções e dos formatos e, portanto, sujeita a incorreções e omissões. Sugestões de melhorias são muito benvindas: envie mensagem para jairo.souza@ufjf.edu.br.

**Palavras-chave:** Monografia, latex, instruções.

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# 1 Introdução

O *code review* é considerado uma das principais técnicas para diminuição de defeitos de software (BOEHM; BASILI, 2001). 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* (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 no desenvolvimento distribuído. Dentro de workflows de trabalho 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 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 (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 os desafios à colaboração co-localizada são potencializados e as soluções tradicionais não são suficientes para fomentar esta aspecto das atividades distribuídas (COSTA; PIMENTEL, 2011). Casey (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 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 (YU et al., 2014b; XIA et al., 2015; JIANG et al., 2017). Esses trabalhos foram estudados e levados em consideração para escrita do presente texto. Também foram revisadas pesquisas que apontam caracterísitcas de revisões, revisores e autores que possivelmente potencializam a colaboração (KEMERER; PAULK, 2009; BIRD; CARNAHAN; GREILER, 2015; BAYSAL et al., 2013).

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 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 (HE; PARRA; VERBERT, 2016).

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 relaçs 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. (FU et al., 2017) descreve uma recomendação baseada num grafo de relacionamento entre desenvolvedores e seu status de trabalho no projeto. Xia et al. (XIA et al., 2017) agrupa desenvolvedores utilizando métodos de proximidade para capturar relações implícitas e explícitas. Yu et al. (YU et al., 2014b; YU et al., 2014a) 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. (YANG et al., 2016) 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 angariar estas informações relevantes é através do uso de ontologias (MIDDLETON; ROURE; SHADBOLT, 2001; MIDDLETON; SHADBOLT; ROURE, 2004). 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 (BACCHELLI; BIRD, 2013). 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*" (GOUSIOS; PINZGER; DEURSEN, 2014).

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.

Assim, o objetivo deste trabalho é discutir como ontologias podem enriquecer uma rede colaborativa de desenvolvedores para apoiar na recomendação de revisores de código, propondo uma ontologia capaz de auxiliar no processo. A principal contribuição é a definição das informações relevantes e propor o modelo de extração destas das bases de dados disponíveis.

– ORGANIZAÇÃO DO TRABALHO –

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

# 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 (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).

### 2.1.2 Histórico

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).

## 2.2 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, submtendo 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).

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*.

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 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

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

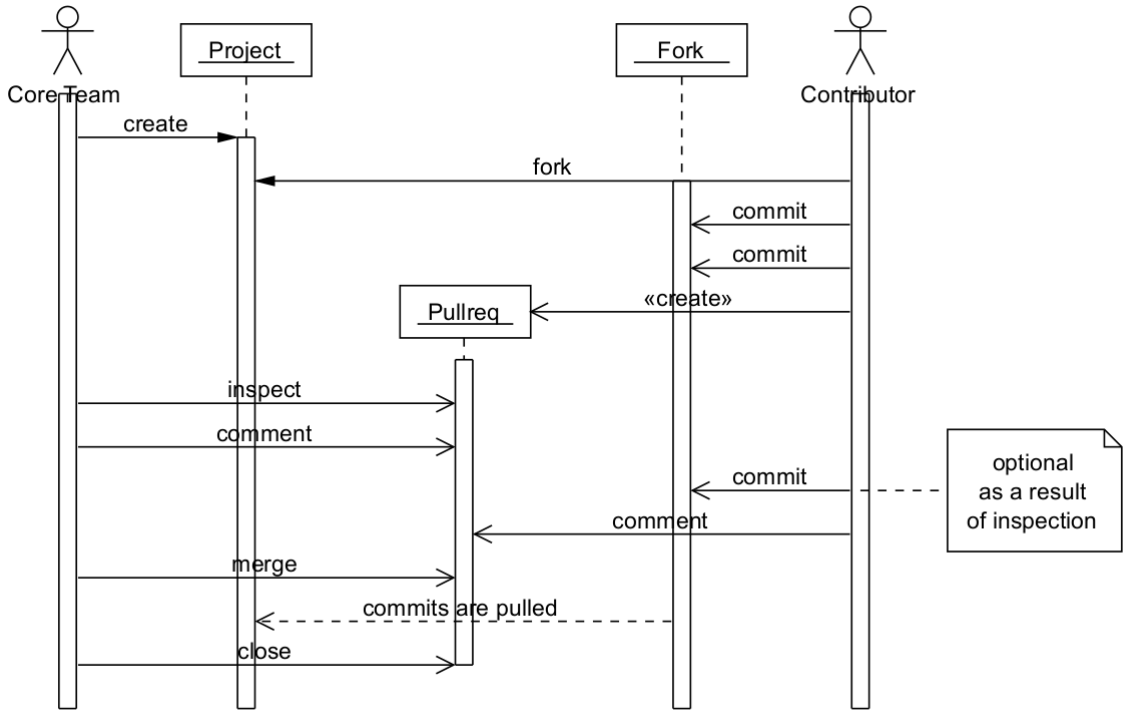


Figura 2.1: Pull Request Process (GOUSIOS; PINZGER; DEURSEN, 2014)

(*closed*) ou aceita *merged*.

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.

## 2.3 Ontologia

Ontologia é a definição de um vocabulário ou linguagem comum para representar conhecimentos (GRUBER, 1995). 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 tornou comum para representação de conhecimento na Web Semântica. 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 novas propriedades e relacionamentos baseadas em conhecimento explícito (BERNERS-LEE; HENDLER; LASSILA, 2001)

# 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 (BALACHANDRAN, 2013) 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 (RAHMAN; ROY; COLLINS, 2016) 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 que o histórico completo do desenvolvedor, entre todas suas contribuições OpenSource que foram feitas no GitHub.

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 (ZANJANI; KAGDI; BIRD, 2016) é outra abordagem que utiliza o histórico dos potenciais revisores. Contudo, aqui ao invés de encontrar através da similaridade do código, é analisado o quão participativo e útil ele foi em revisões 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 (OUNI; KULA; INOUE, 2016) 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.

Além de relevantes para o entendimento geral da área, os trabalhos selecionados também foram avaliados de acordo com a relevância do veículo de publicação e o fator de impacto QUALIS, como mostra a tabela 3.1. Os fatores constantes são oriundos da plataforma oficial da CAPES[5]

[5]https://sucupira.capes.gov.br/sucupira/public/consultas/coleta/veiculoPublicacaoQualis/listaConsultaGeralPeri

| Título | Ano | Fonte | Qualis |
|---|---|---|---|
| Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation | 2013 | ICSE | A1 |
| CoRReCT: code reviewer recommendation in GitHub based on cross-project and technology experience | 2016 | ICSE | A1 |
| Automatically recommending peer reviewers in modern code review | 2016 | IEEE Transactions on Software Engineering | A1 |
| Search-Based Peer Reviewers Recommendation in Modern Code Review. | 2017 | ICSME | A1 |

Tabela 3.1: Chosen Papers Introduction

# 4 Metodologia

Os dados da análise são extraídos do GitHub através da API[6] 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(FIELDING; TAYLOR, 2002). Existe a API em recém lançada v4[7], 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 4.1 mostra o pipeline de extração e estruturação dos dados.

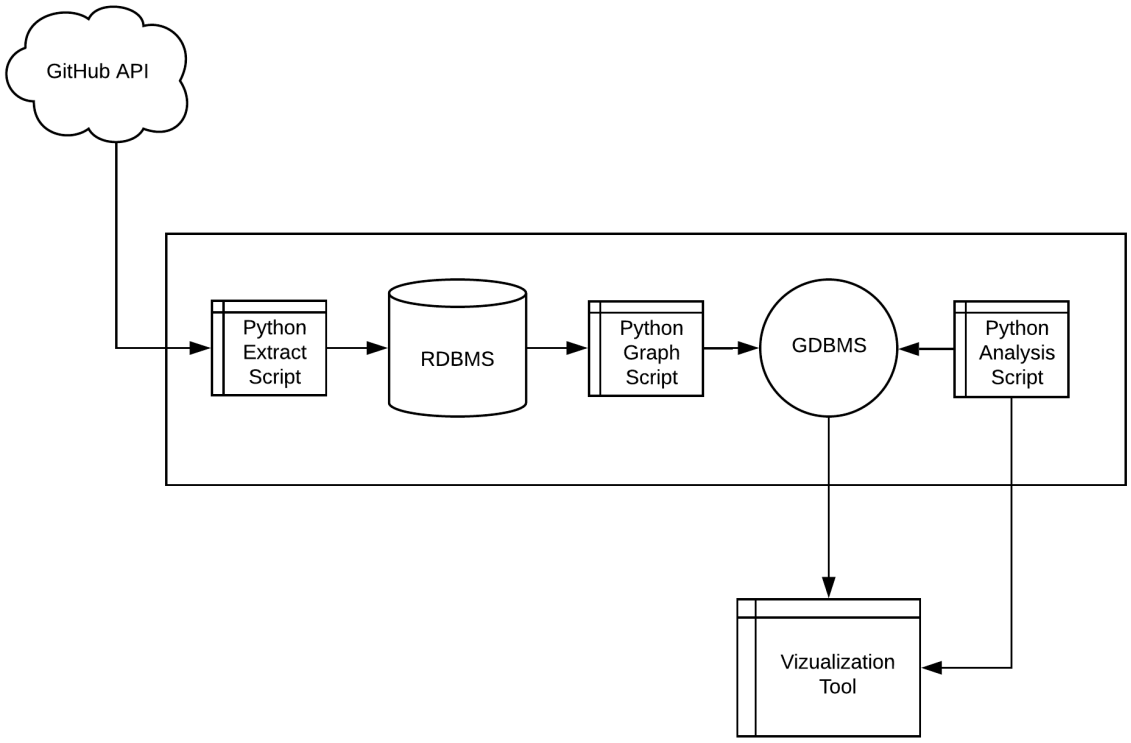


Figura 4.1: Processo de extração dos dados

Os dados extraídos são estruturados em um esquema relacional, mais especificamente em uma instância PostgreSQL 10.5. Este passo intermediário ocorre para que os

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

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 4.3 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 dos 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 4.2.

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

Figura 4.2: Tabela de Reactions Disponíveis

Figura 4.3: Modelo Entidade Relacionamento

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[8]. É a partir dele

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

que as análises são executadas e vizualizadas utilizando o Gephi[9].

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.

[9]https://gephi.org/

## 4.1 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 (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 figur 4.4 Estes são orquestrados atravé do docker-compose, que configura os parâmetros e acessos necessários para o funcionamento do sistema sem que o usuário precise realizar nenhuma ação extra.

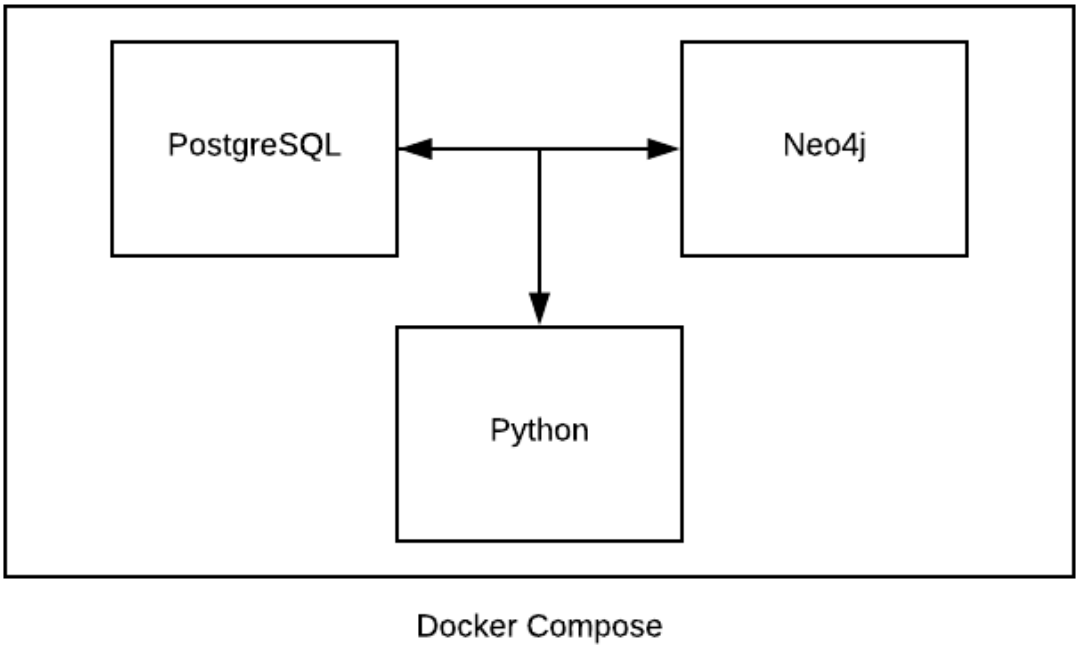


Figura 4.4: Modelo de contâiners

Com um conjunto grande de dependências, tecnologias e minúncias que estão envolvidas neste tipo de experimento, o código fonte e a descrição ainda que detalhada dos dados não é suficiente para alcançar níveis adequados de reprodutibilidade (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 4.5 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 4.6 mostra o detalhamento de cada aspecto que contribui para que esta característica seja evidenciada:

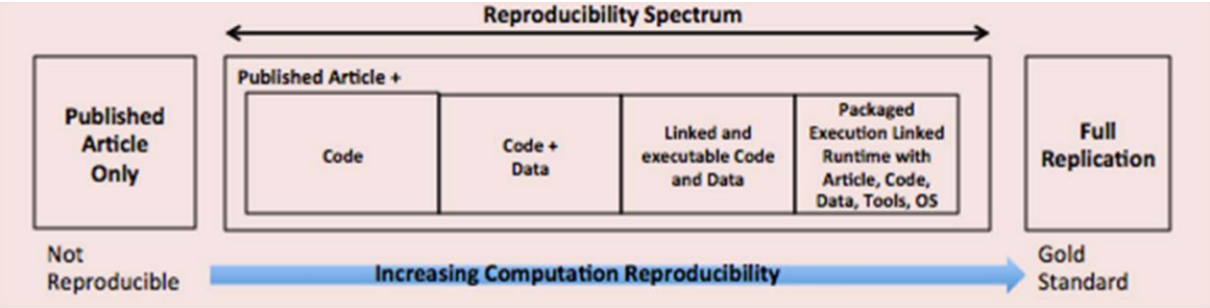


Figura 4.5: Espectro de Reprodutibilidade

| | 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 4.6: Escala de Maturidade de Reprodutibilidade

**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.

**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.

## 4.2 Modelo Proposto

O modelo proposto se baseia nos dados de relacionamento entre autores e revisores, mais espeficificamente 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 4.7 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

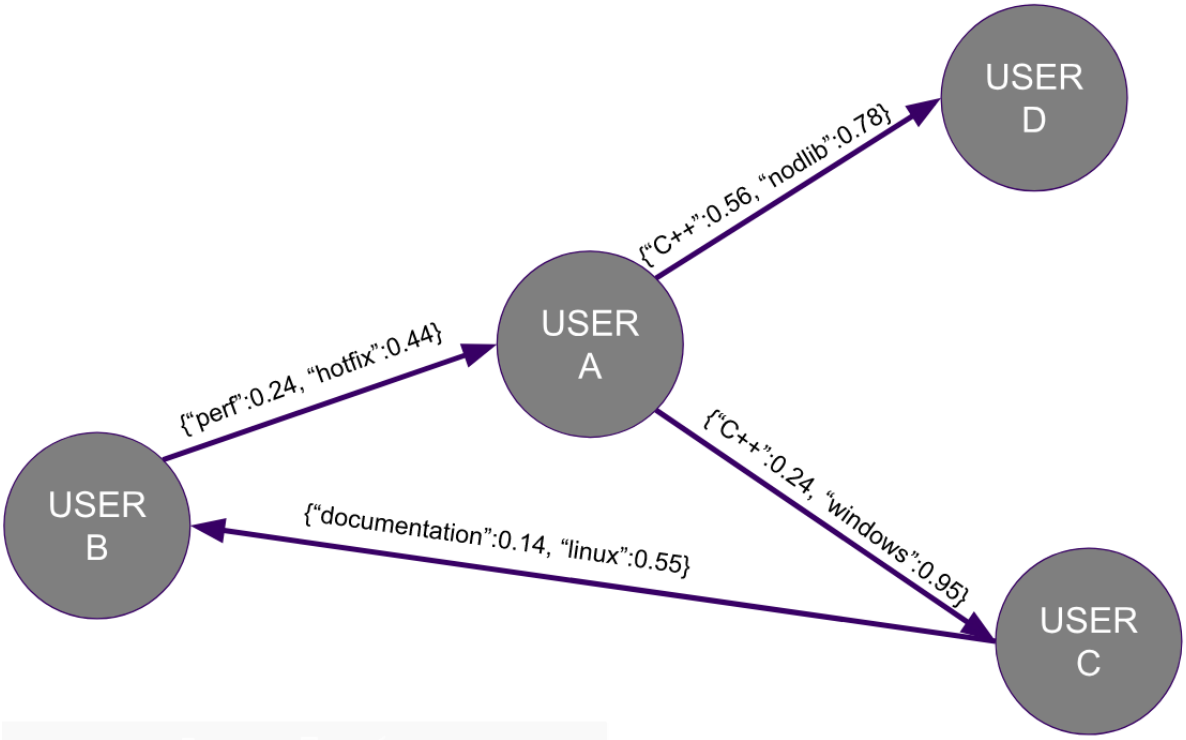


Figura 4.7: Modelo da Rede Colaborativa

conhecimento é definido através das labels de cada pullrequest. O cálculo é feito através da fórmula 4.1.

$$W(A, B, label) = \frac{countRevs(A, B, tag)}{countTotRevs(B, tag)}. \tag{4.1}$$

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.

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 4.2.

$$avgW(A, B) = \frac{countRevs(A, B)}{countTotRevs(B)}. \tag{4.2}$$

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

## 4.3 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[10]

A tabela 4.1 ilustra os projetos escolhidos e informações de contexto e tamanho.

| **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 4.1: Descrição dos Respositórios Selecionados

## 4.4 Integração de repositórios de dados

Apesar das ricas 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.

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

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 (ÖZSU; VALDURIEZ, 2011), 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.

**Dados semanticamente conflitantes:** Muitas vezes entidades análogas são representadas com nomenclaturas diferentes, e o inverso também pode acontecer. O alinhamento destes domínios pode ser feito manualmente, através da sintática dos campos ou 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 (ÖZSU; VALDURIEZ, 2011): física ou lógica. 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. (VASILESCU; FILKOV; SEREBRENIK, 2013) 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. (SILVESTRI et al., 2015) 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.

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[11] que implementa a interface SQL BigQuery[12]. As informações de cada usuários são então aprimoradas, como mostra a figura 4.8.

Para o processo de matching, além das informações de email utilizadas anteri-

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

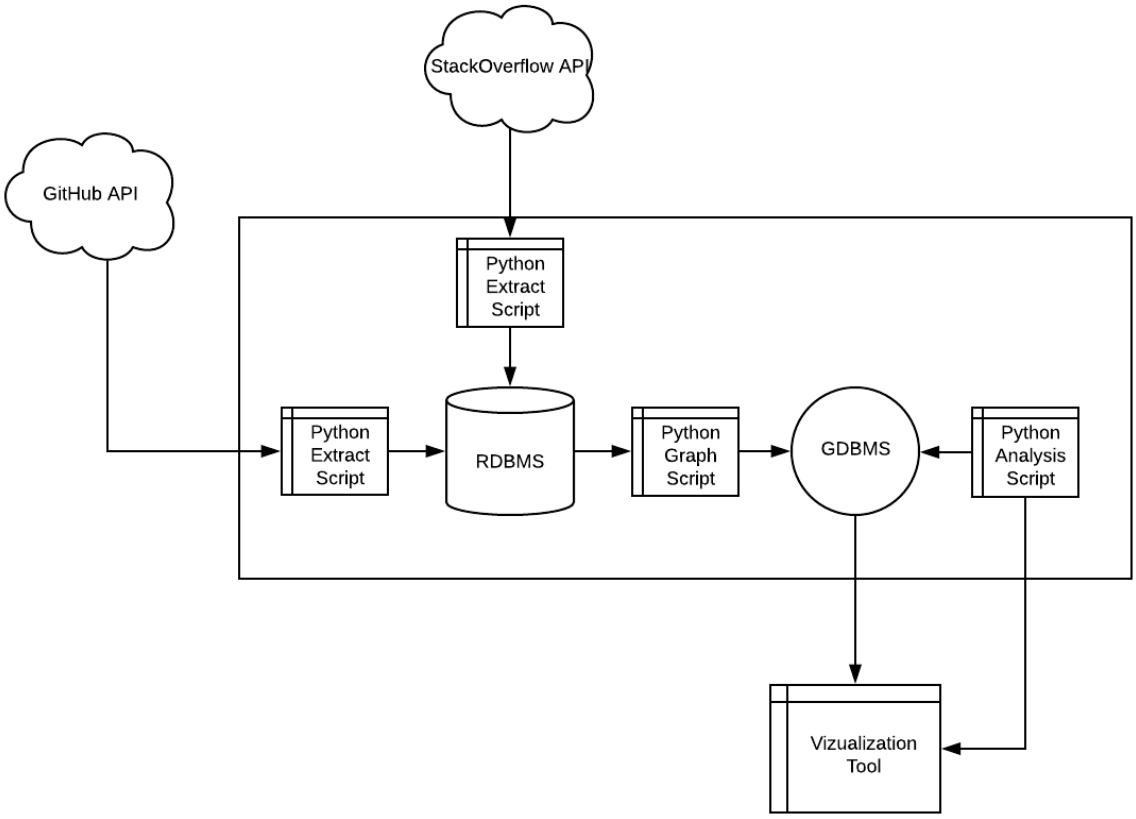


Figura 4.8: Processo de Extração com StackOverflow

ormente, 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 pullrequests.

# Bibliografia

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.

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.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The semantic web. *Scientific american*, JSTOR, v. 284, n. 5, p. 34–43, 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.

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

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.

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.

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.

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.

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.

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.

GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing? *International journal of human-computer studies*, Elsevier, v. 43, n. 5-6, p. 907–928, 1995.

HE, C.; PARRA, D.; VERBERT, K. Interactive recommender systems: A survey of the state of the art and future research challenges and opportunities. *Expert Systems with Applications*, Elsevier, v. 56, p. 9–27, 2016.

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.

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.

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

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.

MIDDLETON, S. E.; ROURE, D. C. D.; SHADBOLT, N. R. Capturing knowledge of user preferences: ontologies in recommender systems. In: ACM. *Proceedings of the 1st international conference on Knowledge capture*. [S.l.], 2001. p. 100–107.

MIDDLETON, S. E.; SHADBOLT, N. R.; ROURE, D. C. D. Ontological user profiling in recommender systems. *ACM Transactions on Information Systems (TOIS)*, ACM, v. 22, n. 1, p. 54–88, 2004.

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.

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.

ÖZSU, M. T.; VALDURIEZ, P. *Principles of distributed database systems*. [S.l.]: Springer Science & Business Media, 2011.

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.

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.

SILVESTRI, G. et al. Linking accounts across social networks: the case of stackoverflow, github and twitter. In: *KDWeb*. [S.l.: s.n.], 2015. p. 41–52.

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.

VASILESCU, B.; FILKOV, V.; SEREBRENIK, A. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In: IEEE. *Social computing (SocialCom), 2013 international conference on*. [S.l.], 2013. p. 188–195.

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, 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.

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.