Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Computação Aplicada

IDENTIFICAÇÃO DE ESPECIALISTAS EM APIS A PARTIR DE CONHECIMENTO EXISTENTE EM REPOSITÓRIOS SOCIAIS DE SOFTWARE

Camille Nogueira de Macedo

Feira de Santana

2018

Universidade Estadual de Feira de Santana

Programa de Pós-Graduação em Computação Aplicada

Camille Nogueira de Macedo

IDENTIFICAÇÃO DE ESPECIALISTAS EM APIS A PARTIR DE CONHECIMENTO EXISTENTE EM REPOSITÓRIOS SOCIAIS DE SOFTWARE

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

Orientador: Roberto Almeida Bittencourt

Coorientador: José Amancio Macedo Santos

Feira de Santana

2018

Ficha Catalográfica – Biblioteca Central Julieta Carteado

Macedo, Camille Nogueira de
M12i ldentificação de especialistas em APIs a partir de conhecimento existente em repositórios sociais de software./ Camille Nogueira de Macedo. – 2018.
73f.: il.

Orientador: Roberto Almeida Bittencourt
Co-orientador: José Amancio Macedo Santos
Dissertação (mestrado) – Universidade Estadual de Feira de Santana, Programa de Pós-Graduação em Computação Aplicada, 2018.

1.API de software – Identificação. 2.Especialista. 3.Métricas. I.Bittencourt, Roberto Almeida, orient. II. Santos, José Amancio Macedo, co-orient. III.Universidade Estadual de Feira de Santana. IV. Titulo.

CDU: 004.42

Maria de Fátima de Jesus Moreira – Bibliotecária – CRB5/1120

Camille Nogueira de Macedo

Identificação de Especialistas em APIs a Partir de Conhecimento Existente em Repositórios Sociais de Software

Dissertação apresentada à Universidade Estadual de Feira de Santana como parte dos requisitos para a obtenção do título de Mestre em Computação Aplicada.

Feira de Santana, 04 de setembro de 2018

BANCA EXAMINADORA

Dr. Roberto Almeida Bittencourt (Orientador)
Universidade Estadual de Feira de Santana

Dra. Christina von Flach Garcia Chavez
Universidade Federal da Bahia

Dr. Rodrigo Tripodi Calumby
Universidade Estadual de Feira de Santana

Abstract

Identification of software development experts usually represents high operational costs for companies. In order to mitigate this problem, some researchers have presented different strategies to find experts. Despite their efforts, such strategies pointo to particular solutions and particular evaluations, leading to different conclusions despite their use of similar inputs. In this work, we built an understanding of the field by selecting some of the recent metrics, and proposed a tool that can use these metrics to identify experts in software APIs from their use of the source code in a set of projects available on a social software repository. From these results, we performed an exploratory study with three software APIs with the goal of evaluating five metrics to identify experts from source code. In this evaluation, we produced expert rankings from the metrics computed by the tool, and we perceived that these metrics show a strong correlation with each other. We also evaluated the metrics against a ground truth based on software development performed after computing the metrics. Results show that, for a small scenario of developers who use the API and with less API complexity, the metrics show better precision. For larger developer groups and larger API complexity, results are less accurate, but still have an average accuracy of 48% from Top-5 rankings and for the three APIs used in the evaluation. This work adds to the body of knowledge on automatic determination of software expertise, pointing to the feasibility of the techniques, and presenting an evaluation of the potential use of expertise metrics in the context of software APIs used in social software repositories.

Keywords: identification, expert, software API, metrics.

Resumo

A identificação de especialistas em desenvolvimento de software geralmente representa um alto custo operacional para as empresas. Para mitigar este problema, alguns pesquisadores apresentaram diferentes estratégias para encontrar especialistas. Apesar destes esforços, tais estratégias apontam para soluções particulares e avaliações particulares, gerando conclusões diferentes apesar de usarem insumos similares. Neste trabalho, construímos um entendimento sobre a área selecionando algumas das métricas recentes e propomos um protótipo de ferramenta capaz de utilizar estas métricas para identificar especialistas em uma API de software a partir de seu uso no código fonte em um conjunto de projetos disponíveis em um repositório social de software. A partir desses resultados, realizamos um estudo exploratório com três APIs de software com o objetivo de avaliar cinco métricas para identificação de especialistas a partir do código fonte. Nesta avaliação, produzimos rankings de especialistas a partir das métricas computadas pelo protótipo e percebemos que estas métricas apresentam uma forte correlação entre si. Avaliamos também as métricas em relação a uma *ground truth* baseada no desenvolvimento de software posterior ao cômputo das métricas. Os resultados apontam que, para um cenário pequeno de desenvolvedores que utilizam a API e com menor complexidade da API, as métricas apresentam uma melhor precisão. Para grupos grandes de desenvolvedores e com maior complexidade da API, os resultados são menos precisos, porém, ainda assim, apresentam uma precisão média de 48%, a partir de rankings Top-5 e considerando as três APIs utilizadas na avaliação. Este trabalho vem somar ao corpo de conhecimento sobre determinação automática de expertise de software, apontando a viabilidade e apresentando uma avaliação do potencial uso de métricas de expertise no contexto de APIs usadas em repositórios sociais de software.

Palavras-chave: identificação, especialista, API de software, métricas.

Prefácio

Esta dissertação de mestrado foi submetida a Universidade Estadual de Feira de Santana (UEFS) como requisito parcial para obtenção do grau de Mestre em Computação Aplicada.

A dissertação foi desenvolvido dentro do Programa de Pós-Graduação em Computação Aplicada (PGCA) tendo como orientador o Dr. **Roberto Almeida Bittencourt** e co-orientador o Dr. **José Amancio Macedo Santos**.

Agradecimentos

Em primeiro lugar agradecer a Deus pelo seu cuidado com a minha vida, por ter me dado forças a continuar e sabedoria nas conduções de cada uma das decisões e passos dados. Sem Ele nada seria possível.

Aos meus mestres e parceiros, Roberto e Amâncio, não tenho palavras para agradecer a parceria e dedicação de vocês. Embarcaram comigo na minha loucura de fazer esse mestrado a "distância", sem medir esforços. Foram muitas reuniões online, escritas de artigo, dissertação e discussões ao longo desse tempo. Meu muito obrigada! Vocês são feras e sou super fã de vocês.

A Douglas, grande parceiro de mestrado, que me ajudou do inicio ao fim a pensar e desenvolver essa pesquisa, meu muito obrigada!

A minha família, meus pais pelo amor incondicional, sempre me apoiando e me dando forças para não desistir. Meus irmãos e minha sobrinha que mesmo sem entenderem nada me apoiaram. Amo muito vocês!

A Jeferson, que nessa reta final se tornou uma pessoa fundamental e indispensável para a conclusão desse trabalho, me apoiando, estando ao meu lado me incentivando, batendo código, escrevendo e traduzindo texto junto comigo. Não tenho dúvidas que sem você nada disso seria igual. Obrigada por tudo. Amo você!

Aos meus amigos que sempre me apoiaram, me deram forças e entenderam a minha ausência momentânea. Por muitas vezes me chamaram de louca, fazer intercâmbio, trabalhar e estudar. A vocês que entenderam o meu sonho e não me fizeram desistir. Meu muito obrigada!

As empresas Transpoco e Total por me ajudarem a validar o protótipo da ferramenta, me fornecendo projetos de software e acesso aos desenvolvedores para realização das pesquisas necessárias e assim que eu pudesse concluir as validações.

A todos que direta ou indiretamente contribuíram para o sucesso desse projeto, deixo aqui os meus sinceros agradecimentos. Todos vocês fazem parte dessa história!

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A identificação de especialistas em desenvolvimento de software é atualmente uma tarefa difícil. Envolve a seleção de currículos, entrevista de candidatos, realização de testes, busca de provas das experiências anteriores do candidato, entre outras atividades. Existem várias maneiras de identificar desenvolvedores apropriados para um projeto, desde processos tradicionais de entrevista até sistemas de recomendação e sistemas especialistas [da Silva et al. 2012]. Tais atividades geralmente representam alto custo operacional para as empresas, porque a experiência dos desenvolvedores é principalmente relacionada às suas contribuições em projetos de software e sua comunicação com a equipe de software. Como resultado, em certos contextos, o resultado final da seleção nem sempre corresponde ao esperado.

Na indústria de software, com muitas vagas abertas e com muitos candidatos não tão qualificados, extrair, classificar e selecionar candidatos com as habilidades certas são tarefas importantes para o sucesso dos projetos de software [McCuller 2012]. Muitas das estratégias comumente usadas para selecionar candidatos possuem limitações. Uma das principais limitações é a seleção de candidatos com bases em suas competências e habilidades técnicas. Por exemplo, quando é desejável contratar um desenvolvedor que conheça uma dada biblioteca de software, em especial como ele domina o uso de sua interface de programação de aplicação (API), não há uma maneira fácil de encontrá-lo.

Muitas dessas habilidades buscadas por profissionais que recrutam desenvolvedores de software não são facilmente extraídas por técnicas ou sistemas de recomendação já existentes, já que buscam por conhecimentos específicos como, por exemplo, o domínio de uma determinada API. Contratar um desenvolvedor errado pode levar a esforços e recursos adicionais para treinar o novo funcionário ou gastar mais tempo e recursos contratando outro. Além disso, decisões erradas de contratação são um fator de risco bem conhecido para o sucesso de um projeto de software [Sommerville et al. 2008].

Nesse contexto, alguns autores propõem diferentes definições para o termo experiência/expertise ou especialista/expert. Mockus e Herbsleb (2002) definem a expertise

como a capacidade do desenvolvedor de ser um especialista em código fonte e, se interpretada quantitativamente, reflete a capacidade de uma pessoa executar uma determinada tarefa. Alonso *et al.* (2008) definem especialistas como pessoas que registraram uma grande quantidade de transações em um sistema de controle de versão durante um dado período de tempo. Ehrlich e Shami (2008) definem especialistas como pessoas que têm conhecimento de informações confiáveis e rápidas e boas recomendações.

Esses artigos propõem estratégias de identificação de especialistas com base em repositórios de software. Eles extraem informações dos repositórios para identificar atributos que serão usados como entrada para as métricas que eles definem. Ma *et al.* (2009), por exemplo, recomendam especialistas baseados na observação do uso de métodos em um projeto. Eles chamam isso de *experiência de uso*. Fritz *et al.* (2010) definem o grau de conhecimento de um desenvolvedor a partir de sua interação com os métodos do projeto e do número de novas estruturas adicionadas e chamam isso *experiência de software*.

Um aspecto importante a ser destacado é que, com a evolução dos repositórios de software, outras informações disponíveis podem oferecer suporte à identificação de especialistas. Os repositórios sociais de software permitem extrair informações não apenas dos projetos armazenados, mas também informações relevantes de desenvolvedores [Dabbish et al. 2012]. O GitHub[1], por exemplo, é uma ferramenta web de controle de versionamento que incorpora funcionalidades sociais, tornando-o assim um repositório social de software. No GitHub é possível encontrar vários projetos, seguir e acompanhar projetos, seguir e acompanhar desenvolvedores, suas atividades e contribuições. Além disso, os desenvolvedores podem colaborar com projetos *open source* de forma ativa. É possível extrair informações de vários projetos e de vários desenvolvedores que contribuem para eles.

O trabalho realizado por Teyton et. al. (2013) aborda a identificação de especialistas em bibliotecas (APIs) de software em repositórios sociais. Este foi um dos trabalhos pioneiros no contexto de APIs de software. Eles propõem o LIBTIC, um mecanismo de pesquisa de especialistas em APIs, preenchido automaticamente pela mineração de repositórios de software. Para este processo de recomendação, eles utilizaram apenas duas métricas relativamente simples e validaram a ferramenta de recomendação através de um estudo de caso com o projeto Apache HBase. Este trabalho demonstra que, com o surgimento dos repositórios sociais de software, outras possibilidades de suporte à identificação de especialistas podem ou devem ser consideradas.

Uma observação importante sobre as estratégias de recomendação identificadas é que elas foram apresentadas de forma independente, ou seja, as estratégias usadas não apresentam relação entre si. Os autores propõem métricas e apresentam resultados satisfatórios de avaliação com base em suas próprias heurísticas, além da especificidade de encontrar especialistas em um dado projeto de software. No entanto,

[1]https://github.com

observamos que, apesar das diferenças entre as métricas, as heurísticas adotam um conjunto similar de insumos. Assim, alguns autores usaram o mesmo conjunto de insumos e alcançaram resultados diferentes. A variedade de métricas de recomendação de especialistas, a relação entre elas e a adequação da métrica para obtenção de um resultado satisfatório é um aspecto que tem sido pouco explorado na área. Finalmente, quando se fala da contratação de especialistas para um projeto, é necessário ir além da identificação de especialistas que já desenvolvem para o projeto. O domínio de habilidades mais gerais como, por exemplo, a experiência de uso de uma API de software em um ou mais projetos é desejável no contexto de contratação de especialistas e, com exceção do trabalho de Teyton *et al.* (2013), o esforço de pesquisa não tem sido prolífico neste contexto mais geral.

1.1 Objetivos da pesquisa

O objetivo geral dessa pesquisa é fazer um estudo exploratório a fim de avaliar um conjunto de métricas para identificação de especialistas em APIs de software a partir do conhecimento existente em repositórios sociais de software e de desenvolvedores.

Os objetivos específicos são:

1. Avaliar um conjunto de métricas existentes para identificação de especialistas em APIs de software;

2. Escolher um conjunto de métricas para identificação de especialistas a partir das métricas avaliadas;

3. Desenvolver um sistema capaz de calcular as métricas e identificar especialistas a partir do conjunto de métricas definido;

4. Avaliar o conjunto de métricas proposto em projetos existentes a partir de critérios de sucesso da área de recuperação de informação.

1.2 Questões de pesquisa

As questões de pesquisa levantadas neste trabalho são:

- Que dados básicos podem ser utilizados para capturar expertise de APIs de software a partir de repositórios sociais de código fonte?
- Como combinar os dados básicos em métricas que mensurem expertise?
- Qual a viabilidade de computar essas métricas em sistemas de software *open source*?
- Das métricas estudadas, quais as adequadas para o contexto de identificação de especialistas em APIs de software?

- Como avaliar os resultados das métricas de identificação de especialistas em APIs de software em termos de eficiência e eficácia?

1.3 Contribuições do Trabalho

Diante do exposto, esse trabalho procurou apresentar algumas contribuições para a temática identificação de especialistas em APIs de software através de um estudo exploratório.

Uma das contribuições está relacionada à correlação entre as métricas existentes. Nós avaliamos um conjunto de métricas e as correlacionamos entre si. Com isso, percebemos que existe uma correlação alta entre elas. Toda esta avaliação, resultados e detalhamento dessa contribuição esta descrito com maiores detalhes no Capítulo 6.

Outra contribuição relevante é que, para um conjunto pequeno de desenvolvedores de software utilizando uma determinada API de menor complexidade, métricas de expertise apresentam uma precisão média de acerto melhor quando comparadas a situações com um grupo maior de desenvolvedores usando uma API de maior complexidade. O detalhamento desta avaliação e discussão também pode ser visto no Capítulo 6.

Foi desenvolvido ainda um protótipo de uma ferramenta capaz de processar as métricas para uma determinada API em um conjunto de n projetos. Os detalhes deste protótipo de ferramenta podem ser vistos no Capítulo 5.

1.4 Organização do Documento

No Capítulo 2, apresentamos os trabalhos relacionados aos quatro pilares desta pesquisa: expertise, recuperação da informação, mineração de repositórios sociais e interface de programação de aplicação (API). No Capítulo 3, é apresentado todo o detalhamento metodológico da pesquisa. No Capítulo 4, apresentamos o processo de classificação e seleção das métricas utilizadas para determinação de expertise. No Capítulo 5, apresentamos o protótipo da ferramenta desenvolvida para recomendação de especialistas em APIs de Software. No Capítulo 6 é apresentado o processo de avaliação das métricas no contexto de APIs de software. E, finalmente, no Capítulo 7, discutimos as conclusões obtidas neste trabalho.

Capítulo 2

Revisão Bibliográfica

Nesta seção iremos apresentar a revisão bibliográfica realizada para este trabalho com base em três pilares que julgamos importante para a identificação de especialistas em API: mineração de repositórios de software, expertise, e Interface de Programação de Aplicação – APIs. Por último, iremos explicar como foi realizado o processo de avaliação desses trabalhos pelos autores e seus resultados.

2.1 Mineração de Repositórios de Software

Em um meio organizacional onde os praticantes geralmente confiam em sua experiência e intuição para tomar decisões importantes, os gerentes alocam recursos de desenvolvimento e teste com base em sua experiência em projetos anteriores e sua intuição sobre a complexidade do novo projeto em relação a projetos anteriores. Os desenvolvedores geralmente usam sua experiência ao adicionar um novo recurso ou a corrigir um *bug*. Os testadores geralmente priorizam o teste de recursos que são conhecidos por serem propensos a erros com base em relatórios de campo e de *bugs*. Os repositórios de software contêm uma riqueza de informações valiosas sobre projetos de software. Usando as informações armazenadas nesses repositórios, os profissionais podem depender menos de sua intuição e experiência e depender mais de dados históricos e de campo [Hassan 2008].

Segundo Hassan (2008), a Mineração de Repositórios de Software analisa e cruza os ricos dados disponíveis em repositórios de software para revelar informações interessantes e acionáveis sobre sistemas e projetos de software.

É importante definir alguns termos relacionados a mineração de repositórios sociais de software para um melhor entendimento dessa pesquisa. São eles:

1. *Arquivo* é um contêiner de armazenamento da informação de software, que permite agrupar as informações de código em um mesmo local lógico;

2. *Commit* é um conjunto de mudanças permanentes realizadas por um desenvolvedor em um sistema de controle de versão, que é visto de forma atômica como uma mudança única;

3. *Contêiners* são arquivos ou trechos de arquivos de projetos (e.g., arquivos, classes, métodos) criados por desenvolvedores que agrupam um conjunto de linhas de código com informações relacionadas;

4. *Desenvolvedor* é o profissional que escreve código e/ou desenvolve software e realiza operações de commit sobre o um sistema de controle de versão;

5. *Simbolos* são unidades elementares de código (e.g., nomes de atributos, nomes de variáveis, nomes de métodos) criados por desenvolvedores com informações relevantes para o desenvolvimento de um software;

6. *Timestamp* é uma informação temporal que diz respeito instante de tempo em que algo foi realizado (e.g., timestamp de um commit).

Alguns autores têm proposto trabalhos relacionados à mineração de repositórios de software em diversas áreas. Zagalsky et. al. (2015) , por exemplo, realizaram um estudo quantitativo com foco em como o GitHub está sendo usado na educação e as motivações, benefícios e desafios que ele traz. Eles apontam que, recentemente, os educadores descobriram no Github um potencial colaborativo onde os desenvolvedores podem melhorar e gerenciar a experiência do aprendizado. Os autores explicam como o Github está emergindo como uma plataforma colaborativa para a educação, e sugerem que ele pode ser uma ferramenta poderosa para o gerenciamento do aprendizado.

Takhteyev e Hilts (2010) também fizeram um estudo empírico com o Github, mostrando que os desenvolvedores que o utilizam são altamente agrupados e concentrados principalmente na América do Norte e Europa Ocidental e do Norte, embora haja participantes espalhados em todo o mundo. Essa pesquisa apontou que esta distância tem um efeito importante na contribuição de código e no surgimento de novos usuários.

Hassan (2008) apresenta uma breve história sobre mineração de repositórios sociais e discute várias conquistas e resultados do uso de técnicas de mineração de repositórios para apoiar pesquisas e práticas de software. Ele apresenta também as oportunidades identificadas de pesquisa como, por exemplo, a extração de informações técnicas para auxiliar na tomada de decisão.

2.2 Expertise

Esta seção elenca definições de expertise e depois detalha trabalhos relacionados a expertise em um projeto de software.

A popularização dos repositórios de software leva-nos a considerar como pesquisas correlatas sobre estes repositórios podem nos oferecer *insights* sobre identificação de especialistas. Assim, esta seção visa apresentar trabalhos prévios que procuram minerar informação destes repositórios. A mineração de repositórios de software oferece uma grande quantidade de informações sobre projetos de desenvolvimento e tem se tornado uma importante área de pesquisa, contribuindo para a melhoria das atividades que envolvem a construção, manutenção e evolução de sistemas de software.

Neste contexto, Dabbish (2012) descobriram que as pessoas frequentemente fazem inferências sociais a partir de informações extraídas de atividades da rede no Github, como inferir metas e visões técnicas de outra pessoa ao editar o código ou adivinhar qual de vários projetos semelhantes tem a melhor chance de prosperar no longo prazo. Além disso, eles combinaram estas inferências em estratégias eficazes para coordenar o trabalho, desenvolver habilidades e gerenciar carreiras.

Entende-se como *expertise* o grau de conhecimento de uma pessoa, em um determinado período de tempo, sobre uma determinada área de conhecimento. Segundo o dicionário Aurélio, *expert* ou especialista é *"1. que ou quem se dedica a uma ciência ou arte; 2. que ou quem se especializou em determinada área do saber ou sabe muito sobre determinada coisa."* [Ferreira 2004].

No contexto de software, Mockus et. al. (2012) definem expertise como a habilidade do desenvolvedor de ser especialista sobre um código fonte, refletindo na habilidade de uma pessoa em executar determinada tarefa.

2.2.1 Expertise em um projeto de software

Nessa seção iremos apresentar os trabalhos relacionados a *expertise* em um único projeto de software. Será apresentado um breve resumo de cada um desses trabalhos.

A maioria dos projetos de desenvolvimento de software é composto por uma equipe com um ou mais desenvolvedores. Contudo, mesmo mantendo um padrão de documentação de projeto e código de forma detalhada, a cada atividade que precisa ser desenvolvida, a equipe costuma questionar quem seria o melhor profissional para desenvolvê-la. Encontrar especialistas em software é uma tarefa complexa, e nem sempre um bom currículo pode atestar que uma pessoa é um especialista na área.

Minto et al. (2007) realizaram um levantamento bibliográfico sobre expertise e classificaram em três tipos as abordagens para recomendar e identificar especialistas em projetos de software: as baseadas em heurísticas, as baseadas em redes sociais e as baseadas em máquinas de aprendizado. Eles apresentam o *Emergent Expertise Locator* (EEL), que usa informações de equipes emergentes para propor especialistas a um desenvolvedor dentro de seu ambiente de desenvolvimento à medida que o desenvolvedor trabalha. Com isso, eles descobriram que o EEL produz, em média,

resultados com maior precisão e maior recordação do que uma heurística existente para recomendação de especialista.

Para Santos et al. (2015), as mais disseminadas são as abordagens baseadas em heurísticas que são aplicadas sobre os dados coletados a partir do histórico do desenvolvimento. Uma das mais simples é baseada na ideia de que especialista é aquele desenvolvedor que foi o último a modificar um dado código. Essa foi a abordagem utilizada por McDonald et al. (2000), que propuseram ainda uma arquitetura para armazenar histórico de mudanças e delas identificar especialistas para solucionar problemas similares.

Mockus et al. (2002) relatam a problemática de encontrar experiências relevantes como necessidade crítica na engenharia de software, principalmente quando há desenvolvedores distribuídos geograficamente. Eles apresentaram uma ferramenta que utiliza dados de um sistema de gestão de mudanças para localizar pessoas com a experiência desejada. Eles propuseram então usar uma quantificação da experiência, denominada de átomos de experiência (EA), e apresentaram evidências para validar essa quantificação como medida de expertise. O átomo de experiência é uma unidade elementar da experiência, sendo a menor unidade significativa de mudanças no código fonte. Os autores definiram que a experiência sobre um objeto é uma coleção de todas as unidades elementares pertencentes a esse objeto. A ferramenta desenvolvida a partir de EA fornece uma maneira de identificar conhecimentos relacionados ao projeto, podendo descobrir facilmente pessoas que trabalham em partes específicas de um projeto, podendo também compreender como a experiência de um desenvolvedor é distribuída sobre o produto.

McDonald et al. (2000) relatam em seu trabalho que localizar o conhecimento necessário para resolver problemas difíceis é considerado um problema social e colaborativo. Neste trabalho, eles descrevem uma arquitetura de recomendação geral que é fundamentada no estudo de localização de especialidade. Nesse sistema de recomendação, os autores detalham o trabalho necessário para utilizar a recomendação de experiência em um ambiente de trabalho. A arquitetura e implementação começam a desvendar os aspectos técnicos de fornecer boas recomendações, observando tantos aspectos sociais quanto colaborativos.

Fritz et al. (2010) buscam identificar o grau de conhecimento de especialistas em um dado artefato de código combinando o grau de autoria (DOA, do inglês, degree-of-authorship) e o grau de interesse (DOI, do inglês, degree-of-interest). O DOA mensura o conhecimento adquirido pelos desenvolvedores decorrentes das mudanças por eles concretizadas sobre os elementos de código. Já o DOI aponta o interesse imediato dos desenvolvedores sobre um dado elemento. Eles propõem um sistema de recomendação com base no grau de conhecimento (DOK, do inglês, degree-of-knowledge) que calcula automaticamente, para cada desenvolvedor, um determinado valor real para cada parte do código. Eles argumentam que esse modelo de conhecimento pode proporcionar melhores resultados que formas até então existentes de encontrar especialistas e também apresentam um relato sobre o estudo de

caso utilizado para avaliação.

Ehrlich et al. (2008) estudaram como as pessoas utilizam ferramentas para encontrar especialistas, não apenas na área de software, mas de uma forma geral. Eles entrevistaram pessoas e pesquisaram algumas ferramentas que fazem busca por especialistas e encontraram diretórios corporativos e redes pessoais, que são mais frequentemente citadas como alternativas para localizar um especialista. A partir dessas descobertas, eles encontraram razões para fazer uso dessas ferramentas, tais como obter respostas às questões técnicas ou reconhecer pessoas que podem fornecer informações relevantes. As ferramentas apoiam a ideia de que a busca de conhecimentos é mais do que apenas encontrar a maioria das pessoas competentes, é possibilitar novas formas de pensar em ferramentas para localizar informações de especialistas.

Para Alonso et al. (2008), a evolução de software costuma gerar um problema considerável. Em grandes projetos open source, as pessoas que possuem um alto nível de conhecimento, são consideradas especialistas. Desta afirmativa surgem as dúvidas de como identificar os especialistas pela mineração de um determinado repositório de código fonte. O código fonte do software possui informações valiosas para descobrir contribuintes e atividades no projeto. Realizando extração e classificação de informações importantes sobre como o trabalho é realizado, é possível identificar as áreas de conhecimento e de equipes de colaboradores de forma automatizada, através de um visualizador que permite explorar a descoberta de padrões. A partir dos commits, eles apresentaram um protótipo que identifica como os desenvolvedores podem ajudar outros, fornecendo uma plataforma de visualização que ajuda a explorar o repositório por meio de commits e categorias.

Para Robbes (2013), a especialização de um desenvolvedor de software é um fator crucial para o tempo necessário no desenvolvimento de uma tarefa. Os autores propuseram várias métricas para identificação de especialistas explorando repositório de código fonte. Eles propuseram usar o tempo necessário para a conclusão de uma tarefa como forma de medir a eficácia das métricas experimentais. Eles definiram duas métricas de especialização com base na atividade passada e constataram que ambas as métricas foram realizadas de acordo com o esperado.

2.2.2 Expertise de uso

Com a disseminação de repositórios sociais de software como o GitHub, é possível conjecturar que há possibilidades de "melhorar" a identificação de especialistas a partir do uso destas "novas" informações disponíveis. A experiência de uso é experiência do usuário determinada pelo grau de conhecimento que dele tem sobre determinado artefato reutilizável de software (e.g., frameworks, bibliotecas, componentes). Esta pode ser medida por quanto o desenvolvedor utilizou o artefato sobre os dados, adquirindo assim mais experiência de uso. Segundo Ma et al. (2009), para determinar a expertise de determinado usuário, não devemos levar em consideração apenas os commits dos usuários. É necessário conhecer melhor o que foi alterado no

código em questão para determinar a sua experiência de uso em determinada área de conhecimento.

Segundo Ma et al. (2009), identificar especialistas sobre um elemento de interesse (e.g., classe, arquivo, método) não precisa depender apenas dos commits que desenvolvedores fizeram sobre o código fonte alterando este elemento de interesse. Os autores fizeram um estudo empírico, comparando a precisão da experiência de uso com a experiência de desenvolvimento, utilizando os projetos do Eclipse e o AspectJ. Foi então desenvolvida uma ferramenta que utiliza dados de um sistemas de controle de versão para localizar pessoas com experiência desejada. Eles usam uma quantificação da experiência e apresentam evidências para validar esta quantificação como uma medida de especialização. A ferramenta permite que os desenvolvedores, por exemplo, facilmente distingam alguém que trabalhou apenas brevemente em uma determinada área do código de alguém que tem mais experiência, e para localizar pessoas com ampla experiência em grandes partes do produto, como um módulo ou mesmo subsistemas. Além disso, ela permite a um usuário descobrir perfis de especialização de indivíduos ou organizações [Ma et al. 2009].

2.3 Interfaces de Programação de Aplicação - APIs

Este trabalho pretende potencializar a localização de especialistas em artefatos reutilizáveis de software, mais especificamente, em suas APIs. Define-se Interface de Programação de Aplicação (API, do inglês, *application programming interface*) como um meio de reutilização de código fonte, fornecendo interface, recursos e funcionalidades em projetos que queiram utilizá-los. Não foram encontrados muitos trabalhos de identificação de especialistas em APIs do software utilizando repositórios de código. No entanto, essa seção irá explorar trabalhos correlatos onde há referências ao uso de APIs para a engenharia de software.

Um dos principais trabalhos estudados, buscando focar em bibliotecas de terceiros, foi o trabalho de Teyton et al. (2013) . Motivados pela grande dependência dessas bibliotecas, estes autores propuseram um modelo para determinação de expertise em bibliotecas baseada na análise de commits. Para desenvolvimento de seu trabalho, Teyton et al. (2013) consideram que o uso de códigos da biblioteca demonstra o conhecimento do desenvolvedor sobre ela. Assim, eles propuseram uma métrica para calculo da expertise baseado no número de símbolos usados pelo desenvolvedor. Além da métrica de expertise, apresentaram um método para comparar a expertise dos desenvolvedores. Usando distância euclidiana entre o conhecimento total de várias bibliotecas e o conhecimento atual de um dado desenvolvedor, eles propuseram definir o grau de conhecimento dos desenvolvedores para diversas bibliotecas. Neste artigo, eles implementaram e propuseram o LIBTIC, um mecanismo de pesquisa de especialistas em bibliotecas, preenchido automaticamente por mineração de reposi-

tórios de software. Eles mostraram que o LIBTIC encontra especialistas relevantes de bibliotecas entre os desenvolvedores do GitHub.

Dentro de um contexto de mineração de uso de APIs, Zhong et al. (2009) desenvolveram uma estrutura de suporte chamada MAPO (Padrões de uso da API de mineração de repositórios de código aberto) para capturar padrões de uso de API automaticamente. MAPO recomenda ainda os padrões de uso da API minerada e seus padrões de código associados às solicitações dos programadores. Os resultados experimentais mostram que, com esses padrões, MAPO ajuda os programadores a localizar trechos de código úteis de maneira mais eficaz do que duas ferramentas de busca de código. Para investigar se a ferramenta pode auxiliar os programadores nas tarefas de programação, os autores realizaram um estudo empírico. Os resultados mostram que, usando MAPO, os programadores produzem código com menos erros quando enfrentam usos de API relativamente complexos, comparando com o uso das duas ferramentas de pesquisa de código.

2.4 Avaliação do uso de métricas de expertise

Essa sessão explicará como se deu o processo de avaliação dos principais trabalhos relacionados a essa pesquisa. O entendimento deste processo é importante para que possamos compreender como foi realizado o processo de avaliação das técnicas utilizadas por eles, assim como os resultados obtidos.

Um dos principais trabalho base para essa pesquisa é o trabalho de Ma et al. (2009). Eles apresentam um trabalho onde a recomendação de especialistas não é apenas baseada em quantitativos de commits de código fonte. Eles apontam que considerar o uso de métodos (ou símbolos, de modo mais geral) é uma heurítica útil para determinar o acúmulo de expertise. Para isso, eles definiram seis métricas importantes para a recomendação de especialistas, sendo seis delas consideradas métricas de uso e duas consideradas métricas de implementação. Como avaliação experimental, Ma et. al (2009) utilizam dois projetos: Eclipse e AspectJ. Para isso, eles processaram os dados de um período de Janeiro de 2006 até Outubro de 2006 para a geração das métricas. Em seguida, eles analisaram os commits posteriores para verificar se as métricas estavam ”acertando”na recomendação de especialistas. Utilizando essa técnica, eles observam que é possível obter um percentual de acerto satisfatório utilizando essas métricas, obtendo resultados de 50% já na primeira validação. Essa técnica de olhar para o “futuro” e verificar se a métrica “acerta” na recomendação foi considerada inspiradora, e nos levou a avaliar a proposta deste trabalho também desta forma.

Um outro trabalho que auxiliou na avaliação e condução dessa pesquisa, foi o trabalho de Teyton et al. (2013) . Eles apresentam na sua pesquisa o processo de recomendação de especialistas em APIs, porém apresentando uma abordagem diferente na proposta por essa pesquisa. A proposta apresenta por eles é de obter um

conjunto de APIs de software utilizados por projetos e recomendar a pessoa que mais conhece sobre determinada API. Para o processo de avaliação experimental, os autores utilizaram de duas técnicas: a primeira, extraindo projetos do Github e verificando se os desenvolvedores recomendados já usaram a API através do Github. A segunda, encontrando em determinado projeto quem sabe sobre uma determinada biblioteca ou sobre mais de uma biblioteca. Para a primeira estratégia citada, eles utilizaram a API REST do Github para buscar os desenvolvedores que utilizaram cada simbolo da API. Mapearam em relação a desenvolvedor vs. API vs. símbolos nos projetos selecionados. Eles relatam que a grande maioria dos programadores (77%) usam apenas poucas bibliotecas (entre 1 e 5 bibliotecas) e menos de 2% usam mais de 20 bibliotecas. Já considerando a outra abordagem apresentada pelos autores, eles estavam interessados em encontrar especialistas em uma ou mais bibliotecas. Para isso, os autores escolheram aleatoriamente 3 bibliotecas e selecionaram 20 projetos que as utilizavam. Para cada uma das bibliotecas, eles executaram uma *query* do LABTIC para encontrar especialistas através da ferramenta. Com a resposta da ferramenta, em forma de ranking, eles enviaram um e-mail para os desenvolvedores que estavam presentes na 3 bibliotecas citadas para responder a um questionário onde eles teriam que informar um valor entre zero e cinco mensurando o grau que cada um se considerava especialista na API. Eles obtiveram 50% de respostas, informando que quem respondeu, de fato, se considerava especialista na API.

Um outro artigo com design experimental de avaliação interessante é o artigo de Fritz et. al. (2010) . Eles apresentam uma métrica principal para determinar o quanto os desenvolvedores conhecem sobre determinado código, chamada de grau de conhecimento (DOK). Essa métrica DOK é computada a partir das interações dos desenvolvedores com o código fonte e a partir das informações provenientes de serviços de revisão de código. Para avaliação, os dados utilizados por eles foram extraídos de dois times de desenvolvimento de software da IBM. Com isso, os autores fizeram uma regressão linear múltipla para quantificar a experiência dos desenvolvedores. Essa regressão linear tentou encontrar uma equação linear da expertise dos desenvolvedores com base em quatro variáveis: FA (*primeira autoria*), DL (*deliveries*), AC (*acceptances*) and DOI (*grau-de-interesse*). Com essa equação definida, experimentos foram realizados com as equipes da IBM. Eles pediram para cada desenvolvedor da equipe classificar 40 elementos de código aleatórios, e, para cada um desses elementos, eles calcularam o valor do DOK com base na equação encontrada. Com estes valores, eles utilizaram a correlação de Spearman para comparar os rankings dados pela equação com os rankings fornecidos explicitamente pelos desenvolvedores. Como resultado, eles identificaram que existe uma correlação de 0.3847, correlação considerada média. Além de se utilizar desta correlação para a avaliação dos dados, Fritz et. al. (2010) também realizaram estudos de caso para entender cenários onde é possível utilizar o DOK. Em um estudo de caso eles buscaram identificar qual o membro da equipe que mais sabia sobre determinada parte do código. Para essa situação, eles compararam pacotes previstos no modelo DOK aos atribuídos pelos desenvolvedores relados de forma "manual". Com isso,

eles verificaram que 55% dos resultados computados concordam com os fornecidos pelos desenvolvedores. Uma outra situação investigada pelos autores é se os valores DOK de um desenvolvedor podem ser usados para selecionar as mudanças de interesse para o desenvolvedor por conta da sobreposição entre a mudança de código fonte e o modelo DOK do desenvolvedor. Para isso, os autores calcularam o DOK para cada um de três desenvolvedores de uma das equipes de desenvolvimento. O modelo DOK forneceu informações relevantes para os desenvolvedores em quatro de seis casos, gerando poucos erros comparados à base fornecida pelos desenvolvedores dos valores DOK.

Um trabalho recentemente publicado e que também vale a pena ser avaliado é o trabalho de Santos et al. (2018). A proposta do trabalho deles é ranquear especialistas em bibliotecas com base em conhecimento que os desenvolvedores produzem a partir do GitHub e cruzando esses dados com as informações coletadas a partir do LinkedIn. Como avaliação experimental, foi realizado um experimento para identificar possíveis especialistas em três bibliotecas. Os autores classificaram os 100 maiores desenvolvedores para cada tecnologia e em seguida, compararam os perfis selecionados do GitHub com os perfis destes mesmos desenvolvedores na rede social LinkedIn para verificar se as habilidades apontadas correspondem às apresentadas no GitHub. Eles também fizeram uma pesquisa com estudantes de uma universidade para verificar se o método de recomendação proposto é válido. Segundo os autores, os resultados mostram que 89% dos desenvolvedores do GitHub selecionados relatam suas habilidades em sites de redes sociais como o LinkedIn.

Capítulo 3

Metodologia de Pesquisa

O processo metodológico deste trabalho foi dividido em seis etapas: (1) Revisão bibliográfica, (2) Escolha das métricas, (3) Construção da ferramenta para recomendação de especialistas em APIs de software, (4) Validação da ferramenta, (5) Avaliação da ferramenta e (6) Avaliação das métricas.

3.1 Revisão bibliográfica

Para a melhor compreensão do cenário a ser estudado, é necessário um entendimento sobre os trabalhos relacionados e revisão da bibliografia.

Com isso, buscando encontrar os trabalhos relacionados e entender a bibliografia a cerca do tema, foram utilizadas bases de dados bibliográficas para busca direta por trabalhos relacionados por meio de palavras chaves. A partir desses trabalhos encontrados, buscamos citações e referências a eles até chegar a uma abrangência considerada satisfatória que cobria os trabalhos referentes a expertise de software, sistemas de recomendação, APIs de software e repositórios sociais de código fonte.

3.2 Escolha das métricas

Com essa categorização definida, cada artigo sobre expertise de software foi relido a fim de entender melhor cada uma das métricas por eles utilizadas. Este entendimento foi dado através discussões em grupo, resumo dos artigos e debates.

Com este processo, identificamos alguns pontos em comum e outros divergentes entre os artigos e, mais particularmente, entre as métricas utilizadas por eles, principalmente quando abordavam a recomendação de especialistas. Para que possamos entender melhor estas semelhanças e divergências, mapeamos algumas perguntas

que poderiam nos auxiliar a entender melhor quais as entradas mais utilizadas por cada uma das métricas e como elas poderiam ser agrupadas.

As partir desse questionamento realizado, identificamos quais as entradas mais utilizadas de cada uma das métricas mapeadas. Com as principais entradas mapeadas, conseguimos identificar as métricas que mais utilizam estas entradas e consideramos estas as métricas escolhidas para esta pesquisa.

Foram então selecionadas seis métricas, sendo elas concentradas nos trabalhos dos autores de Ma et al. (2009) e Teyton et al. (2013). Todo o detalhamento da escolha das métricas é definida no Capítulo 4. São elas:

Tabela 3.1: Métricas implementadas a partir do trabalho Expert recommendation with usage expertise [Ma et al. 2009]

Métrica	**Modelo**	**Descrição**
Depth of Method Knowledge	$E_d(Q) = \sum_{m \in Q} ufreq_d(m)$	Para todos os métodos de uma API, calcula a soma das frequências de uso de cada método por um desenvolvedor.
Breadth of Method Knowledge	$E_d(Q) = \|m \| m \in U_d, m \in Q\|$	Calcula a soma da quantidade de métodos distintos, de uma API, usados por um desenvolvedor
Relative Depth of Method Knowledge	$E_d(Q) = \sum_{m \in Q} \frac{ufreq_d(m)}{ufreq_D(m)}$	Calcula a frequência de uso de um desenvolvedor com relação a frequência dos outros desenvolvedores.
Relative Breadth of Method Knowledge	$E_d(Q) = \sum_{m \in Q} \frac{c_d(m)}{\sum_{d \in D} c_d(m)}$	Calcula a quantidade de métodos distintos usados pelo desenvolvedor em relação a todos os métodos usados no projeto pelos outros desenvolvedores.

Tabela 3.2: Métricas implementadas a partir do trabalho Find Your Library Experts [Teyton et al. 2013]

Métrica	**Modelo**	**Descrição**
Library Expertise	$e(d, apis) = \frac{\lvert u(d,apis)\rvert}{\lvert p(apis)\rvert}$	Relação entre os métodos usados por um desenvolvedor e todos os métodos da API.
Expertise Distance	$Ed(d, apis) = \lvert \overrightarrow{1} - \overrightarrow{e}(d, apis)\rvert$	Calculo da distância euclidiana dos da Library Expertise de cada desenvolvedor

3.3 Construção da ferramenta

Foi desenvolvida uma ferramenta capaz de extrai commits de projetos originados no Github e calcula as métricas identificadas na etapa anterior para a recomendação de especialistas em APIs de software.

A ferramenta foi desenvolvida em Python e pode extrair commits em projetos escritos em Java, PHP, JavaScript ou Python (a linguagem pode ser parametrizável). A busca pelos dados brutos existente no repositório Git é feita executando scripts bash executados a partir de chamadas Python. Presentemente, a ferramenta necessita ter acesso a um repositório Git para conseguir extrair as informações necessárias. Atualmente os dados extraídos são armazenados em arquivos CSV.

O processo de recomendação de especialistas em API de software apresentado nesse projeto está baseado na análise das seis métricas identificadas anteriormente.

A partir dessas métricas, e exatamente como elas foram definidas pelos autores, a ferramenta apresenta um ranking dos desenvolvedores extraídos para cada uma das métricas.

Neste primeiro momento, o objetivo da ferramenta é apenas recomendar especialistas em API de software com base em cada uma dessas métricas.

3.4 Validação da ferramenta

Em paralelo ao desenvolvimento da ferramenta, iniciou-se o processo de testes. Para os testes, foram desenvolvidos dois projetos fictícios e controlados para que fosse possível validar se as interações dos desenvolvedores com o código estavam sendo computadas corretamente pela ferramenta a fim de gerar os resultados corretamente.

Como API de software utilizada para testes, foi utilizada a API do log4j. Essa API foi escolhida por ser de fácil uso, com algum métodos já conhecidos por nós, e que não teria um custo alto para entendimento e uso.

Para os testes, consideramos três desenvolvedores fictícios. Em cada commit manipulado, era registrado manualmente em um arquivo do tipo texto todo o histórico da simulação realizada. Todo este histórico foi útil para validação de cada commit dos desenvolvedores.

Durante todo o processo de validação da ferramenta, a cada problema encontrado, os erros foram sendo corrigidos, falsos positivos foram sendo removidos, até que a ferramente estivesse razoavelmente validada.

A ferramenta foi dada como concluída e validada quando todos os cenários de testes definidos previamente foram cobertos e quando todas as métricas foram validadas manualmente.

3.5 Avaliação da ferramenta

Após a validação da ferramenta por meio de projetos controlados, foi necessária a sua validação em um projeto real. Para isto, uma empresa privada disponibilizou um de seus projetos para que fosse validada a ferramenta, assim como disponibilizou seu time de desenvolvimento para responder a questionários para entendimento e definição do oráculo.

Com isso, o primeiro passo para definição do oráculo foi entender quais as APIs mais utilizadas no projeto e selecionar as duas mais utilizadas para que fosse possível validar a recomendação por meio da ferramenta. As APIs selecionadas foram: JQuery e Google Maps.

Após definidas as APIs, foi enviado um questionário para os desenvolvedores responderem (em uma escala de 0 a 10) com a conhecimento que cada desenvolvedor considera que os outros desenvolvedores têm em relação às APIs. Com estas respostas computadas, nós definimos o oráculo para cada API e assim comparamos com os resultados de cada métrica extraída pela ferramenta. Todo esse processo avaliativo está detalhado no Capítulo 5 deste trabalho.

3.6 Avaliação das métricas

Definindo as métricas, e avaliando a ferramenta com essas métricas geradas, se fez necessário uma avaliação dessas métricas para entender o comportamento delas em alguns cenários de desenvolvimento de software.

Para que pudéssemos avaliar se as métricas, e consequentemente o seu ranking de recomendação, está coerente com a realidade nós optamos por algumas estratégias. Essas estratégias, que serão explicadas no Capítulo 06, foram realizadas buscando responder a algumas perguntas. Nós primeiramente optamos perguntar aos desenvolvedores dos projetos e APIs envolvidas algumas perguntas a fim de entender o comportamento das métricas. Em seguida, nós utilizamos algumas ferramentas da estatística para avaliar e analisar o comportamento também das métricas: a correlação de Spearman e através do cálculo da precisão média. Todo esse processo avaliativo está definido no Capítulo 6 deste trabalho.

Capítulo 4

Métricas para Determinação de Expertise

Em estudos anteriores, descritos na seção do referencial teórico, nós observamos diferentes métodos e métricas que identificam expertise em projetos de software. Neste capítulo, propomos uma classificação e mapeamento dos *inputs* e métricas usadas por esses estudos. Esse mapeamento se tornou necessário porque cada um dos artigos apresentados não detalham os inputs que foram utilizados em suas métricas. O resultado descreve um cenário de como a área de identificação de especialistas abordam o tema.

Primeiro mostramos os critérios que usamos para descobrir os insumos utilizados para avaliação e classificação das métricas. Então, para cada artigo revisado, as relações de cada métrica com os *inputs* identificadas são analisadas. Finalmente, descrevemos as semelhanças entre métricas com base em seus *inputs* e classificação de métricas.

4.1 Identificação dos *inputs*

Para iniciar o processo de identificação de *inputs* se fez necessário definir quais os artigos seriam selecionados para esta avaliação. Esse processo de seleção dos artigos levou em consideração os artigos que apresentaram métricas de recomendação de especialistas de software consideradas comuns e aderente a proposta dessa pesquisa.

Nossa revisão mostrou pontos comuns e divergentes sobre como lidar com cada um dos *inputs* e métricas identificadas. Para realizar nosso mapeamento, durante a leitura de cada artigo, nós identificamos questões que ajudam a compreender os *inputs* apresentados em cada artigo. Durante esse mapeamento, foram realizados *brainstormings* com a equipe de pesquisa a fim de definir qual o melhor questionamento a ser realizado a cada artigo para extrair os *inputs* corretos. No final, lemos os artigos

Tabela 4.1: Codificação dos artigos mapeados

Código	Artigo
ERU	Expert Recommendation with Usage Expertise [Ma et al. 2009]
FLE	Find your Library Experts [Teyton et al. 2013]
EBA	Expertise Browser: A Quantitative Approach to Identifying Expertise [Mockus e Herbsleb 2002]
DOK	A Degree-of-Knowledge Model to Capture Source Code Familiarity [Fritz et al. 2010]
DCS	Using Developers' Contribution on Software Vocabularies to Identify Experts [Santos et al. 2015]
RET	Recommending Emergent Teams [Minto e Murphy 2007]

mais uma vez para responder as questões levantadas. Essas questões ajudaram a esclarecer as semelhanças existentes nos modelos apresentados pelos artigos, bem como as discrepâncias. A Tabela 4.2 mostra as questões que foram levantadas para a identificação desses *inputs*, com cada artigo codificado com a legenda apresentada na Tabela 4.1.

Tabela 4.2: Questões levantadas para a Identificação de Especialistas

	ERU	**FLE**	**EBA**	**DOK**	**DCS**	**RET**
Considera mais de um projeto para identificar especialista?		X				
Considera API / Biblioteca para recomendar especialista?		X				
Investiga histórico de commit?				X		X
Analisa commit de código?	X	X	X	X	X	X
Analisa o conteúdo dos arquivos alterados por commit?	X	X			X	X
Analisa criação de trecho de código que chama método da API?		X				
Considera perda de expertise?				X		
Considera que o tempo é um fator importante para medir os conhecimentos?	X					
Considera a autoria de código um fator de expertise?				X		

A partir dessas questões, percebemos que existe uma similaridade entre as abordagens apresentadas em cada artigo em relação a cada uma das questões apresentadas. Nós classificamos essas semelhanças e as relacionamos com as entradas específicas para cada métrica. Este conjunto de *inputs* tem como objetivo classificar as várias

abordagens apresentadas nesses artigos para identificar especialistas em desenvolvimento de software.

A partir desse entendimento da relação entre os artigos e as métricas, foram mapeados nove *inputs* com base nos critérios acima. Esses *inputs* são usados em pelo menos um dos artigos. Cada artigo analisado aborda pelo menos um *inputs* em suas métricas. Esses *inputs* são:

1. **Projeto:** Caracteriza o uso de um ou mais projetos de software como *inputs* para o resultado da métrica.
2. **Commit:** Caracteriza o uso da análise do conjunto de mudanças realizadas pelo CSV (seja ele proveniente de qualquer Sistema de Controle de Versão).
3. **Timestamp:** Considera aspectos temporais como fator importante para o resultado da métrica. Ex. Tempo em que o *commit* foi realizado.
4. **Desenvolvedor:** Considera as informações do desenvolvedor de software (extraídas do projeto) importantes como resultado da métrica.
5. **Criação de Contêineres:** Considera a autoria de qualquer arquivo do projeto (e.x., arquivo, classe, método) pelo desenvolvedor como informação relevante para o resultado da métrica.
6. **Mudanças de Contêineres:** Considera toda e qualquer alteração de arquivos de projeto por um desenvolvedor como informação relevante para o resultado da métrica.
7. **Uso de Símbolos:** Considera toda e qualquer alteração de símbolos (métodos, atributos, variáveis) pelo desenvolvedor como informação relevante para o resultado da métrica.
8. **Frequência de Mudança:** Considera relevante a quantidade de vezes que um desenvolvedor altera determinado método/atributo ou arquivo de um projeto para resultado da métrica.
9. **Interação:** Considera a quantidade de interações (uso) por meio de uma IDE[1] que determinado desenvolvedor realizou sobre o código fonte de um determinado projeto.

4.2 Relacionamento entre os *inputs* e métricas

Depois de identificar os *inputs*, precisamos recordar quais estão relacionados a cada artigo. Para fazer isso, extraímos as métricas usadas por cada artigo para determinar os conhecimentos, relacionando-os com os *inputs* anteriores. A Tabela 4.3 apresenta

[1] Ambiente integrado de desenvolvimento de Software

a definição de cada uma das métricas extraídas de cada artigo. A definição formal de cada uma das métricas pode ser vista nos artigos relacionados.

Ma et al. (2009) faz uma comparação de dois grupos de métricas: métricas de uso e métricas de implementação. As métricas de uso fornecem uma visão quantitativa da interação do desenvolvedor com o projeto. Algumas métricas de uso consideram as interações de outros desenvolvedores, calculando valores relativos dentro do projeto. As métricas de uso incluem métricas de *profundidade relativa* e *largura*, que levam em consideração outros desenvolvedores de projetos e de *profundidade não-relativa* e *largura*, que dependem apenas do desenvolvedor em questão. As métricas de implementação incluem *frequência de mudança* e *mudanças recentes*, que representam, respectivamente, o número de alterações em cada método "comitado"pelo desenvolvedor e o tempo decorrido desde a última alteração.

Teyton et al. (2013) preferem examinar a experiência de forma mais generalizada. Enquanto outros autores tratam o número de usos de cada símbolo, eles consideram que o uso da unidade de um símbolo é suficiente para conhecê-lo. A partir dessa ideia, apresentam duas métricas que não levam em conta o volume, mas sim a ocorrência do uso de símbolos. A primeira métrica, *library expertise*, consiste no número de símbolos usados por um desenvolvedor sobre o número de símbolos existentes em uma biblioteca. A segunda métrica, *expertise distance*, é a distância euclidiana do ponto de conhecimento do desenvolvedor ao ponto que reflete o conhecimento completo de uma ou mais bibliotecas.

Em 2002, Mockus et al. introduzimos uma ferramenta que usa dados de sistemas de gerenciamento de mudanças para localizar pessoas com o conhecimento desejado. Eles usaram a quantificação da experiência e apresentaram evidências par validar essa quantificação como uma medida de *expertise*. Essa ferramenta permite que os desenvolvedores, por exemplo, facilmente distingam alguém que tenha trabalhado apenas brevemente em uma área específica do código de alguém que tenha uma experiência mais extensa e localize pessoas com ampla experiência em grandes partes do produto, como módulo ou até subsistemas. Com isso, eles trouxeram a ideia do átomo da experiência (EA), representando a menor unidade de conhecimento. Para eles, esses átomos são capazes de quantificar qualquer domínio de conhecimento dentro de um software através da sua soma. Cada átomo representa uma alteração de arquivo cometida por um desenvolvedor.

Nota-se que os insumos utilizados pelos autores são semelhantes e, em vários casos, os mesmos. Para Fritz et al. (2010), a experiência pode ser determinada como a combinação de dois fatores, autoria e interação. Assim, eles determinam que as influências que o uso do código deve ser diferentes da implementação e que os desenvolvedores afetam suas experiências simultaneamente. Ao alterar o código criado por outro desenvolvedor, o desenvolvedor adiciona experiência a si próprio e reduz a experiência do autor do código. Por outro lado, a interação não altera a experiência de autores anteriores, só aumenta a experiência do próprio usuário do código. Eles resumem esses tipos de conhecimentos no grau de conhecimento (DOK)

de um desenvolvedor sobre um arquivo, que é a combinação de autoria e interação.

Minto et al. (2007) relatam que para construir sistemas de software complexos e bem sucedidos, os desenvolvedores devem colaborar uns com os outros para resolver problemas. Os autores apresentam o *Emergent Expertise Locator* (EEL), que usa informações de equipes emergentes para propor especialistas a um desenvolvedor dentro de seu ambiente de desenvolvimento à medida que o desenvolvedor trabalha. Eles descobriram que o EEL produz, em média, resultados com maior precisão e maior recordação do que uma heurística existente para recomendação de *experts*. Para fazer isso, eles modelaram os dados no formato de matrizes de relacionamento. Essas matrizes listam os arquivos de projeto e os desenvolvedores que os alteraram. Assim, eles podem determinar conhecimentos calculando a distância do desenvolvedor para um arquivo.

A tabela 4.3 apresenta uma breve definição de cada uma das métricas utilizadas. A tabela 4.4 mostra a relação entre as métricas com cada um dos artigos apresentados em relação aos *inputs*.

A partir da análise das tabelas 4.3 e 4.4, é possível observar que existem algumas semelhanças entre as métricas nos artigos e os *inputs* identificados. A primeira semelhança identificada é que quase todos os artigos consideram *commit* e *desenvolvedor* como entrada para cada métrica, sendo DOI a única exceção. Este é um conjunto mínimo típico de *inputs* necessários para a classificação das métricas de expertise.

Uma outra semelhança identificada está relacionada aos artigos ERU e FLE, em que a maioria das métricas usam como *inputs commit*, *desenvolvedor* e *uso de simbolos*. Isso porque a maioria das métricas se preocupam com o uso de símbolos, tentando detectar especialistas através das chamadas de método. No entanto, as métricas LE e ED do artigo FLE também consideram o projeto um *input* importante porque eles usam mais de um projeto para detectar e recomendar conhecimentos. VE é diferente dessas métricas, mas é de alguma forma semelhante ao LE e ED porque considera *projeto* como uma importante métrica também.

Outra similaridade encontrada é que as métricas MCF, MRC, SEA, DOA, DOK, FA e EM, são diferentes das métricas citadas acima, elas usam *mudanças de contêineres* para obter todas as alterações realizadas por um desenvolvedor em um recipiente dentro de um projeto. As métricas DOA e DOK também consideram a entrada *criação de contêineres* porque consideram que a criação de contêiner implica na propriedade e que um proprietário provavelmente sabe mais sobre esse código do que outro desenvolvedor.

Observamos também que apenas as métricas DOI e DOK usam a entrada *interação*, porque são as únicas em considerar as edições do desenvolvedor no código-fonte por meio de um IDE. Nenhuma outra métrica considera este fator para recomendar conhecimentos.

Finalmente, percebemos que apenas um autor usou os *inputs timestamp* e *frequência de mudança* na mesma métrica, o DOK. Seus significados são diferentes entre si,

considerando que *timestamp* usa o tempo que um desenvolvedor fez algo no código, implicando memória recente, a *frequência de mudança* considera quantas vezes o desenvolvedor mudou algo no código, o que implica na quantidade de conhecimento.

Diante desse mapeamento e entendimento da relação dos *inputs* com as métricas, se fez necessário selecionar apenas parte dessas métricas para avaliação posterior dentro do processo de identificação de especialistas em API de software.

Os critérios de escolha das métricas que serão utilizadas nessa análise se deu pelos seguintes fatores: (1) Métricas que consideraram de alguma forma a recomendação de APIs de software, pois está diretamente relacionada a finalidade dessa pesquisa; (2) Métricas que consideram o uso um ganho de conhecimento, pois a nossa proposta visa analisar projetos extraídos do github através de iterações. Então, é de fundamental importância que métricas que já possuem esse conhecimento sejam levadas em consideração.

Diante desses critérios, as métricas DK, BK, RDK, RBK, LE e ED foram selecionadas para as avaliações e identificação de especialistas em APIs de software a partir de repositórios sociais de código fonte. Essas métricas selecionadas podem ser vistas na tabela 4.4, onde estão marcadas com uma cor diferente.

Este mapeamento, além de dar suporte à avaliação que será apresentada no Capítulo 6, representa também uma contribuição dessa dissertação, pois os artigos aqui citados apresentarem suas próprias métricas, mas nenhum outro trabalho realizou um mapeamento afim de classificá-los de acordo com as similaridades dos *inputs* adotados.

Tabela 4.3: Definição das métricas de cada artigo

[gray].8 Artigos	Métricas	Definição
ERU	**[DK]** Depth of method knowledge [Ma et al. 2009]	Para todos os métodos em uma consulta / biblioteca, é a soma das frequências do uso de cada método por um desenvolvedor.
	[BK] Breadth of method knowledge [Ma et al. 2009]	Para todos os métodos em uma consulta / biblioteca, é a soma do número de métodos distintos usados por um desenvolvedor.
	[RDK] Relative depth of method knowledge [Ma et al. 2009]	Para todos os métodos em uma consulta / biblioteca, é a soma das proporções da frequência do uso de cada método por um desenvolvedor sobre a frequência do uso do mesmo método por todos os desenvolvedores no projeto.
	[RBK] Relative breadth of method knowledge [Ma et al. 2009]	Para todos os métodos em uma consulta / biblioteca, é a soma das proporções de cada método distinto usado por um desenvolvedor sobre todos os métodos distintos do mesmo método por todos os desenvolvedores no projeto.
	[MCF] Method Change Frequency [Ma et al. 2009]	Para todos os métodos em uma consulta, é a soma das frequências da mudança de cada método por um desenvolvedor.
	[MRC] Most recent change(s) [Ma et al. 2009]	Para todos os métodos em uma consulta, é a soma dos *timestamps* mais recentes da mudança de cada método por um desenvolvedor.
FLE	**[LE]** Library expertise [Teyton et al. 2013]	Relação do número de métodos distintos em uma biblioteca usada por um desenvolvedor em todos os métodos fornecidos pela biblioteca.
	[ED] Expertise distance [Teyton et al. 2013]	Dado n bibliotecas, a distância euclidiana é o vetor entre expertises de biblioteca de um desenvolvedor sobre as n bibliotecas e o vetor de experiência completa de todas as n bibliotecas.
EBA	**[SEA]** Sum of Experience Atoms [Mockus e Herbsleb 2002]	Número de alterações feitas por um desenvolvedor em um recipiente (por exemplo, arquivo, subsistema).
DOK	**[DOA]** Degree-of-Authorship [Fritz et al. 2010]	Dado um arquivo e um desenvolvedor, é a soma ponderada da primeira autoria do arquivo pelo desenvolvedor, mais todas as alterações do arquivo pelo desenvolvedor, menos todas as alterações do arquivo por outros desenvolvedores.
	[DOI] Degree-of-Interest [Fritz et al. 2010]	Dado um arquivo e um desenvolvedor, é a soma das interações sobre o arquivo feito pelo desenvolvedor em um IDE.
	[DOK] Degree-of-Knowledge [Fritz et al. 2010]	Soma ponderada do grau de autoria e do grau de interesse.
DCS	**[VE]** Vocabulary Expertise [Santos et al. 2015]	Similaridade entre o vocabulário de código-fonte de um desenvolvedor e o vocabulário do código-fonte de uma entidade de código-fonte (por exemplo, arquivo, método, classe).
RET	**[FA]** File Authorship [Minto e Murphy 2007]	Número de vezes que um desenvolvedor modificou um arquivo.
	[EM] Expertise matrix [Minto e Murphy 2007]	Uma célula na matriz é quantifica pela quantidade de experiência que um desenvolvedor j tem em relação ao desenvolvedor i, ambos dando autoria e dependência de arquivos.

Tabela 4.4: Relacionamento das Métricas com os *inputs*

[gray].8 Artigos	Metricas	*Projeto*	*Commit*	*Timestamp*	*Desenvolvedor*
ERU	[HTML]FFE4E1[**DK**]	[HTML]FFE4E1	[HTML]FFE4E1X	[HTML]FFE4E1	[HTML]FFE4E1
	[HTML]FFE4E1[**BK**]	[HTML]FFE4E1	[HTML]FFE4E1X	[HTML]FFE4E1	[HTML]FFE4E1
	[HTML]FFE4E1[**RDK**]	[HTML]FFE4E1	[HTML]FFE4E1X	[HTML]FFE4E1	[HTML]FFE4E1
	[HTML]FFE4E1[**RBK**]	[HTML]FFE4E1	[HTML]FFE4E1X	[HTML]FFE4E1	[HTML]FFE4E1
	[**MCF**]		X		X
	[**MRC**]		X	X	X
FLE	[HTML]FFE4E1[**LE**]	[HTML]FFE4E1	[HTML]FFE4E1X	[HTML]FFE4E1	[HTML]FFE4E1
	[HTML]FFE4E1[**ED**]	[HTML]FFE4E1	[HTML]FFE4E1X	[HTML]FFE4E1	[HTML]FFE4E1
EBA	[**SEA**]		X		X
DOK	[**DOA**]		X		X
	[**DOI**]			X	X
	[**DOK**]		X	X	X
DCS	[**VE**]	X	X		X
RET	[**FA**]		X		X
	[**EM**]		X		X

Capítulo 5

Protótipo da Ferramenta

Neste capítulo, é apresentada uma visão geral do protótipo da ferramenta desenvolvida[1] para o processamento das métricas discutidas no capítulo anterior.

O protótipo da ferramenta desenvolvida deve ser capaz de extrair dados de projetos com foco em informações sobre o uso dessas APIs no projeto. A extração dos dados será feita de projetos de código aberto armazenados em repositórios Git. Implementamos esse protótipo em linguagem Python para o tratamento dos dados com o suporte de scripts Shell para que estes dados sejam extraídos.

Esse Capítulo foi dividido em quatro seções: (1) apresentação do processo utilizado para a extração da informação para a identificação de especialistas em APIs de Software, (2) a apresentação do protótipo desenvolvido e o seu funcionamento, (3) apresentação do oráculo utilizado para uma validação do protótipo, e (4) uma apresentação e análise dos resultados obtidos durante a validação da ferramenta. Essas seções estão descritas abaixo.

5.1 Processo para extração de informação

Para que seja possível processar todas as métricas para APIs de software e extrair dados a partir delas, definimos um processo para tal. Este processo permite a seleção de commits e extração dos dados de interesse dentro de repositórios Git. O processo segue o fluxo de escrita do código fonte e da estrutura de armazenamento das informações dentro dos repositórios. A representação deste processo pode ser visto na Figura 5.1.

Na Figura 5.1, é possível observar que para o processo ocorrer são necessários alguns *inputs* (assinatura da API, lista de métodos da API e arquivo de parâmetros). Com esses *inputs*, são processados todos os *commits* dos projetos dentro do período

[1] http://www.expertfindingongithub.com

estabelecido no arquivo de parâmetros para verificar se houve algum uso da API e, em seguida, são calculadas as métricas.

Este processo considera o projeto do ponto de vista do sistema de controle de versão (Git). A visão atual de um projeto não reflete toda a dinâmica pela qual os desenvolvedores passaram para que ela fosse atingida. Consideramos que o ganho de expertise é incremental, e que uma remoção de algo existente não anula a expertise ganha anteriormente.

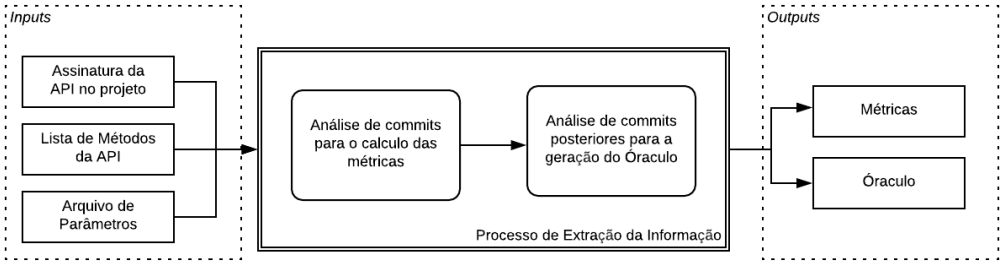

Figura 5.1: Representação do Processo para Identificação de Especialistas em APIs de Software

A extração de dados passa pelo entendimento do processo de escrita de código. Cada linguagem de programação, em sua estrutura sintática, necessita costumeiramente que fragmentos de código que serão utilizados no projeto sejam referenciados através dos *inputs*. Os *inputs* são a maneira de informar ao modelo de extração a localização dos códigos que serão usados, assim como a lista de métodos, a assinatura da API nos projetos, dentre outras informações relevantes que podem estar no arquivo de parâmetros.

Com isso, para que os dados sejam extraídos, o processo faz uma busca por todos os arquivos de *commit* para identificar aqueles que utilizaram código da API dentro do período informado no arquivo de parâmetros. A partir desses arquivos, são salvas estas informações em uma tupla, que posteriormente será utilizada para o cálculo das métricas. Essa tupla contém as seguintes informações:

- Desenvolvedor. O nome do desenvolvedor que utilizou a API;
- *Commit*. O *hash* correspondente ao *commit* identificado como contendo utilização da API;
- Email. Email do desenvolvedor que utilizou a API;
- Símbolo. O nome do símbolo da API utilizado;
- Frequência de Inseridos. A frequência de símbolos da API inseridos no *commit*;
- Frequência de Removidos. A frequência de símbolos da API removidos no *commit*;

- Data do *commit*. Data e hora em que o *commit* foi realizado;
- Quantidade de inseridos. A quantidade de símbolos da API inseridos no *commit*;
- Quantidade de removidos. A quantidade de símbolos da API removidos no *commit*;
- Ação. Informa, para a tupla, inserção ou remoção do símbolo da API.

Com a tupla definida, e com todos os *commits* registrados, o processo extrai as métricas já definidas. Os arquivos que foram extraídos permitem que os projetos sejam visitados de modo completo. Observar o histórico de *commits* feitos para cada arquivo é fundamental para determinar o real uso da API. Assim, a ferramenta extrai todos os commits que registraram alguma alteração com relação aos símbolos da API, alterações de inserção ou remoção.

Os *outputs* descritos na Figura 5.1. O cômputo das métricas foi apresentado no Capítulo 4. A geração do oráculo é apresentada na Seção 5.3.

5.2 Descrição do Protótipo da Ferramenta

Baseado no processo de extração, foi desenvolvido um protótipo de uma ferramenta capaz de extrair as informações necessárias para computação das métricas. Esse protótipo apresenta uma estrutura composta por três módulos principais, onde etapas previstas pelo processo são implementadas e etapas incrementais que se fazem necessárias são também executadas. Uma visão arquitetural dinâmica do protótipo da ferramenta pode ser vista na Figura 5.2.

A extração de dados passa pelo entendimento do processo de escrita de código. O processo de escrita de códigos em linguagem Java passa por três etapas: (1) Criar arquivos; (2) Referenciar pacotes de código do próprio projeto ou de API externas; (3) Fazer uso destes códigos referenciados. Fazer referência a códigos, denominado importação, é a maneira de informar ao projeto a localização das bibliotecas que serão usadas.

Assim, a primeira etapa para extração dos dados é a busca por arquivos que utilizaram pacotes de códigos da API. Esta primeira etapa é feita a partir da busca por *commits* que registraram a inserção ou remoção de linhas que contenham o caminho onde o pacote da API está localizado. Este tratamento, buscando o uso das importações, busca garantir que apenas os *commits* onde houve algum uso dos pacotes da API seja selecionado. Com isso, o histórico de uso da API no projeto é garantido.

Após a primeira etapa, todos os *commits* que registraram a inserção ou remoção de caminhos onde estão os pacotes da API deverão ser extraídos. Um *commit* registra as alterações em função dos arquivos, assim, é possível saber em qual arquivo foi

feita a alteração com o caminho da API. De posse dos *commits*, podemos extrair os arquivos que sofreram alguma alteração que foi registrada por este *commit*. Isso garante identificar todos os arquivos onde fez-se uso da API.

Observar o histórico de *commits* feitos para cada arquivo é fundamental para determinar o real uso da API. Assim, nesta etapa são extraídos todos os *commits* que registraram alguma alteração com relação aos símbolos da API, podendo ser alterações de inserção ou remoção. Como resultado desta etapa, temos uma lista dos *commits* necessários para a busca dos dados que devem ser extraídos para o cômputo das métricas.

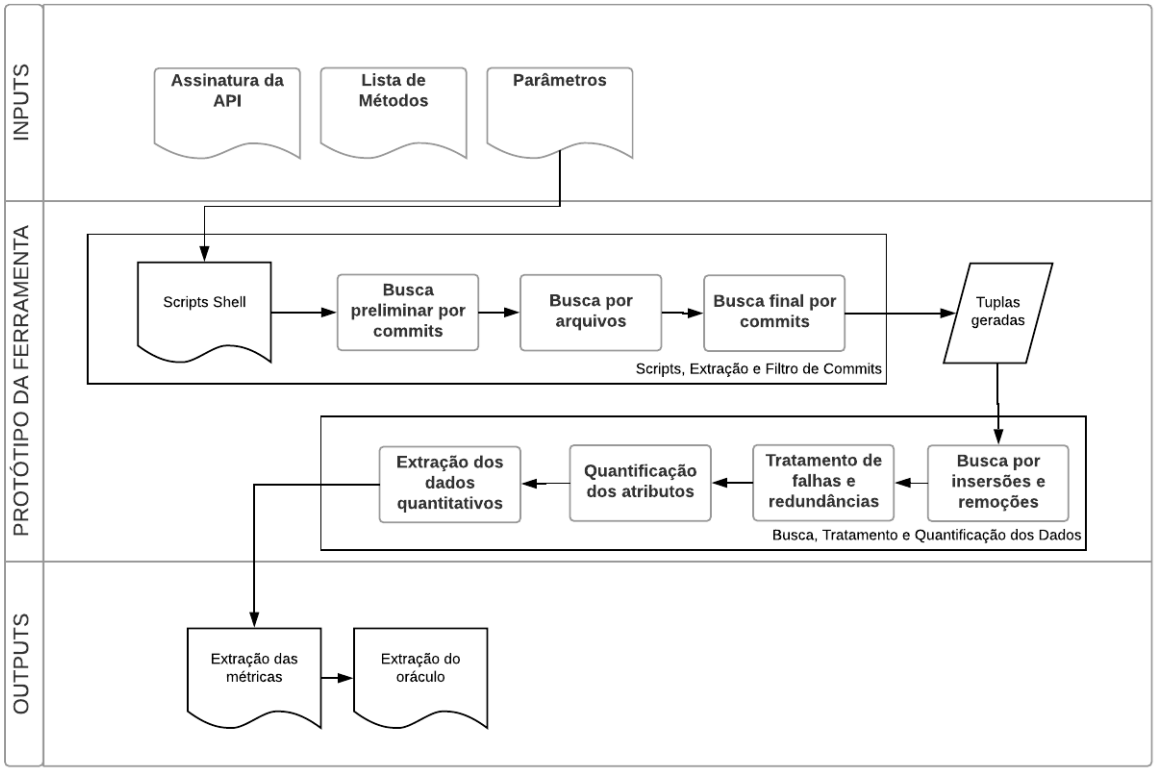

Figura 5.2: Visão Arquitetural Dinâmica do Protótipo da Ferramenta

Este protótipo tem a capacidade de extração de mais de um projeto para uma determinada API de software. Todo o processo é executado para cada projeto individualmente e os resultados são incrementados à base de dados durante o processo.

5.2.1 Scripts, Extração e Filtro de *Commits*

A etapa de extração e filtro de commits é realizada a seguir. Porém, para acesso aos projetos ela recebe o auxílio de uma camada escrita em *Shell Script*, para acesso

ao sistema operacional. Assim, foram produzidos scripts para acesso dos dados via Git. O uso dos scripts para acesso direto ao controle de versão dos projetos permite parametrizar a busca. Estes scripts são chamados e executados em segundo plano pela aplicação, escrita em Python. Do ponto de vista de praticidade de implementação, apesar da parametrização de scripts Git adicionar alguma complexidade, eles garantem que algumas etapas de tratamento, como filtragem e busca de strings não precise ser implementada em Python. Exemplo de scripts usados durante os processo de extração são apresentados nos Códigos 1, 2, 3.

Nas etapas de extração e filtragem de *commits*, determina-se quais *commits* são relevantes para a extração. Esta etapa é dividida em três fases:

- **Busca preliminar por *commits*:** Conforme a Seção 5.1, sobre o processo, esta é uma busca preliminar que serve para encontrar arquivos que possam ser de interesse. Para isso, um *shell script* é responsável por buscar os caminhos da API e fornecer os identificadores dos *commits* encontrados. O script pode ser visto no Código 1.

```
1        git -C <caminho_projeto>  log --since "<inicio>" --until "<fim>" --follow
          ↪  -G"<caminho_pacote>" --format=format:"%H|%an|%at" .]
```

Listing 1: Script usado para buscar commits que registraram importação da API.

- **Busca por arquivos:** Esta busca resulta na extração dos arquivos registrados nos *commits* da etapa preliminar. Aqui, todos os arquivos são extraídos, independente de sua existência no estado atual do projeto. Como pode ser visto no Código 2, a busca é feita de forma unitária para cada *commit*, buscando apenas os arquivos de extensão *.java*.

```
1 git -C <caminho_projeto> show --pretty="" --name-only <commit_id> | awk
   ↪  "/^.*\.(java)"
```

Listing 2: Script usado para extrair arquivos dos commits.

- **Busca final por *commits*:** Nesta etapa, os arquivos são verificados buscando alterações com uso dos símbolos da API pela execução do script presente no Código 3. A partir desta verificação, os *commits* são registrados para que, na próxima etapa, possam ser analisados e contabilizados os usos de símbolos e os desenvolvedores responsáveis por este uso. Ainda nesta etapa, os *commits* são catalogados iniciando a extração dos dados da Tabela **??**, extraindo-se nome do autor, ID do *commit*, timestamp do *commit*.

```
1 git -C <caminho_projeto>  log --since "<inicio>" --until "<fim>" --follow
   ↪  -G"<simbolos>" --format=format:"%h|%an|%at" <caminho_arquivo>
```

Listing 3: Script usado para extrair arquivos dos commits.

5.2.2 Busca, Tratamento e Quantificação dos Dados

Esta etapa engloba a busca pelas inserções e remoções, o tratamento das falhas que o Git pode ter registrado, a quantificação dos dados extraídos e a formalização e saída destes dados. Aqui, já estamos de posse dos dados dos *commits* encontrados na última etapa da extração e filtragem de *commits*. A partir desta etapa, a busca pelos dados deixa de ser feita através dos *Scripts Shell* e passamos a fazer uma análise textual através de casamento de padrões.

Os passos detalhados são apresentados a seguir e são executados de forma unitária para cada *commit* encontrado:

- **Busca por inserções e remoções:** Começamos contando os símbolos da API em cada *commit*. Relacionamos cada *commit* com seus arquivos registrados para sermos capazes de fazer uma contagem correta para cada arquivo do *commit*. Um *commit* apresenta uma estrutura de diferenças que identifica as adições e remoções com sinais de adição e subtração, respectivamente.

  Um exemplo de estrutura de diferenças de um *commit* pode ser visto na Figura 5.3. Percebe-se, neste exemplo, a presença de símbolos de adição e subtração, representando linhas alteradas pelo desenvolvedor.

```
ion.java
index b3b2263..0f86c3f 100644
--- a/src/main/java/net/spy/memcached/MemcachedConnection.java
+++ b/src/main/java/net/spy/memcached/MemcachedConnection.java
@@ -713,18 +713,15 @@ public class MemcachedConnection extends SpyThread {
     * @param node th enode to read write from.
     * @throws IOException if an error occurs during read/write.
     */
-  private void handleReadsAndWrites(final SelectionKey sk,
-    final MemcachedNode node) throws IOException {
-    if (sk.isValid()) {
-      if (sk.isReadable()) {
-        handleReads(node);
+  private void handleReadsAndWrites(final SelectionKey sk, final MemcachedNode node) throws IOException {
+      if (sk.isValid() && sk.isReadable()) {
+          handleReads(node);
       }
-      if (sk.isWritable()) {
-        handleWrites(node);
+
+      if (sk.isValid() && sk.isWritable()) {
+          handleWrites(node);
       }
-    }
   }
-
   /**
    * Finish the connect phase and potentially verify its liveness.
    *
```

Figura 5.3: Estrutura de diferenças armazenada por um *commit*.

Buscamos as linhas iniciadas com um único sinal de remoção ou adição. Então, agrupamos as informações do *commit* em dois grupos: (1) linhas adicionadas que possuem algum símbolo da API; (2) linhas removidas que possuem algum símbolo da API. Isso nos permite mensurar a frequência absoluta do uso da API. Porém, a forma como os *commits* são registrados ao longo do tempo pode

levar a algumas falhas ou até mesmo a situações onde estas adições e remoções não representem real uso da API.

- **Tratamento de falhas e redundâncias:** Para tratar redundâncias, identificamos que, algumas vezes, o controle de versão pode registrar a refatoração de código como alterações de adição e remoção de novos trechos. Nesses casos decidimos considerar o módulo da diferença entre adições e inserções de um mesmo método. Desta forma, a existências de duas linhas iguais em que uma tenha sido removida e adicionada não irão resultar em adição de expertise para o autor do *commit*. O uso das diferenças entre duas linhas permite levar em conta se a alteração envolve os parâmetros das chamadas, evitando eliminações incorretas.

- **Quantificação dos símbolos:** Quantificar os símbolos consiste em contar a quantidade de adições e remoções de cada símbolo. Com as linhas agrupadas e as redundâncias removidas, os símbolos de interesse são comparados com cada linha de adição e remoção para determinar cada quantidade. Neste cômputo, os símbolos não são discriminados pela quantidade de parâmetros ou outras características que possam diferenciar suas possíveis chamadas.

- **Extração dos dados quantificados:** A extração dos dados quantificados consiste na estruturação destes dados e apresentação dos resultados. Concluímos com a saída dos dados em dois formatos: um formato completo, apresentando os dados no formato citado na Seção 5.1, e uma versão resumida com os dados contidos na Tabela 5.1, que representam, para cada tupla, o uso de símbolos da API por um dados desenvolvedor para os projetos extraídos.

Tabela 5.1: Dados extraídos apresentados na forma resumida.

Dado extraído	**Descrição**
Desenvolvedor	Desenvolvedor responsável pelos commits
Métodos	Símbolos da API encontrados nos commits feitos pelo desenvolvedor
Quantidade inserida	Quantidade de símbolos distintos da API inseridos pelo desenvolvedor no conjunto de commits encontrados
Frequência de adições	Frequência de uso de símbolos da API inseridos pelo desenvolvedor no conjunto de commits encontrados
Quantidade de remoções	Quantidade de símbolos da API removidos pelo desenvolvedor no conjunto de commits encontrados
Frequência de remoções	Frequência de uso de símbolos da API removidos pelo desenvolvedor no conjunto de commits encontrados
Quantidade total	Quantidade total de símbolos da API usados pelo desenvolvedor no conjunto de commits encontrados
Frequência total	Frequência de uso de todos os símbolos da API pelo desenvolvedor no conjunto de commits encontrados

5.3 Definição do oráculo para validação

Para que possamos validar as métricas calculadas pela ferramenta, comparando os resultados por ela apresentados com a realidade, tivemos de definir um oráculo para validação.

Para chegar à definição de um oráculo, buscamos entender junto aos desenvolvedores do projeto Synx Move de uma empresa irlandesa quais as APIs de software que eram mais utilizadas no projeto. Duas APIs foram as mais citadas: JQuery e Google Maps.

Estas duas APIs foram selecionadas para validação da ferramenta no projeto em questão. O pass seguinte foi definir o oráculo dos desenvolvedores experientes nestas duas APIs junto ao projeto.

Para a definição do oráculo, nós realizamos uma survey com todo o time de desenvolvimento do projeto: desenvolvedores, gerente de projeto e testadores. O questionário aplicado nesta survey pode ser visto no Apêndice A.

O time de desenvolvimento é composto por 6 pessoas, porém as notas de expertise só foram computadas para os desenvolvedores de software, já que apenas estes podem

ser avaliados por meio da ferramenta, a partir dos dados extraídos do Git. Para cada membro do time, pedimos para eles quantificarem o conhecimento dos desenvolvedores do projeto em uma escala de zero a dez. Os desenvolvedores deste projeto são: Desenvolvedor_1, Desenvolvedor_2, Desenvolvedor_3 e Desenvolvedor_4. Cada resposta foi única e não teve nenhuma interferência dos outros desenvolvedores sobre cada uma das notas dadas. Estas respostas podem ser visualizadas nas Tabelas 5.2, sobre o JQuery, e 5.3, sobre o Google Maps.

Tabela 5.2: Respostas dos desenvolvedores para definição do oráculo na API JQuery.

Membro do Time	Dev_1	Dev_2	Dev_3	Dev_4
Dev_1	8	8	8	9
Dev_2	8	5	7	7
Dev_3	10	7	8	10
Dev_4	8	6	8	7
Dev_5	9	8	9	10
Dev_6	7	10	9	9

Tabela 5.3: Respostas dos desenvolvedores para definição do oráculo na API Google Maps.

Membro do Time	Dev_1	Dev_2	Dev_3	Dev_4
Dev_1	9	7	8	9
Dev_2	5	1	3	4
Dev_3	10	6	7	9
Dev_4	10	4	5	4
Dev_5	9	2	6	7
Dev_6	10	8	8	8

A partir das notas atribuídas por cada um dos desenvolvedores aos colegas, calculamos a média aritmética das notas para cada desenvolvedor para determinar a expertise de cada desenvolvedor. Desta forma, o oráculo definido para a API JQuery é o ranking apresentado na Tabela 5.4, e o oráculo definido para a API Google Maps pode ser visto na Tabela 5.5.

Tabela 5.4: Oráculo para a API do Jquery

Ranking	Desenvolvedor	Média das Respostas
1	Dev_4	8,6666
2	Dev_1	8,3333
3	Dev_3	8,1666
4	Dev_2	7,3333

Tabela 5.5: Oráculo para a API do Google Maps

Ranking	Desenvolvedor	Média das Respostas
1	Dev_1	8,8333
2	Dev_4	6,8333
3	Dev_3	6,1666
4	Dev_2	4,6666

5.4 Validação do Protótipo de Ferramenta

Com os oráculos definidos, nosso próximo passo foi verificar os dados das métricas calculadas pelo protótipo desenvolvido e verificar se a protótipo computou os resultados com boa acurácia.

Para a API do JQuery, os resultados de cada métrica são exibidos na Tabela 5.6. Com esses dados calculados, nós utilizamos a correlação de Spearman para analisar a intensidade da relação existente entre os oráculos e cada uma das métricas calculadas. Correlação é uma medida de relacionamento linear entre variáveis. Já o coeficiente de correlação de Spearman é uma estatística não paramétrica que pode ser usada para cálculo de correlação de amostras pequenas, onde não há uma grande variedade de dados a serem comparados.

Assim, para que possamos correlacionar as informações definidas pelo oráculo com as métricas extraídas pelo protótipo, utilizamos esta correlação, que pode ser visualizada na Tabela 5.8.

Para a API do Google Maps, pudemos observar os resultados das métricas extraídas pelo protótipo através da tabela 5.7. Da mesma forma que foi verificada a correlação de Spearman para a API do JQuery, calculamos esta medida para a API do Google Maps. Os resultados para a correlação de Spearman do oráculo com as métricas utilizadas podem ser observadas na Tabela 5.9.

Tabela 5.6: Resultados do Protótipo para a API do Jquery

Métricas	Dev_1	Dev_2	Dev_3	Dev_4
Depth of Method Knowledge	8712	4229	14347	8904
Breadth of Method Knowledge	71	32	65	69
Relative Depth of Method Knowledge	21.04	5.86	27.07	19.52
Relative Breadth of Method Knowledge	22.50	7.09	17.33	20.50
Library Expertise	0.31	0.14	0.28	0.30
Expertise Distance	0.69	0.86	0.71	0.69

Tabela 5.7: Resultados do Protótipo para a API do Google Maps

Métricas	Dev_1	Dev_2	Dev_3	Dev_4
Depth of Method Knowledge	123	25	131	145
Breadth of Method Knowledge	5	2	5	5
Relative Depth of Method Knowledge	1.37	0.43	1.46	1.31
Relative Breadth of Method Knowledge	1.28	0.5	1.28	1.28
Library Expertise	0.05	0.02	0.05	0.05
Expertise Distance	0.95	0.98	0.95	0.95

Tabela 5.8: Correlação Spearman para a API do Jquery

Métrica	Correlação de Spearman (Oráculo)	Valor-p
Depth of Method Knowledge	0.400	0.600
Breadth of Method Knowledge	0.800	0.200
Relative Depth of Method Knowledge	0.200	0.800
Relative Breadth of Method Knowledge	0.800	0.200
Library Expertise	0.800	0.200
Expertise Distance	0.200	0.800

Tabela 5.9: Correlação Spearman para a API do Google Maps

Métrica	Correlação de Spearman (Oráculo)	Valor-p
Depth of Method Knowledge	0.400	0.600
Breadth of Method Knowledge	0.775	0.225
Relative Depth of Method Knowledge	0.400	0.600
Relative Breadth of Method Knowledge	0.775	0.225
Library Expertise	0.775	0.225
Expertise Distance	0.775	0.225

Em estatística, resultados da correlação de Spearman acima de 0.5 são considerados de correlação alta. Entre 0.3 e 0.5 são considerados de correlação média, e os resultados entre 0.1 e 0.3 são considerados de baixa correlação.

Para os resultados da correlação de Spearman para a API do JQuery, podemos observar que a maioria das métricas apresentam uma correlação alta, apenas uma apresenta média correlação e duas apresentam baixa correlação comparadas ao oráculo. Já os resultados da correlação para a API do Google Maps apresentam quatro métricas com correlações altas e duas com correlações médias.

Capítulo 6

Avaliação das Métricas para Identificação de Especialistas

Neste capítulo, é apresentado o processo de avaliação das métricas para a identificação de especialistas. Esse processo avaliativo foi realizado usando o protótipo da ferramenta desenvolvida. Para tanto, foram analisadas APIs de software conhecidas, utilizadas em projetos de software aberto disponíveis no GitHub.

Dividimos o processo de avaliação das métricas em cinco subseções, explicando desde a escolha das APIs e projetos, até o processo de avaliação das métricas, seus resultados e análise.

6.1 APIs Avaliadas

As APIs definidas foram escolhidas com base em uma pesquisa que aponta as 100 APIs mais usadas da linguagem Java[1]. Foram analisadas algumas das APIs apontadas e três delas foram escolhidas por sua popularidade e pela facilidade de extração de seus métodos através de sua documentação em Javadoc.

Para que possamos avaliar as métricas de identificação de especialistas, nós utilizamos três APIs como referência:

1. Apache Commons IO;
2. Google Guava;
3. Log4j.

A Tabela 6.1 descreve alguns detalhes das APIs escolhidas, tais como a versão delas utilizada, a quantidade de métodos e uma breve descrição.

[1]https://blog.takipi.com/the-top-100-java-libraries-in-2016-after-analyzing-47251-dependencies/

Tabela 6.1: APIs Utilizadas

API	Versão	Número de métodos	Descrição da API
Apache Commons IO	2.4	325	Projeto Apache focado em todos os aspectos dos componentes Java reutilizáveis. O Commons IO é uma API de utilitários para auxiliar no desenvolvimento da funcionalidade de IO.
Google Guava	19.0	1341	Conjunto de bibliotecas comuns para Java, desenvolvido principalmente pelos engenheiros do Google.
Log4j	2.11.0	80	Utilitário de logging baseado em Java

6.2 Seleção dos Projetos

Após a definição das APIs, foi utilizada a pesquisa avançada do GitHub para selecionar projetos *open source* que as utilizam.

Para a seleção desses projetos, foram utilizados os seguintes critérios:

1. Projetos que contenham *strings* de importação da API para uso, e.g., "import org.apache.commons". Assim, foram filtrados os projetos que as utilizam.
2. Linguagem dos projetos. Para essa avaliação, nós utilizamos apenas projetos em Java, embora a ferramenta possa ser parametrizada para qualquer linguagem.
3. Quantidade de desenvolvedores ativos. Foram considerados para a análise projetos que tenham mais de 10 desenvolvedores contribuindo ativamente com o projeto.
4. Número de commits. Considerou-se apenas projetos com mais de mil *commits*.
5. Existência de commits recentes, i.e., commits realizados a partir do ano de 2017. Isso porque toda a análise foi feita a partir de commits realizados no período entre Janeiro de 2017 e Março de 2018.

Para as APIs Apache Commons e Google Guava, foram utilizados três projetos extraídos do GitHub. Para a API Log4j, foram utilizados quatro projetos. A Tabela 6.3 ilustra os projetos utilizados.

Tabela 6.2: Projetos Utilizados para cada API

API	Projeto	Descrição do Projeto	Tamanho do Projeto	Quantidade de Contribuidores	Número de Commits
Apache Commons	xwiki-platform	Plataforma wiki genérica que oferece serviços de tempo de execução para aplicativos construídos sobre ela.	47,7MB	85	34.342
	pentaho-kettle	Ferramenta de análise de negócios (BI) porém esse projeto é voltado a integração com o Kettle.	27,3MB	157	20.336
	dxm	Projeto de envolvendo marketing digital	38,4MB	36	26.994
Google Guava	durid	Projeto de armazenamento de dados analíticos distribuído, orientado por coluna e em tempo real, que é comumente usado para fornecer painéis exploratórios em ambientes com vários usuários.	17,5MB	159	7.786
	presto	Projeto que desenvolve um mecanismo de consulta SQL distribuída para big data.	16,4MB	230	13.985
	sonarJava	Analisador de código para projetos Java.	6.9MB	67	5.671
Log4j	grouper	Sistema de gerenciamento de acesso corporativo projetado para o ambiente de gerenciamento altamente distribuído e o ambiente de tecnologia de informação heterogêneo comum às universidades.	316MB	18	8.576
	hops	Distribuição da próxima geração do Apache Hadoop com metadados escaláveis, altamente disponíveis e personalizáveis.	41,2MB	54	9.071
	OWLTools	Conjunto de APIs de Java que roda na API do OWL.	13,8MB	12	2.574
	cosmo	Java Async SDK para APIs SQL do Azure Cosmos DB	689KB	10	1182

Tabela 6.3: Projetos Utilizados para cada API

API	Projeto	Tamanho do Projeto	Quantidade de Contribuidores	Número de Commits
Apache Commons	xwiki-platform	47,7MB	85	34.342
	pentaho-kettle	27,3MB	157	20.336
	dxm	38,4MB	36	26.994
Google Guava	durid	17,5MB	159	7.786
	presto	16,4MB	230	13.985
	sonarJava	6.9MB	67	5.671
Log4j	grouper	316MB	18	8.576
	hops	41,2MB	54	9.071
	OWLTools	13,8MB	12	2.574
	cosmo	689KB	10	1182

6.3 Design Experimental

Com as APIs e projetos definidos, o objetivo principal é entender o comportamento das métricas em alguns contextos e como elas se relacionam entre si. A partir desta perspectiva, alguns questionamentos surgiram a fim de entender e como avaliar cada uma delas. Esses questionamentos são:

1. Como estas métricas se relacionam entre si? Os rankings produzidos pelas métricas são diferentes ou iguais ao analisarmos esses projetos usando métricas diferentes?
2. Como as métricas se comparam com um oráculo produzido a partir de commits reais realizados após o período usado para construção dos rankings das métricas? As métricas são assertivas na identificação de especialistas se considerarmos APIs de software em vários projetos?
3. Como identificar especialistas em APIs de software utilizando este conjunto de métricas?

Buscando responder a essas perguntas, definimos algumas estratégias para entender e avaliar as métricas para a identificação de especialistas, a saber:

1. Perguntar aos próprios desenvolvedores dos projetos se os desenvolvedores identificados pela ferramenta correspondem a especialistas reais;
2. Correlacionar estatisticamente as métricas em si e analisar o seu comportamento;
3. Correlacionar estatisticamente as métricas com um oráculo e analisar os resultados;
4. Verificar estatisticamente a *recall* médio de acerto das métricas em relação a um oráculo;

5. Analisar o resultado das diferentes estatísticas e avaliar o rankings de especialistas produzidos com o uso das métricas.

As estratégias acima serão detalhados e discutidos nas subseções a seguir.

6.3.1 Perguntando aos Desenvolvedores

Para avaliar os resultados apontados pelas métricas de identificação de especialistas em APIs de software, a nossa primeira estratégia foi perguntar a alguns desenvolvedores de cada projeto quem são os especialistas na API com base nos projetos em que eles estavam envolvidos.

Para tanto, selecionamos os cinco primeiros desenvolvedores que apareceram na listas de recomendação das métricas utilizadas para cada API. A partir deles, foi elaborado um questionário com cinco perguntas, onde cada pergunta permitia que o respondente avaliasse cada desenvolvedor. Cada respondente deveria dar uma nota de zero a dez para o expertise do desenvolvedor em questão. O modelo desse questionário enviado aos desenvolvedores pode ser visualizado no Apêndice B desse documento.

Esses questionários foram enviados via e-mail para os desenvolvedores dos projetos selecionados. Durante um período de 30 dias e duas tentativas de envio, não obtivemos nenhuma resposta por parte deles. Diante disso, esta estratégia foi desconsiderada para a avaliação por falta de dados.

6.3.2 Correlação de Spearman para a avaliação das métricas

Como estratégia de avaliação das métricas para melhor entendimento do processo de recomendação de especialistas foi utilizada a correlação de Spearman. A correlação de Spearman é uma estatística não-paramétrica, que pode ser usada quando os dados não respeitarem uma distribuição normal. Dado que r denote esta correlação, $-1 <= r <= 1$. Na sua interpretação, quando mais próximo de $+1$ r estiver, mais forte é a correlação entre as variáveis. Quanto mais próximo de zero, mais fraca é a correlação. Quanto mais próximo de -1, a correlação é mais forte, mas no sentido inverso. A correlação de Spearman é útil em nosso caso, pois é baseada em uma comparação dos postos dos elementos em duas listas diferentes. No contexto de identificação e priorização de especialistas, interessa saber não apenas se o especialista está presente na lista, mas também em que ordem, i.e., posto, este aparece na lista.

Usamos a correlação de Spearman neste contexto de avaliação das métricas com dois objetivos: 1) verificar se as métricas se aproximam entre si, ou seja, se elas se aproximam de uma mesma dimensão de avaliação; 2) o quanto as métricas se correlacionam com um oráculo determinado pelos commits dos desenvolvedores em um período posterior ao período no qual as métricas foram computadas. Em um cenário

onde vários autores propõem diferentes métricas para identificação de especialistas, estes dois objetivos permitem compreender melhor as semelhanças e diferenças entre os resultados apontados pelas métricas.

Uma primeira etapa de um ano de commits (no período de 01/01/2017 a 31/12/2017) permite computar as métricas a partir do protótipo da ferramenta. Para cada API e para o conjunto dos projetos selecionados que usam aquela API, foram extraídas as métricas para cada desenvolvedor, e foi gerada uma lista de especialistas ordenada de acordo com o valor da métrica.

Em seguida, usamos um período adicional de três meses de commits (de 01/01/2018 a 30/03/2018) para a geração de um oráculo de desenvolvedores que realizaram commits nos projetos e utilizaram a API em questão. Este oráculo é uma lista de desenvolvedores, ordenada do desenvolvedor que mais usou a API para o desenvolver com o menor.

A partir das listas de especialistas para cada métrica e da lista de especialistas do oráculo, foi calculada a correlação de Spearman. Primeiro, a correlação entre as métricas, para determinar se tratam de uma mesma dimensão. Depois, a correlação da lista de especialistas dada por cada métrica com a lista de especialistas do oráculo.

Os resultados serão apresentados na seção a seguir.

6.3.3 *Recall* médio para a avaliação das métricas

Como uma outra estratégia de avaliação das métricas para identificação de especialistas, nós utilizamos o calculo do *recall* médio em relação a um conjunto de *commits* imediatamente posteriores ao período usado para o cômputo das métricas. Essa estratégia foi utilizada e baseada no artigo de Ma. et. al. [Ma et al. 2009], onde o mesmo usou o processo de validação por meio de *recall* médio para analisar o comportamento das métricas em um período posterior à análise.

Uma primeira etapa de um ano de commits (no período de 01/01/2017 a 31/12/2017) permite computar as métricas a partir do protótipo da ferramenta. Para cada API e para o conjunto dos projetos selecionados que usam aquela API, foram extraídas as métricas para cada desenvolvedor, e foi gerada uma lista de especialistas ordenada de acordo com o valor da métrica. O tamanho das listas foi variado de uma lista com apenas o desenvolvedor com o maior valor da métrica (Top-1) até uma lista com os 10 desenvolvedores com os maiores valores das métricas (Top-10).

Em seguida, usamos um período adicional de três meses de commits (de 01/01/2018 a 30/03/2018) para a geração de um oráculo conforme descrito a seguir.

Primeiramente, tomamos o primeiro commit deste período e verificamos se o desenvolvedor usou a API em questão neste commit. Caso positivo, verificamos se este desenvolvedor aparece em cada lista Top-K (K variando de 1 a 10) de cada métrica.

Se positivo, consideramos que a lista produzida por aquela métrica foi precisa e obteve um acerto. Caso negativo, consideramos que a métrica não foi precisa e gerou um erro.

Em seguida, adicionamos o commit analisado ao conjunto de treinamento e recomputamos as métricas para desenvolvedor e as listas Top-K para cada métrica. Posteriormente, tomamos o segundo commit do período adicional e verificamos, de modo similar, se o desenvolvedor usou a API em questão neste commit e, se positivo, se ele aparece em cada lista Top-K de cada métrica, computando o acerto ou erro para este caso.

Repetimos todo o processo para o conjunto de todos os commits seguintes do período adicional de três meses, computado o acerto ou erro para cada lista Top-K de cada métrica. Finalmente, computamos o *recall* médio de acerto de cada métrica, ao variarmos o valor de K entre 1 e 10.

Os resultados do *recall* médio serão apresentados na seção a seguir.

6.4 Resultados

Nesta seção, apresentamos os resultados das duas avaliações realizadas, primeiro pela correlação de Spearman, depois pelo *recall* médio.

Vale ressaltar, no entanto, que a métrica *expertise distance* não foi considerada por termos avaliado APIs individualmente, e não um conjunto de APIs.

6.4.1 Correlação de Spearman

Correlação entre as métricas

Como avaliação dos resultados da correlação de Spearman entre as métricas, apresentamos esta correlação para cada uma das APIs utilizadas: Apache Commons, Google Guava e Log4j.

Na Tabela 6.4, é possível observar a correlação de Spearman entre as métricas para a API do Apache Commons. É possível verificar que elas possuem uma correlação alta, pois todas as métricas apresentam resultados próximo de +1.

Tabela 6.4: Correlação de Spearman entre as métricas na API do Apache Commons

Métrica	Relative Depth	Depth	Relative Breadth	Breadth	Library Expertise
Relative Depth	1.0000	0.9079	0.9662	0.9171	0.9111
Depth	0.9079	1.0000	0.9137	0.9834	0.9834
Relative Breadth	0.9662	0.9137	1.0000	0.9295	0.9295
Breadth	0.9171	0.9834	0.9295	1.0000	1.0000
Library Expertise	0.9111	0.9834	0.9295	1.0000	1.0000

Para a API do Google Guava, as mesmas análises foram realizadas. É possível observar na Tabela 6.5 que o resultado da correlação entre as métricas para o Google Guava também apresentam correlação alta.

Tabela 6.5: Correlação de Spearman entre as métricas na API do Google Guava

Métrica	Relative Depth	Depth	Relative Breadth	Breadth	Library Expertise
Relative Depth	1.0000	0.9159	0.9889	0.9383	0.9383
Depth	0.9159	1.0000	0.9261	0.9687	0.9687
Relative Breadth	0.9889	0.9261	1.0000	0.9634	0.9634
Breadth	0.9383	0.9687	0.9634	1.0000	1.0000
Library Expertise	0.9383	0.9687	0.9634	1.0000	1.0000

Para a API do Log4j, podemos observar que o resultado da correlação entre as métricas não é tão alto como nas APIs anteriores. De acordo com a tabela 6.6, podemos visualizar que há correlações altas entre algumas das métricas (por exemplo entre a métrica *Depth* e *Breadth*), correlações médias (por exemplo entre a métrica *Relative Depth* e *Breadth*) e correlações mais baixas (por exemplo entre a métrica *Depth* e *Library Expertise*).

Tabela 6.6: Correlação de Spearman entre as métricas na API do Log4j

Métrica	Relative Depth	Depth	Relative Breadth	Breadth	Library Expertise
Relative Depth	1.0000	0.5952	0.8095	0.6827	0.7600
Depth	0.5952	1.0000	0.8571	0.9403	0.4251
Relative Breadth	0.8095	0.8571	1.0000	0.9403	0.7085
Breadth	0.6827	0.9403	0.9403	1.0000	0.5819
Library Expertise	0.7600	0.4251	0.7085	0.5819	1.0000

Correlação das métricas com o oráculo

Na Tabela 6.7, é possível observar o valor da correlação de cada uma das métricas com o oráculo. Para a maioria dos resultados, esta correlação é maior que 0.5, considerada alta.

Para a API do Apache Commons, os resultados da correlação apresentam valores de correlação alta, acima de 0.5.

Considerando a correlação das métricas com o oráculo para o Google Guava, é possível observar que, para esta API, a correlação das métricas com o oráculo é considerada muito alta, acima de 0.7.

Já na correlação com o oráculo para a API do log4j, é possível observar diferenças na correlação entre o oráculo e as métricas. É considerada uma correlação muito alta aquelas acima de 0.7, correlações alta acima de 0.5, média acima de 0.3, baixa acima de 0.1 e sem correlação entre 0 e 0.1. Diante disso, é possível observar que algumas delas apresentam uma correlação muito alta, como por exemplo a *depth, breadth e relative breadth*, outra uma correlação alta, como *relative depth*, e uma última apresenta correlação média, caso de *library expertise*. Porém, podemos observar que há uma predominância de correlação alta.

Tabela 6.7: Correlação de Spearman das métricas com o oráculo

Métrica	Apache Commons	Google Guava	Log4j
Relative Depth	0.5866	0.8846	0.5952
Depth	0.5478	0.9759	1.0000
Relative Breadth	0.5611	0.8924	0.8571
Breadth	0.5856	0.9376	0.9403
Library Expertise	0.5856	0.9376	0.4251

6.4.2 *Recall* médio

Nesta seção, apresentamos os resultados do calculo do *recall* médio conforme o design experimental. Os resultados são apresentados na forma de gráficos de *recall* médio de cada métrica, onde o eixo horizontal permite variar o tamanho da lista Top-K entre 1 e 10.

A Figura 6.1 apresenta os resultados da API do Apache Commons para as métricas *relative depth*, *depth*, *relative breadth*, *breadth* and *library expertise*. Percebe-se que, quanto maior o conjunto de *commiters*, maior é o *recall* médio (taxa média de acerto). Considerando um conjunto Top-4, por exemplo, a média aproximada de acerto para as métricas está em torno de 40% para a API do Apache Commons, sendo a métrica *depth* a que apresenta o melhor resultado. A partir do Top-7, é

possível observar que a taxa de acerto começa a atingir 50% para a métrica *depth*. As demais métricas apresentam um resultado bem próximo, entre 40% e 50%.

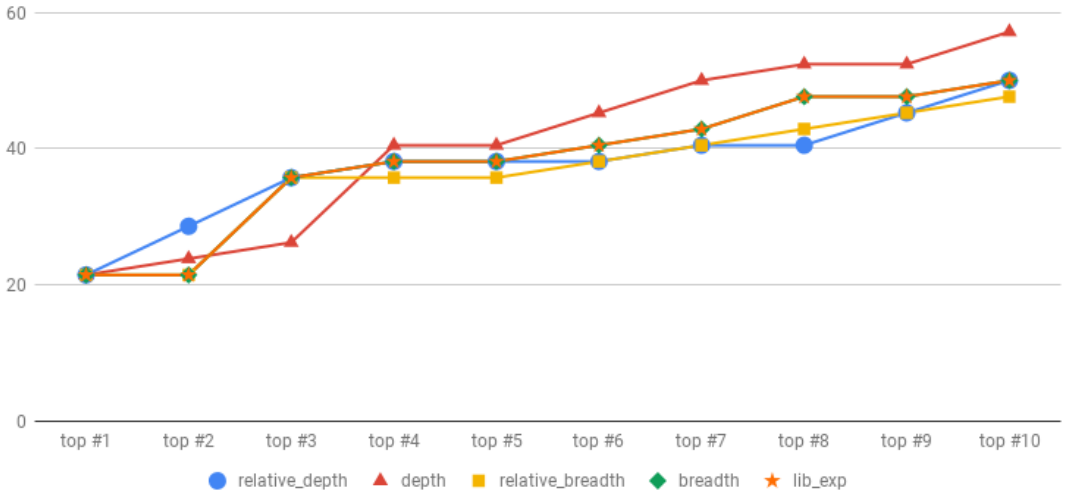

Figura 6.1: *Recall* médio para a API do Apache Commons

Na Figura 6.2, podemos observar que o comportamento das métricas para a API do Google Guava é bem similar ao apresentado para a API do Apache Commons. Em um cenário onde são apresentados um conjunto de dados similares e períodos iguais, é possível observar que o comportamento do gráfico é bem parecido. Até o Top-3, por exemplo, o *recall* médio está em torno de 20% para todas as métricas. A partir do Top-7, é possível observar que a métrica *depth* apresenta os melhores resultados a nível de *recall* médio, em torno de 40%, variando entre 38% e 43%.

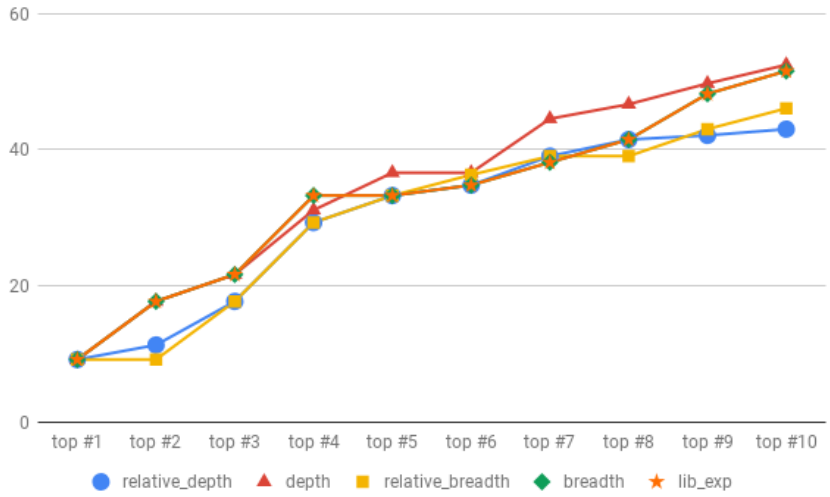

Figura 6.2: *Recall* médio para a API do Google Guava

A Figura 6.3 apresenta os resultados das métricas para a API do Log4j. Para esta API, é possível observar que os resultados apresentam uma diferença em relação aos resultados para as duas APIs anteriores. A partir do TOP-1 é possível observar que o percentual de acerto é superior a 50% para todas as métricas. A partir do TOP-7, todas as métricas já apresentam um percentual de acerto superior a 85%. Os TOP-9 e Top-10 apresentam 100% de acerto para todas as métricas.

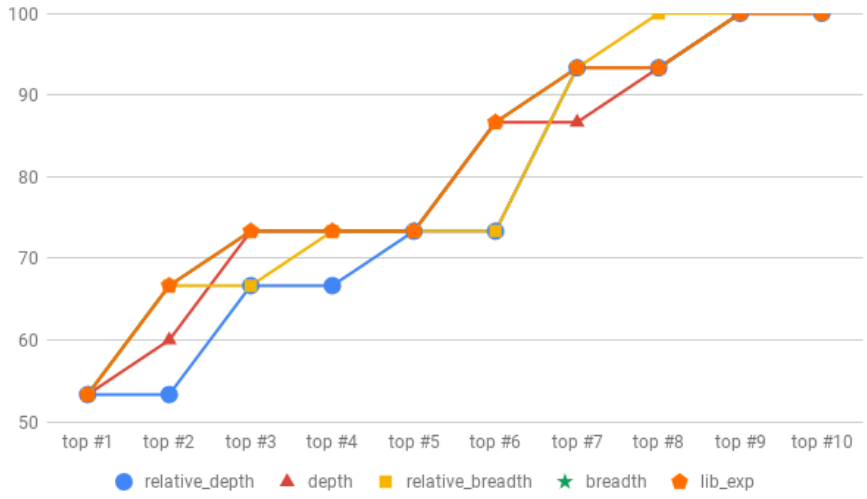

Figura 6.3: *Recall* médio para a API do Log4j

6.5 Análise

Nesta seção, analisamos os resultados e tecemos uma discussão sobre seus desdobramentos. Nesta análise, os questionamentos realizados na sessão do design experimental são respondidas e discutidas.

O primeiro questionamento realizado foi: *Como estas métricas se relacionam entre si? Os rankings produzidos pelas métricas são diferentes ou iguais ao analisarmos esses projetos usando métricas diferentes?*. Este questionamento foi realizado buscando entender o comportamento das métricas. Diante de um contexto onde existem vários autores propondo métricas para recomendação de especialistas, estas métricas possuem uma correlação alta entre si, ou seja, seus respectivos rankings de recomendação são próximos. Toda esta análise foi realizada utilizando a correlação de Spearman.

Diante dos resultados apresentados da correlação de Spearman, é possível observar que, em um cenário de grupos médios ou grandes de desenvolvedores (projetos que usam Google Guava ou Apache Commons), esta correlação é alta para todas as métricas. Se considerarmos um grupo pequeno de desenvolvedores (projetos que

usam log4j), esta correlação é um pouco menor se compararmos as métricas, porém ainda é considerada alta. Diante disso, deve-se ter mais cuidados ao propor mais uma métrica para recomendação de especialistas já que, para a maioria das métricas atuais, seus resultados são bastante similares. As métricas são muito próximas e apontam para uma mesma tendência. Antes da proposição de novas métricas, é necessário entender em que aspecto ela deve se diferenciar das outras, pois o esforço para definição de métricas com tão alto grau de correlação com as outras parece não capturar novos atributos relevantes.

O segundo questionamento realizado foi: *Como as métricas se comparam com um oráculo produzido a partir de commits reais realizados após o período usado para construção dos rankings das métricas? As métricas são assertivas na identificação de especialistas se considerarmos APIs de software em vários projetos?* Para buscar responder a essa pergunta, nós utilizamos a correlação de Spearman e o calculo do *recall* médio.

Na análise realizada através dos resultados apontados pela correlação de Spearman, nós podemos observar dois aspectos: (1) independente da métrica utilizada, já que percebemos na análise anterior que elas apresentam uma correlação alta entre si, a correlação das métricas com o oráculo é considerada de média para muito alta na maioria das APIs; (2) Se analisarmos em termos de assertividade, podemos observar que, para um conjunto de médio a grande de desenvolvedores (que é o caso dos projetos que usam Apache Commons e Google Guava), a assertividade é considerada alta. Já para a análise com grupos pequenos de desenvolvedores, que é o caso do Log4j, essa identificação de especialistas sofre variações a depender da métrica utilizada.

Já com os resultados apontados pela análise do *recall* médio, é possível observar que, para um conjunto pequeno de desenvolvedores, o percentual de acerto é maior do que para um conjunto grande. Se compararmos as Figuras 6.1, 6.2 e 6.3 podemos perceber que, a partir do Top-1, todas as métricas acertam mais de 50% para a API do log4j. Já se analisarmos a API do Apache Commons, uma das métricas atinge 50% apenas no Top-7, e, para a API do Google Guava, apenas no Top-9. Ou seja, considerando o *recall* médio, é possível observar que, para um conjunto pequeno de desenvolvedores, que é o caso da API do Log4j, as métricas apresentam melhores resultados se comparados a um conjunto médio a grande de desenvolvedores, que é o caso da API do Apache Commons e Google Guava, respectivamente.

Porém, diante dessas análises realizadas com os resultados da correlação de Spearman e *recall* médio, há outras possíveis explicações para essa assertividade. Algumas hipóteses levantadas para esse comportamento são:

1. *a complexidade da API.* Nas análises realizadas, nós utilizamos APIs de complexidade mediana. São APIs de relativamente fácil utilização onde, através da sua documentação, é possível entender como elas podem ser utilizadas. A API do Apache Commons é uma API de uso geral, assim como a API do Google Guava. Já a API do Log4j é uma API de geração de logs.

2. *a quantidade de métodos da API utilizados por cada desenvolvedor.* Foi possível perceber que a quantidade de métodos da API utilizados pelos desenvolvedores no período analisado não se aproxima de sua totalidade. Do universo de 100% dos métodos da API, para a API do Apache Commons, apenas 22% dos métodos da API foram usados durante esse período. Para a API do Google Guava 37% dos métodos e para a API do Log4j, 33% dos métodos foram utilizados pelos desenvolvedores.

O terceiro questionamento realizado foi: *Como identificar especialistas em APIs de software utilizando este conjunto de métricas?*. Diante dos resultados apresentados, é possível perceber que o processo de identificação de APIs de software pode ser realizado utilizando qualquer uma das métricas analisadas durante esta pesquisa, pois o resultado final delas não varia muito de acordo com a correlação de Spearman. Porém, o que podemos perceber também, diante dos resultados apresentados, é que o tamanho do conjunto de desenvolvedores e a complexidade da API podem influenciar nos resultados.

Se compararmos nossos resultados com o apresentado por Ma et. al. (2009), podemos observar que eles apresentaram resultados muito mais significativos em termos de *recall* médio, embora eles não tenham avaliado em seu trabalho a identificação de especialistas em APIs, mas de especialistas em arquivos/classes de um projeto de software.

Considerando as APIs de software analisadas neste trabalho nos projetos que usam esta API, os nossos resultados de *recall* médio (considerando as três APIs) chegam, em média, a 49% de acerto no Top-5. O valor mínimo de *recall* médio para o Top-5 é de 36.73% e valor máximo de 73.33%. Podemos observar através da Tabela 6.8 o valor médio, o desvio padrão, o valor mínimo e o valor máximo das três APIs para os rankings Top-1, Top-5 e Top-10.

Tabela 6.8: Valores de *recall* médio para os Top-1, Top-5 e Top-10

	Top-1	Top-5	Top-10
Valor Médio	27.5898	49.1999	69.6535
Desvio Padrão	20.2898	37.6811	56.5217
Valor Mínimo	9.1463	36.5853	52.4390
Valor Máximo	53.3333	73.3333	100.0000

É possível observar também que, se compararmos os resultados apresentados pelo cálculo da correlação de Spearman com os resultados do *recall* médio, podemos observar que esses resultados apresentam uma coerência. A média de *recall* médio para as três APIs fica em torno de 48% a partir do Top-5 (variando entre 38 e 73% entre as APIs). Considerando a correlação de Spearman, seu valor é geralmente alto, em média, em torno de $0,5$ para a API do Apache Commons, $0,9$ para o Google Guava e $0,7$ para o Log4j. Em ambos os casos, tanto para o *recall* médio,

como para a correlação de Spearman, nosso protótipo de ferramenta gera resultados potencialmente úteis para cenários de identificação de especialistas. No caso do *recall* médio, procura-se identificar ao menos um especialista numa lista dada pela métrica, o que dá resultados razoáveis a partir de listas Top-5. Na correlação de Spearman, procura-se ordenar um conjunto de especialistas, do mais *expert* para o menos *expert*. Neste caso, as correlações altas apontam para a qualidade das métricas para a determinação destes rankings.

Uma outra análise que pode ser feita nessa seção é em relação a comparação de uma das métricas para o conjunto das APIs analisadas. Se compararmos, por exemplo, a métrica mais bem sucedida das três APIs utilizadas (a que obteve as correlações mais altas em dois das três APIs), a *depth*, podemos observar através da figura 6.4 que elas apresentam resultados próximos para as APIs do Apache Commons e Google Guava. Para a API do Log4j, é possível observar que ela apresenta um *recall* médio superior já no Top-1 pelos mesmos motivos já discutidos anteriormente.

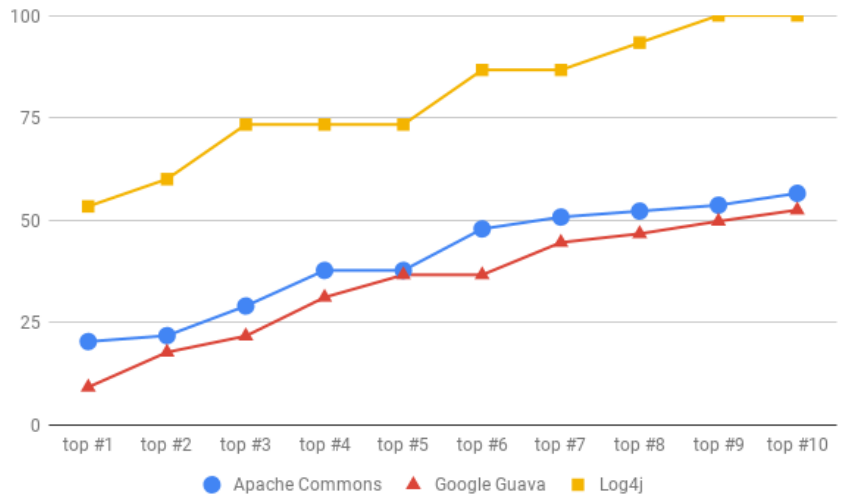

Figura 6.4: *Recall* médio para a métrica Depth em todas as APIs

6.5.1 Ameaças à Validade

Diante dos resultados discutidos, vale ressaltar que ainda são necessários alguns avanços para a melhoria dos resultados do *recall* médio. Todo a pesquisa foi realizada utilizando apenas três APIs de complexidade baixa ou média, envolvendo três ou quatro projetos para cada uma delas. Tal limitação pode ser justificada pela finalidade deste trabalho de entender a viabilidade do protótipo e fazer uma primeira análise de projetos em cenários reais.

Outra ameaça à validade são os oráculos utilizados. É sabido que os oráculos apresentados são limitados por representar uma "verdade" dada pelo desenvolvimento

posterior a um dado período, o que pode ser influenciado por outros fatores como a própria disponibilidade dos desenvolvedores de projetos open source e mudanças na dinâmica do projeto. Porém, devido a dificuldade de resposta por parte dos desenvolvedores, optamos por validar o protótipo da ferramenta por meio destes dois oráculos. Por outro lado, não nos distanciamos muito da literatura, já que o oráculo utilizado para a análise da *recall* médio, por exemplo, foi baseado no trabalho de Ma. et. al. [Ma et al. 2009].

Capítulo 7

Conclusões

Neste projeto, fizemos um estudo exploratório a fim de avaliar um conjunto de métricas para a identificação de especialistas em APIs de software a partir de conhecimentos existentes em repositórios sociais de software. Para isso, buscamos identificar as principais métricas utilizadas pelos autores dos trabalhos relacionados e avaliamos o uso destas métricas para identificar especialistas em APIs de software.

Diante da diversidade de métricas mapeadas para determinação de expertise em APIs, concluímos que elas são bastante variadas, usam diversos tipos de entradas, mas que não pode se afirmar um consenso sobre a melhor métrica para as diversas situações de determinação de expertise.

Para que essas métricas fossem computadas e avaliadas em um cenário real no contexto de uso de APIs de software, foi desenvolvido um protótipo de uma ferramenta. Este protótipo é capaz de ler n projetos para uma determinada API de software e computar as métricas selecionadas, produzindo rankings de desenvolvedores que dominam as APIs, ordenados pelas valores das métricas em questão.

Com o resultado computado pelo protótipo da ferramenta para as métricas, foi possível avaliá-las e entender o seu comportamento na identificação de especialistas em APIs de software. Para a avaliação, procuramos identificar especialistas a partir dos rankings produzidos para cada métrica a partir do protótipo de ferramenta. Ordenamos os conjuntos de especialistas produzidos e comparamos estes rankings. Através desta estratégia, percebemos que as métricas selecionadas apresentam uma forte correlação entre si. Ou seja, estas métricas apresentam um ranking de identificação de especialistas relativamente próximo, apontando para uma mesma tendência. Por outro lado, ao analisar as métricas em relação a uma *ground truth* baseada no desenvolvimento do software posterior ao cômputo das métricas, percebemos que, em um cenário com um grupo pequeno de desenvolvedores que utilizam a API e com menor complexidade da API, as métricas apresentam uma melhor precisão. Para grupos maiores de desenvolvedores e com uma complexidade maior da API, os resultados são menos precisos, porém, ainda assim, apresentam uma precisão média,

considerando as três APIs avaliadas, de 48% a partir de rankings Top-5, ou seja, rankings com cinco desenvolvedores.

Diante do exposto, percebe-se que o processo de identificação de especialistas em APIs de software é viável a partir da utilização do protótipo da ferramenta desenvolvida. Como todas as métricas selecionadas apontam para uma mesma tendência, qualquer uma delas pode ser utilizada para a identificação de especialistas em APIs. Os resultados apresentados são passíveis de melhoria, porém apontam para a viabilidade de identificar especialistas em APIs a partir de métricas de expertise.

Embora esta área demande ainda uma exploração mais sistemática para avaliação do potencial de repositórios sociais de software como determinantes na identificação de especialistas em APIs de software, considerando todo o trabalho realizado aqui, é possível afirmar que os resultados obtidos sugerem a viabilidade destas fontes de dados como primeiro passo na direção de uma solução, automática ou semiautomática, para identificação de especialistas em APIs de software.

7.1 Limitações do Trabalho

Algumas limitações foram encontradas para a realização dessa pesquisa e serão elencadas nessa seção.

Uma primeira limitação encontrada foi uma limitação técnica para a seleção das APIs. APIs com um grande número de métodos e mais complexas levam a um tempo muito grande de análise para a geração das métricas no protótipo da ferramenta. Diante disso, optou-se, para a avaliação, por APIs menos complexas.

Outra limitação encontrada foi a abrangência da avaliação. Para cada uma das APIs, foram utilizadas apenas três ou quatro projetos extraídos do GitHub. O GitHub possui um conjunto de projetos bastante abrangente, mas, pelo tempo disponível para a avaliação, não foi viável avaliar toda a sua base de dados.

7.2 Trabalhos Futuros

A avaliação descrita nesta dissertação visou identificar especialistas em APIs de software a partir de repositórios sociais de código fonte. Com isso, consideramos relevante a realização de trabalhos adicionais a partir dos resultados deste trabalho. Dentre eles, elencamos:

1. Agregar nesta avaliação um conjunto mais amplo de projetos de software disponíveis nos repositórios sociais de software como o GitHub;
2. Realizar avaliação similar com APIs mais complexas, onde a identificação de especialistas se faz ainda mais necessária;

3. Realizar uma avaliação adicional a partir da visão dos próprios desenvolvedores dos projetos sobre os especialistas nas APIs identificados pelo protótipo da ferramenta;
4. Propor novas métricas a partir do aprendizado deste trabalho;
5. Analisar outras fontes além do código-fonte para auxiliar na identificação de especialistas em APIS como, por exemplo, redes sociais de profissionais como o LinkedIn.

Referências Bibliográficas

[Alonso et al. 2008] Alonso, O., Devanbu, P. T., e Gertz, M. (2008). Expertise identification and visualization from cvs. In *Proceedings of the 2008 international working conference on Mining software repositories*, pp. 125–128. ACM.

[da Silva et al. 2012] da Silva, J. T., Gerosa, M. A., Wiese, I., Ré, R., e Steinmacher, I. (2012). An extensible service for experts recommendation on distributed software development projects. In *Global Software Engineering Workshops (ICGSEW), 2012 IEEE Seventh International Conference on*, pp. 18–21. IEEE.

[Dabbish et al. 2012] Dabbish, L., Stuart, C., Tsay, J., e Herbsleb, J. (2012). Social coding in github: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pp. 1277–1286. ACM.

[Ehrlich e Shami 2008] Ehrlich, K. e Shami, N. S. (2008). Searching for expertise. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems*, pp. 1093–1096. ACM.

[Ferreira 2004] Ferreira, A. d. H. (2004). *Novo dicionário Aurélio da língua portuguesa*. Editora Positivo.

[Fritz et al. 2010] Fritz, T., Ou, J., Murphy, G. C., e Murphy-Hill, E. (2010). A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pp. 385–394, New York, NY, USA. ACM.

[Hassan 2008] Hassan, A. E. (2008). The road ahead for mining software repositories. In *Frontiers of Software Maintenance, 2008. FoSM 2008.*, pp. 48–57. IEEE.

[Ma et al. 2009] Ma, D., Schuler, D., Zimmermann, T., e Sillito, J. (2009). Expert recommendation with usage expertise. In *Proceedings of the 25Nd IEEE International Conference on Software Maintenance*, ICSM '09, New York, NY, USA. IEEE.

[McCuller 2012] McCuller, P. (2012). *How to recruit and hire great software engineers: building a crack development team*. Apress.

[McDonald e Ackerman 2000] McDonald, D. W. e Ackerman, M. S. (2000). Expertise recommender: a flexible recommendation system and architecture. In *Proce-*

edings of the 2000 ACM conference on Computer supported cooperative work, pp. 231–240. ACM.

[Minto e Murphy 2007] Minto, S. e Murphy, G. C. (2007). Recommending emergent teams. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pp. 5–, Washington, DC, USA. IEEE Computer Society.

[Mockus e Herbsleb 2002] Mockus, A. e Herbsleb, J. D. (2002). Expertise browser: A quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pp. 503–512, New York, NY, USA. ACM.

[Robbes e Röthlisberger 2013] Robbes, R. e Röthlisberger, D. (2013). Using developer interaction data to compare expertise metrics. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 297–300. IEEE Press.

[Santos et al. 2018] Santos, A., Souza, M., Oliveira, J., e Figueiredo, E. (2018). Mining software repositories to identify library experts. In *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse*, pp. 83–91. ACM.

[Santos et al. 2015] Santos, K. F., Guerrero, D. D. S., e Figueiredo, J. C. A. (2015). Using developers contributions on software vocabularies to identify experts. In *2015 12th International Conference on Information Technology - New Generations*, pp. 451–456.

[Sommerville et al. 2008] Sommerville, I., Arakaki, R., e Melnikoff, S. S. S. (2008). *Engenharia de software*. Pearson Prentice Hall.

[Takhteyev e Hilts 2010] Takhteyev, Y. e Hilts, A. (2010). Investigating the geography of open source software through github.

[Teyton et al. 2013] Teyton, C., Falleri, J., Morandat, F., e Blanc, X. (2013). Find your library experts. In *20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013*, pp. 202–211.

[Zagalsky et al. 2015] Zagalsky, A., Feliciano, J., Storey, M.-A., Zhao, Y., e Wang, W. (2015). The emergence of github as a collaborative platform for education. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, pp. 1906–1917. ACM.

[Zhong et al. 2009] Zhong, H., Xie, T., Zhang, L., Pei, J., e Mei, H. (2009). Mapo: Mining and recommending api usage patterns. In *European Conference on Object-Oriented Programming*, pp. 318–343. Springer.

Apêndice A

Questionário do Levantamento para Validação do Protótipo de Ferramenta

A.1 Search/Recommendation System on Software APIs

Hello Guys!

I'm developing a recommendation system for software API experts as my master thesis proposal. The system is ready and I need to validate if it is recommending correctly.

For this, I will ask your help to answer this small questionnaire below giving scores of 0-10 for each of the developers. For each developer, you will assign a note that you think the other developer knows about this API / Library.

At the end, I will make a comparison of the results that you informed me with the results extracted from the tool.

Email: _____

How much do you think DEVELOPER_1 knows about the Google Maps API?

0 1 2 3 4 5 6 7 8 9 10

How much do you think DEVELOPER_2 knows about the Google Maps API?

0 1 2 3 4 5 6 7 8 9 10

How much do you think DEVELOPER_3 knows about the Google Maps API?

0 1 2 3 4 5 6 7 8 9 10

How much do you think DEVELOPER_4 knows about the Google Maps API?

0 1 2 3 4 5 6 7 8 9 10

How much do you think DEVELOPER_5 knows about the Google Maps API?

0 1 2 3 4 5 6 7 8 9 10

How much do you think DEVELOPER_6 knows about the Google Maps API?

0 1 2 3 4 5 6 7 8 9 10

Apêndice B

Questionário do Levantamento para os Desenvolvedores do GitHub

B.1 Expert Finding on GitHub

Hello,

How are you? I hope you are doing fine :)

Firstly, I would like to introduce myself. My name is Camille Nogueira and I am a Master's student at the State University of Feira de Santana, Bahia, Brazil. I am doing a software engineering research that tries to identify experts in software APIs from open source projects extracted from GitHub.

And how did we come to you? We are looking to identify experts in the [API'S NAME] API from the [PROJECT'S NAME] and your name (and a few other contributors) has appeared on the contributors' list for these projects. It would be really cool if you could help us to identify who the experts are, as well as to help my completing this research work.

However, to evaluate whether our tool and methods are correct, I'd like to ask you a few questions just to validate our recommendation list. At the end of the process, in case you want to know the results, we can send them to your email as well.

Do not worry, this survey is very quick and objective. It will take only 2 (two) minutes of your time.

Any questions you might have, you may contact me via email: camillenogueira@gmail.com

On a scale from 0 to 10, how much do you think DEVELOPER_1 about the Apache Commons API?

0 1 2 3 4 5 6 7 8 9 10

On a scale from 0 to 10, how much do you think DEVELOPER_2 about the Apache Commons API?

0 1 2 3 4 5 6 7 8 9 10

On a scale from 0 to 10, how much do you think DEVELOPER_3 about the Apache Commons API?

0 1 2 3 4 5 6 7 8 9 10

On a scale from 0 to 10, how much do you think DEVELOPER_4 about the Apache Commons API?

0 1 2 3 4 5 6 7 8 9 10

On a scale from 0 to 10, how much do you think DEVELOPER_5 about the Apache Commons API?

0 1 2 3 4 5 6 7 8 9 10