Instituto Federal de Educação, Ciência e Tecnologia da Paraíba

Campus João Pessoa

Programa de Pós-Graduação em Tecnologia da Informação

Nível Mestrado Profissional

WEMERSON THAYNE VITAL PORTO

ANÁLISE COMPARATIVA ENTRE TÉCNICAS DE AUTORIA DE CÓDIGO-FONTE

DISSERTAÇÃO DE MESTRADO

JOÃO PESSOA

2023

Wemerson Thayne Vital Porto

Análise comparativa entre técnicas de autoria de código-fonte

Dissertação apresentada como requisito parcial para obtenção do título de Mestre em Tecnologia da Informação, pelo Programa de Pós-Graduação em Tecnologia da Informação do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba – IFPB.

Orientador: Prof. Dr. Katyusco de Farias Santos

João Pessoa

2023

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Nilo Peçanha do IFPB, *campus* João Pessoa

P853a Porto, Wemerson Thayne Vital.

Análise comparativa entre técnicas de autoria de código-fonte / Wemerson Thayne Vital Porto. - 2023.
96 f. : il.
Dissertação (Mestrado -Tecnologia da Informação) - Instituto Federal de Educação da Paraíba / Programa de Pós-Graduação em Tecnologia da Informação (PPGTI), 2023.
Orientação : Profº. D.r Katyusco de Farias Santos.

1. Desenvolvimento de *software* – análise de código-fonte. 2. Conhecimento – código-fonte. 3. Especialista em código-fonte. 4. Métricas de conhecimento. I. Título.

CDU 004.4(043)

Lucrecia Camilo de Lima
Bibliotecária – CRB 15/132

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

PROGRAMA DE PÓS-GRADUAÇÃO *STRICTO SENSU*

MESTRADO PROFISSIONAL EM TECNOLOGIA DA INFORMAÇÃO

WEMERSON THAYNE VITAL PORTO

ANÁLISE COMPARATIVA ENTRE TÉCNICAS DE AUTORIA DE CÓDIGO-FONTE

Dissertação apresentada como requisito para obtenção do título de Mestre em Tecnologia da Informação, pelo Programa de Pós- Graduação em Tecnologia da Informação do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba – IFPB - Campus João Pessoa.

Aprovado em 24 de Fevereiro de 2023

Membros da Banca Examinadora:

Dr. Katyusco de Farias Santos

IFPB - PPGTI

Dr. Ivaldir Honório de Farias Junior

UPE - PPGTI

Dra. Juliana Dantas Ribeiro Viana de Medeiros

IFPB - PPGTI

João Pessoa/2023

Este trabalho é dedicado ao meu Senhor e Redentor Jesus Cristo,
à minha família, em especial à minha amada esposa,
e à minha falecida avó, Mãe Neta.

AGRADECIMENTOS

Agradeço a Deus, por me guiar e ajudar nos desafios da vida. Dando saúde, força e oportunidades que me permitiram chegar até este momento.

À minha amada esposa, Francyanne Porto, por toda a força, paciência e incentivo dedicado a mim durante toda essa jornada.

À minha família, que dia após dia, sempre estiveram ao meu lado, me guiando para o caminho correto, por meio da educação e dos bons costumes, e por sempre me apoiar nos momentos mais difíceis, não me deixando desistir de tudo aquilo que eu sempre almejei.

À minha falecida avô, mãe neta, que tanto me inspirou em ser uma pessoa melhor, e em não desistir do mestrado.

Agradeço a todos os professores membros do PPGTI/IFPB pelo conhecimento compartilhado dentro e fora das salas de aula, em especial ao meu orientador, Katyusco de Farias Santos.

Por fim, obrigado a todos os colegas e amigos que, direta ou indiretamente, contribuíram para a conclusão deste mestrado.

RESUMO

Encontrar especialistas em desenvolvimento de software geralmente é uma despesa operacional significativa para as empresas. Para aliviar esse problema, alguns pesquisadores propuseram diferentes estratégias para encontrar especialistas. Apesar desses esforços, tais estratégias implicam em soluções específicas e julgamentos específicos que levam a diferentes conclusões, apesar de contribuições semelhantes. Neste trabalho, construímos a compreensão do domínio selecionando métricas de autoria de código-fonte por meio de uma revisão da literatura, a partir daí propusemos um protótipo de ferramenta que usa essas métricas na identificação de especialistas em código-fonte de software em vários projetos, disponível no repositório de código-fonte, GitHub. A partir desses resultados, realizamos uma avaliação com três abordagens baseadas: em número de *commit*, linhas de código modificadas e em grau de conhecimento, buscando analisar 100 projetos *open-source* para identificação de especialistas a partir do código-fonte. Nesta avaliação, realizamos uma comparação da convergência de especialistas a partir das métricas computadas pelo protótipo e percebemos que estas métricas apresentam uma relação de convergência de 50,15% nas indicações de especialistas. Avaliamos também as métricas em um projeto privado, para tanto, foi construído um oráculo para comparar os resultados apontados pelo protótipo e o conhecimento real dos desenvolvedores. Os resultados apontam que, para um cenário, de um pequeno projeto com poucos desenvolvedores, as métricas apresentam uma taxa de assertividade de 55%, 65% e 70%, para abordagens baseadas em número de linhas modificadas, métricas de número de *commit* e grau de conhecimento, respectivamente. Este trabalho adiciona-se ao corpo de conhecimento sobre determinação automática de conhecimento de código-fonte de software, mostrando a viabilidade e fornecendo uma avaliação do potencial uso de métricas de conhecimento de código-fonte usadas em repositórios de código-fonte de software.

Palavras-chaves: Análise de especialista de código-fonte; Conhecimento em código-fonte de software, Métricas de conhecimento em código-fonte.

ABSTRACT

Finding software development experts is often a significant operating expense for businesses. To alleviate this problem, some researchers have proposed different strategies for finding experts. Despite these efforts, such strategies imply specific solutions and specific judgments that lead to different conclusions, despite similar contributions. In this work, we build understanding of the domain by selecting source code authorship metrics through a literature review, from there we proposed a tool prototype that uses these metrics to identify experts in software source code in various projects, available in the source code repository, GitHub . Based on these results, we carried out an evaluation with three approaches based on: number of *commit*, modified lines of code and degree of knowledge, seeking to analyze 100 *open-source* projects to identify experts from the source code. In this evaluation, we performed a comparison of the convergence of specialists from the metrics computed by the prototype and we noticed that these metrics present a convergence ratio of 50.15% in the indications of specialists. We also evaluated the metrics in a private project, for that, an oracle was built to compare the results pointed out by the prototype and the real knowledge of the developers. The results indicate that, for a scenario, of a small project with few developers, the metrics present an assertiveness rate of 55%, 65% and 70%, for approaches based on the number of modified lines, metrics of number of *commit* and knowledge level, respectively. This work adds to the body of knowledge on automatic determination of software source code knowledge, showing the feasibility and providing an assessment of the potential use of source code knowledge metrics used in software source code repositories.

Key-words: Source code expert analysis; Software source code knowledge, Source code knowledge metrics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AC	*Acceptances*
API	*Application Programming Interface*
CSV	*Comma Separated Values*
CLI	*Command Line Interface*
CVCS	*Centralized Version Control Systems*
DVCS	*Distributed Version Control Systems*
DOA	*Degree of Authorship*
DOI	*Degree Of Interest*
DOK	*Degree Of Knowledge*
DL	*Deliveries*
DSR	*Design Science Research*
FA	*First Authorship*
LoC	*Lines of Codes*
SMO	*Social Media Optimization*
SVM	*Support Vector Machine*
REST	*Representational State Transfer*
RSL	Revisão Sistemática da Literatura
RS	Revisão Sistemática
VCS	*Version Control System*
YMAL	*Yet Another Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Identificar especialistas em desenvolvimento de software é uma tarefa complexa e envolve a seleção de currículos, realização de entrevistas, avaliação de experiência prévia e aplicação de testes. Existem vários métodos para identificar desenvolvedores apropriados para um projeto, desde processos tradicionais de entrevista até sistemas de recomendação e sistemas especialistas(SILVA, 2012; PERERA, 2017; WALKER, 2019; KUMAR; SINGH; NEVES, 2020).

Durante as etapas do processo de desenvolvimento de um projeto de *software*, gera-se uma abundante quantidade de artefatos novos, como documentação, diagramas e artefatos de código. Desses, o código-fonte está em constante evolução. Trechos são criados por um desenvolvedor, mas podem sofrer modificações de diferentes desenvolvedores que participam e colaboram com o projeto(FRITZ et al., 2010).

Os desenvolvedores, ao se depararem com partes do código que não tem conhecimento, em média levam cerca de 50% do tempo disponível para executar uma tarefa, para compreender seu funcionamento (BENNETT; RAJLICH, 2000). Já em estudos mais recentes, o trabalho de Yoon et al. (2016) descobriu que os desenvolvedores, ao se depararem com partes do código desconhecidas, levavam cerca de 60% do tempo disponível para compreender a tarefa. Além de Yoon et al. (2016), outros trabalhos realizados para avaliar o tempo gasto por desenvolvedores para entender códigos desconhecidos. Por exemplo, o estudo de Bhat e Robillard (2016) percebeu que os desenvolvedores gastavam cerca de 55% do tempo disponível para compreender códigos desconhecidos. Além destes, o artigo (RASTOGI; ARASU; ROBILLARD, 2018) detectou que os desenvolvedores gastavam cerca de 65% do tempo disponível para compreender códigos desconhecidos.

É comum não saberem a quem, entre os demais contribuidores, solicitar ajuda para realizar uma tarefa de manutenção: correção de um bug, revisão de código ou adição de nova funcionalidade. Os custos de manutenção são elevados, chegando a representar cerca de 70% a 90% do valor total de um software (ROGER; BRUCE, 2015). No estudo proposto por Abbasi et al. (2017) observou que os custos de manutenção de software podem representar até 85% do total de um software. Outro estudo que corroboram com a perspectiva dos altos custos de manutenção de software podem representar até 80% do total de um software(HOSSAIN et al., 2018). Em 2020, o trabalho de Gao et al. (2020) descobriu que os custos de manutenção podem representar até 95% do total de um software.

Em projetos de softwares é habitual ter arquivos com vários desenvolvedores, entre esses, alguns são autores das principais alterações nos arquivos, no entanto, outros podem ser apontados como desenvolvedores periféricos, que realizam apenas tarefas de manutenção menores e menos

importantes(CORLEY; KAMMER; KRAFT, 2012). No trabalho desenvolvido por (TORKAR; PARNAS; LANO, 2019) descreveram três tipos principais de participantes no desenvolvimento de software: os principais autores, os participantes-chave e os participantes periféricos.

Velasquez e Silva (2016) descobriram que os desenvolvedores de software têm hábitos de codificação variados. Eles descreveram quatro hábitos de codificação comuns entre os desenvolvedores de software: adicionar comentários, refatorar, testar e documentar. Eles também discutiram como esses hábitos de codificação afetam o desenvolvimento de software, como aumentam a qualidade do código e facilitam a manutenção e a modificação ao longo do tempo.

Segundo Fritz et al. (2010), define que a especialidade em código-fonte é habilidade do desenvolvedor ser detentor do conhecimento sobre os elementos e estruturas no qual compõem um dado código-fonte. Existem diversas abordagens para identificar os especialistas(FRITZ et al., 2014; SANTOS; GUERRERO; FIGUEIREDO, 2015; SVIRIDOV; EVTIKHIEV; KOVALENKO, 2021), porém boa parte têm como princípio minerar dados de autoria existentes nos repositórios de código, como, por exemplo: número de contribuições em uma entidade de código, número de linhas modificadas. No artigo de Wang e Gao (2019) descreveram uma abordagem baseada em indicadores de contribuição, como o número de commits, número de bugs corrigidos para identificar os especialistas.

É comum que partes dos códigos criados inicialmente por um desenvolvedor, no andamento do projeto, recebam contribuições de códigos provenientes de outros desenvolvedores, essas contribuições realizadas, sobre um código-fonte podem ser tantas que o especialista que antes era o autor do código, após as mudanças, pode passar ser aquele que mais contribuiu para o dado artefato de código(FRITZ et al., 2014). A recomendação correta de um especialista de código pode aumentar a produtividade, uma vez que ele realize as mudanças no código-fonte e é provável, que sejam executadas com o menor impacto em termos de tempo e esforço, por isso, é necessário definir quem, entre os desenvolvedores de uma equipe, é o desenvolvedor mais adequado para realizá-las(STEEB; BISSYANDÉ, 2017).

1.1 Motivação e Definição do Problema

Estudos anteriores mostram que desenvolvedores de *software*, normalmente, recebem uma enxurrada de informações diárias, como, por exemplo, reuniões de planejamento de tarefas, acompanhamento de métricas e andamento de atividades. Assim, melhorar a assertividade dos desenvolvedores em filtrar essas informações pode acarretar um significativo aumento na produtividade(OU, 2009). Como nos mostram os artigos, que a curva de aprendizado dos desenvolvedores de software(LIN; WENG, 2017), o desenvolvimento de software ágil(SREEKANTH; RUBAN, 2015) e as melhores práticas dos gestores de projetos(BANAEI; GRAY, 2014) podem aumentar significativamente a produtividade dos desenvolvedores de software.

Os repositórios armazenam o histórico de mudança, registrando um *log* para cada altera-

ção no código, assim, representa uma rica quantidade de dados sobre as mudanças de um sistema de *software*. Eles permitem rastrear toda e qualquer mudança ocorrida em qualquer um dos artefatos em um projeto, como o código-fonte, manuais de documentação(MALHOTRA, 2016). Neste contexto, as técnicas mais utilizadas nos estudos para identificar especialistas têm como princípio recuperar informações contidas nos repositórios de código(SANTOS; GUERRERO; FIGUEIREDO, 2015). Então extrair, manipular e analisar o repositório de código-fonte para mostrar dados referente aos conhecimentos de cada desenvolvedor nos projetos de software é de grande importância para a vida dos projetos de *softwares*.

Com intuito de identificar o desenvolvedor especialista no código-fonte, escolhemos as métricas baseadas no número de *commit*, números de linhas e grau de autoria. A escolha se deu por dois motivos, o primeiro deles foi a vasta exploração delas nos trabalhos presentes na literatura(ZHU et al., 2016; GUO et al., 2018; WANG et al., 2019; ZHOU et al., 2020). O outro fator é que as métricas capturam situações diferentes como, por exemplo, uma modificação de endentação não é capturada pela métrica LoC(Linhas de código), mas sim por *commits*, logo cada uma tem sua precisão e assertividade.

Em virtude de cada abordagem ter situações distintas na sua computação, podem apresentar desenvolvedores especialistas diferentes entre elas, é comum haver divergência entre si na recomendação de um desenvolvedor como especialista, por exemplo, a métrica de *commit* indica que o desenvolvedor X é especialista, a métrica LoC aponta que é o desenvolvedor Y é o especialista e a métrica de grau de autoria indicar o desenvolvedor Z é o especialista.

A granularidade das informações de autoria de código-fonte oferecidas pelos sistemas de versionamento atuais não refletem adequadamente a relação entre os desenvolvedores e código-fonte. Logo, a recomendação de um desenvolvedor especialista pode gerar dúvidas na sua escolha.

Então, o que gostaríamos de verificar é se conseguimos aumentar a assertividade fazendo a convergência entre as 3 principais métricas, número de *commit*, números de linhas modificadas e grau de autoria, no qual a solução pode ser aplicada para auxiliar na identificação dos possíveis desenvolvedores especialistas do código-fonte de um projeto de *software*.

Ademais, é, inicialmente, destinado a desenvolvedores e pesquisadores da área de engenharia de *software* que desejam extrair dos seus repositórios de código-fonte, dados sobre o conhecimento de código.

1.2 Objetivos

1.2.1 Objetivo geral

Este trabalho visa analisar as abordagens de autoria de código-fonte utilizadas na identificação de desenvolvedores especialistas em código-fonte a partir da medição da convergência das

recomendações indicadas pelas métricas de autoria baseados em número de *commits*, números de linhas modificadas e pelo grau de autoria.

1.2.2 Objetivos específicos

- Realizar uma revisão bibliográfica *quasi*-Sistemática da Literatura baseado no trabalho de Abrantes e Travassos (2007) sobre as técnicas de identificação de desenvolvedores especialistas em código-fonte e possíveis ferramentas;
- Avaliar se a convergência entre as técnicas consegue aumentar a assertividade nas indicações dos desenvolvedores especialistas;
- Desenvolver um ferramental para extrair, manipular e analisar os repositórios de código-fonte para mostrar dados referente aos conhecimento de cada desenvolvedor nos projetos de software:
  - Implementar pelo menos três técnicas(LoC, *Commit* e DOA(Grau de autoria) diferentes de identificação de especialistas;
  - Executar cada uma das três técnicas sobre projetos reais em repositórios Git;
- Realizar uma avaliação experimental do ferramental implementado, a fim de validar as funcionalidades e verificação das diferentes métricas usadas;

1.3 Questões de Pesquisa

Após essa breve contextualização acerca da motivação, problema, e apresentações dos objetivos, as seguintes questões de pesquisa foram levantadas:

- QP1) Qual o percentual da divergência entre as métricas de autoria de código-fonte?
- QP2) Convergência entre pelo menos duas métricas aumenta a assertividade da indicação de especialista?
- QP3) Convergência total das três métricas aumenta a assertividade da indicação de especialista?

1.4 Estrutura do Documento

Estruturalmente, o documento encontra-se organizado da seguinte maneira:

No Capítulo 2, é apresentado uma rápida explanação sobre o desenvolvimento e evolução do código-fonte, sistemas de controle de versão, conhecimento de código, forma de metrificar o

conhecimento sobre o código-fonte e trabalhos relacionados na qual é apresentada uma série de artigos que representam o estado da arte relacionado ao tema em estudo.

No Capítulo 3 é descrito a metodologia utilizada no desenvolvimento da pesquisa, incluindo também uma breve descrição do processo de construção do protótipo da ferramenta, procedimento de avaliação da ferramenta e métricas.

No Capítulo 4, é apresentada uma visão geral do protótipo da ferramenta desenvolvida para a identificação de especialista de código-fonte.

No Capítulo 5, é apresentado o processo de avaliação das métricas para a identificação de especialistas.

Finalizando o documento, o Capítulo 6 apresenta as considerações finais, limitações da pesquisa e propostas de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

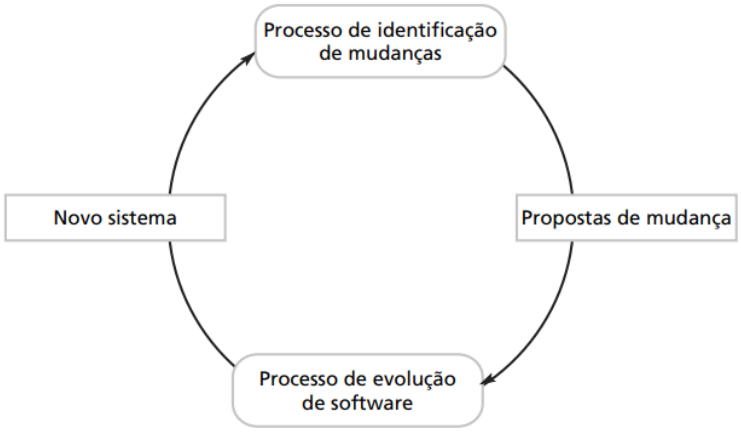
Nesta seção, serão apresentados os principais fundamentos teóricos utilizados para o desenvolvimento deste trabalho. Uma rápida explanação sobre os conceitos fundamentais da evolução do código-fonte, sistemas de controle de versão, conhecimento de código, formas de metrificar o conhecimento sobre o código-fonte. São apresentados os Trabalhos Relacionados com o tema em estudo, em que são discutidos artigos relevantes sobre o tema.

2.1 Desenvolvimento e evolução de software

Software é um produto abstrato projetado para automatizar atividades difíceis e de valor elevado, em esforço e tempo, para serem realizadas manualmente por seres humanos(SANTOS; GUERRERO; FIGUEIREDO, 2015). O desenvolvimento de um software envolve algumas etapas como planejamento, levantamento de requisitos, documentação e codificação, onde nestas vários artefatos são gerados, dentre elas a codificação gera a maior delas(GIRBA et al., 2005).

A evolução de um sistema de software podem variar dependendo do tipo de projeto que esteja sendo criado, e o processo para atender as solicitações de evoluções podem sofrer mudanças a depender da empresa. Em algumas organizações, a evolução pode ser um processo informal (conversa entre os envolvidos) e em outras empresas um processo formal (documentação estruturada). Como demonstrado na Figura 1, o processo de evolução e modificação em um software(SOMMERVILLE, 2013).

Figura 1 – Processos de identificação de mudanças e de evolução.



Fonte: (SOMMERVILLE, 2013).

As etapas apresentadas na Figura 1 por (SOMMERVILLE, 2013) são divididas em quatro fases: identificação de mudanças, propostas de mudança, processo de evolução do *software* e Novo sistema. As mudanças podem ser desde simples alterações para correção de erros de

codificação, até mudanças mais complexas, como melhorias significativas para corrigir erros de especificação ou implementação de novos requisitos.

Na etapa de proposta de mudanças são avaliadas as condições atuais do sistema, para identificar os pontos de modificações e impactos no funcionamento do *software*, após identificação da mudança e impactos a implementação acontece na terceira fase, com isso gerando um novo artefato de *software*, no qual esse processo ocorrem várias vezes ao longo da vida de um sistema.

2.2 Sistemas de controle de versão e Repositório

Um sistema de controle de versão de código-fonte, VCS(*Version Control System*) é uma ferramenta de software projetada para auxiliar projetos de software na modificação de seu código-fonte. O sistema deve fornecer facilidades para armazenar, modificar e baixar todas as versões dos módulos por número de versão para controlar direitos para modificar o código-fonte(ROCHKIND, 1975; RIEHLE; LAGO, 2007; KORKMAZ, 2018; UDDIN; RAHMAN; AL-HADDAD, 2020).

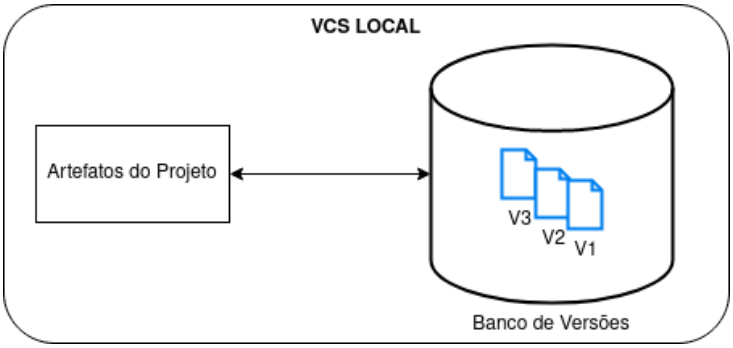
O versionamento foi criado para solucionar problemas entre diferentes alterações de software em suas diversas versões. Assim nasceu o sistema de versão para registrar o histórico da versão, para coletar informações sobre as alterações feitas no desenvolvimento do software(MALHOTRA, 2016). O sistema de controle de versão do código-fonte com as seguintes funções:

- Armazenamento;
  - Uma ferramenta de VCS deve ter funcionalidade de registro de mudanças, pois sem registro não é possível fornecer funcionalidade de controle de versão de código. Todas as versões de código devem ser preservadas.
- Identificação;
  - Um VCS deve ter um identificador para que cada código e arquivo seja atribuído aos autores. O sistema coleta informações de identificação do desenvolvedor, além do número da versão, data, hora e mensagens que identificam o envio do código.
- Proteção;
  - O VCS dever conseguir fazer o controle de acesso aos códigos e módulos do software, possibilitando somente aqueles com permissão de alteração poderá realizá-la.

O VCS são categorizados da seguinte forma pelo(http://git-scm.org), como VCS Local, Centralizado, Distribuído, abaixo descrevamos em breves palavras o funcionamento de cada tipo.

- VCS Local: Emprega um banco de dados simples que registra e mantém todas as alterações nos artefatos do projeto de software sob controle de revisão. A Figura 2 apresenta o conceito de um VCS local, o método deste tipo de controle opera simplesmente registrando os conjuntos de *patch*(ou seja, as diferenças entre dois artefatos) enquanto se move de uma revisão para outra em um formato específico.

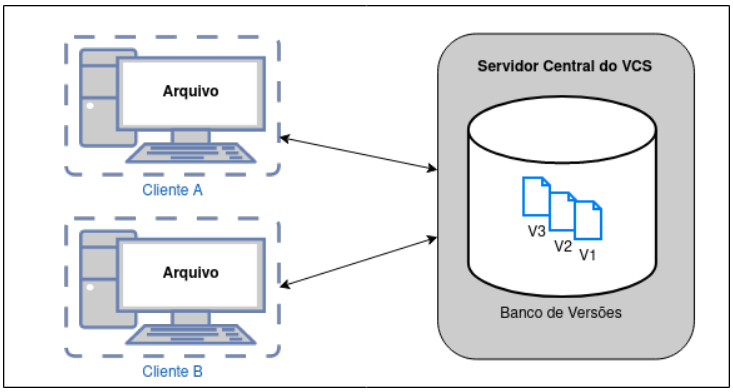
Figura 2 – VCS local.



Fonte: autoria própria

- VCS Centralizado: Sistemas de controle de versão centralizada(CVCS), foram desenvolvidos para facilitar o compartilhamento das versões entre os desenvolvedores do software. Possuem um único servidor que contém todos os arquivos de controle de versão e muitos clientes(usuários do CVCS) que acessam os arquivos desse local central. A Figura 3 apresenta o conceito de um CVCS.

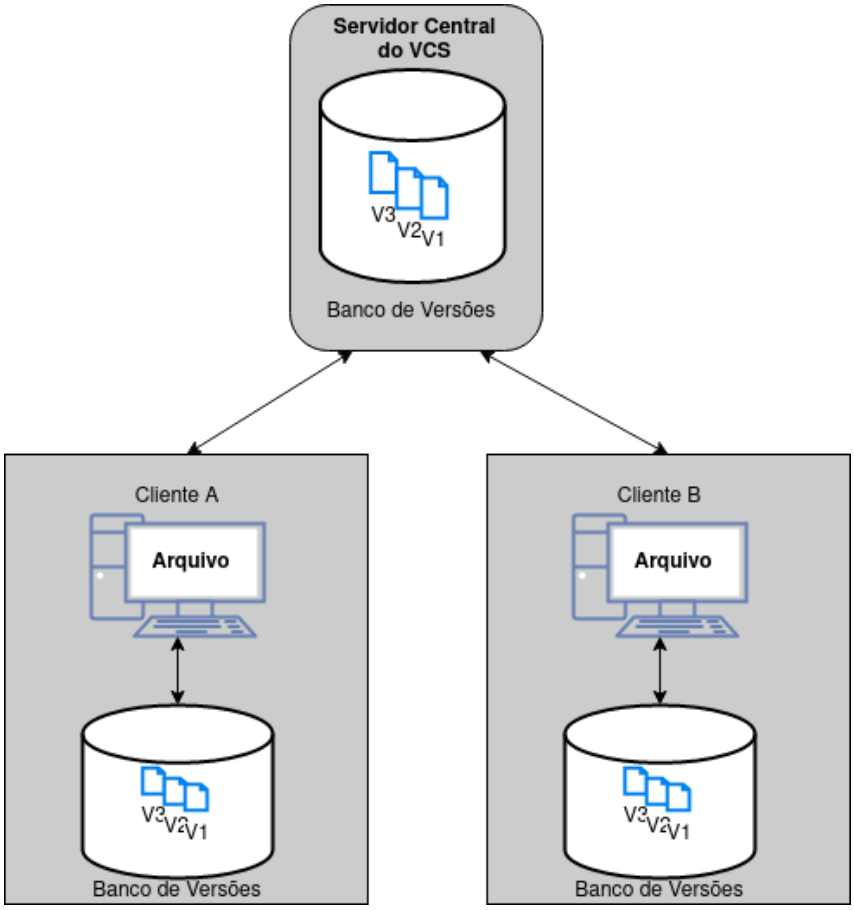
Figura 3 – VCS centralizado.



Fonte: autoria própria

- VCS Distribuído: Sistemas de controle de versão centralizada(DVCS) permitem que os clientes não usem apenas o estado mais recente dos arquivos: eles copiam todo o arquivo localmente. Portanto, se um servidor morrer, o armazenamento de dados de cada cliente poderá ser copiado de volta para o servidor para restaurá-lo. Cada clone é, na verdade, um backup completo de todos os dados. Exemplos de DVCS: *Git, CVS, Subversion (SVN), Perforce e ClearCase*.A Figura 4 apresenta o conceito de um DVCS.

Figura 4 – VCS distribuído.



Fonte: autoria própria

Neste trabalho, o *Git* é considerado o principal controle de versão de código-fonte, pois é amplamente utilizado pela comunidade de software livre, além de desenvolvedores da indústria de software.(MAJUMDAR et al., 2017).

2.2.1 Repositório de código

Os repositórios de código contêm uma rica quantidade de informações sobre as mudanças sofridas por um sistema de software, armazenam um conjunto de dados que variam entre: identificadores, registros de autoria da ação e de *timestamp*, comentários com informações complementares e os artefatos de código-fonte e suas diferenças entre as versões. Ao aplicar as informações extraídas desses repositórios, pesquisadores e profissionais de engenharia de software não precisam depender principalmente de sua intuição e experiência, mas sim de dados históricos(ALALI; KAGDI; MALETIC, 2008).

Como as mudanças são inevitáveis em quase todos os grandes projetos de *software* (SOMMERVILLE, 2013), os requisitos do sistema mudam, logo, é imprescindível que o código acompanhe as mudanças para atender os novos requisitos. No trabalho realizado por Hattori e Lanza (2008) são definidos os tipos de mudança de um *software*, alterações, inserções e remoções

definidas respectivamente como, mudança na escrita de um código, adição de novos códigos e remoções linhas código-fonte ou arquivos do projeto.

A submissão de um conjunto de alterações no código-fonte nos registros do VCS e sua respectiva efetivação é denominado de *commit*. Ao criar o *commit* diversos dados são gerados, dependendo do VCS, mas no geral explicam o porquê, quem e quando da mudança ocorreu no código-fonte. O sistema de controle de versão *Git* possui um comando no qual consegue representar, recuperar as informações contidas em *commit* realizado no código-fonte, o comando é: *git log*

No Quadro 1 visualizamos o resultado da execução do comando *git log*. Onde conseguimos identificar, informações do autor(usuário e e-mail) do *commit*, data e hora da modificação e comentário do porquê da modificação.

Tabela 1 – Exemplo da execução do comando *git log*

```
commit e37acc99dc487286126f79a81b5c47cbc5b13ba2 (HEAD -> master, origin/master, origin/HEAD)
Author: author1 <author1@email.com>
Date: Tue Apr 19 10:47:46 2022 -0300

Author1 modificou File.java adicionando um novo método de 3 linhas, mais comentário.
```

Fonte: autoria própria

2.3 Conhecimento de código

A maioria dos projetos de desenvolvimento de software consiste em uma equipe de um ou mais desenvolvedores. No entanto, mesmo que um padrão detalhado de projeto e documentação de código seja mantido, para cada atividade que exija desenvolvimento, a equipe normalmente perguntará quem seria o melhor especialista para desenvolvê-la. Encontrar especialistas em software não é uma tarefa trivial, e um bom currículo não garante que uma pessoa é especialista para realizar uma mudança num dado sistema(DANTAS; MEIRA; MEDEIROS, 2016).

Em projetos de *software*, é intrínseco a necessidade de que seu desenvolvimento seja dividido entre os membros da equipe de desenvolvedores. Mesmo seguindo processos de desenvolvimento, manter constantemente atualizada a informação de quem é o desenvolvedor mais adequado para realizar uma determinada tarefa de programação, é difícil(SANTOS; GUERRERO; FIGUEIREDO, 2015). Como esse desenvolvedor já detêm um conhecimento adequado sobre o código, logo ele perderá menos tempo com o entendimento do mesmo. Isso abre espaço para ele trabalhar com menos tempo e esforço, portanto, mais eficiente nas atividades de evolução do código.

Existem maneiras que podem provar os conhecimentos de um desenvolvedor diretamente, por exemplo, um diploma de certificação. Um desenvolvedor iniciante com certificação Java concedida pela Oracle está apto a fazer parte de um time de desenvolvimento de um sistema

Java. No entanto, esse mesmo desenvolvedor muito provavelmente não será o mais indicado para corrigir um defeito (*bug*) reportado sobre uma dada entidade, ou parte do sistema, já que ele não terá conhecimento prévio sobre os conceitos, abstrações e comportamento da entidade, e funcionalidades do sistema.

Mockus e Herbsleb (2002) definem conhecimento de código como a habilidade do desenvolvedor de ser especialista sobre um código-fonte, refletindo na habilidade de uma pessoa em executar determinada tarefa. A identificação de especialistas é o problema de encontrar entre o conjunto de desenvolvedores aquele que têm o conhecimento sobre determinada parte do código. Acrescentou também em seu trabalho que é difícil medir diretamente o nível de conhecimento de cada membro da equipe e também qual a medida mais indicada para medir a especialidade do desenvolvedor(MOCKUS; HERBSLEB, 2002).

2.4 Medição do conhecimento de código

Para Santos, Guerrero e Figueiredo (2015), as técnicas mais disseminadas são as abordagens baseadas em heurísticas, aplicadas sobre os dados coletados a partir do histórico do desenvolvimento, contidas nos repositórios de códigos-fontes registrados pelos VCS para determinar o conhecimento sobre o código.

Uma das mais simples é baseada na ideia de que especialista é aquele desenvolvedor que foi o último a modificar um dado código-fonte. Essa foi a abordagem utilizada por McDonald e Ackerman (2000), que propuseram ainda uma arquitetura para armazenar histórico de mudanças e delas identificar especialistas para solucionar problemas similares.

Um conceito importante para determinar um especialista de código, ou seja, um desenvolvedor, fundamentam-se em dois pontos: autoria(*Authorship*) e propriedade(*Ownership*). A autoria acontece quando um programador escreve um código, tornando-se o autor dessa parte. No que lhe concerne, a propriedade ou posse se dá quando um programador agrega um novo código ou realiza alterações no repositório(CORLEY; KAMMER; KRAFT, 2012).

A seguir, apresentamos os principais tipos de contribuições consideradas nos estudos para identificar um especialista de código.

2.4.1 Contribuições por arquivos

É definida pela quantidade de arquivos modificados pelo desenvolvedor(MCDONALD; ACKERMAN, 2000). As constantes criações e alterações nos arquivos gerados durante o processo de desenvolvimento podem determinar quais arquivos são atualizados exclusivamente por apenas um único desenvolvedor e por mais nenhum outro. Assim, um desenvolvedor que realiza *commits* de alterações em arquivos dispõe (ou adquire) o conhecimento desses arquivos(ALALI; KAGDI; MALETIC, 2008).

Arquivos que, por muito tempo, tenham sucessivas modificações por um único desenvolvedor, muito provavelmente terão seus códigos fontes dominados por apenas esse desenvolvedor. Usualmente as contribuições por arquivos são utilizadas em conjunto com outras medidas de conhecimento de código, por exemplo, com contribuições por desenvolvedores.

2.4.2 Contribuições por desenvolvedores e arquivos

A regularidade e o tamanho das mudanças em arquivos concretizadas pelos desenvolvedores podem ser utilizadas também como uma medida de contribuição. Um desenvolvedor que realiza modificações nos arquivos tem (ou obtém) conhecimento sobre esses arquivos(MOCKUS; HERBSLEB, 2002).

Em relação à quantidade de *commits* em determinados arquivos por um dado desenvolvedor, indica que, quanto maior o número, maior a probabilidade do seu conhecimento estar naqueles arquivos. Esse desenvolvedor pode ser considerado especialistas com conhecimento em profundidade(CROWSTON; HOWISON; ANNABI, 2006). Em contrapartida, especialista com conhecimento espalhado, é um programador que realiza *commits* em vários arquivos, o que sugere que seu conhecimento sobre o projeto é transversal, e não específico a poucos arquivos(MEIRELLES et al., 2010).

2.4.3 Contribuições por *commits*

É comum em um sistema de *software* ao longo do seu ciclo de vida sofrer uma enorme quantidade de *commits*. Um desenvolvedor pode contribuir com múltiplos *commits* para um único arquivo. Também, múltiplos desenvolvedores podem mudar o mesmo arquivo por *commits* distintos. Deste modo, *commits* permitem analisar as contribuições dos programadores(ALALI; KAGDI; MALETIC, 2008).

A fim de determinar o conhecimento do desenvolvedor, calcula-se a quantidade de *commits* para uma parte do código é dada pelo total ou percentual de *commits* por ele realizado sobre o referido código (MOCKUS; HERBSLEB, 2002; FRITZ et al., 2010). Neste caso, o especialista de um trecho de código é aquele desenvolvedor que realizou o maior número de *commits*.

2.4.4 Contribuições pelo número de linha de código

A métrica por número de linhas de códigos (*Lines of Codes* — LoC) pode ser baseadas em: linhas adicionadas; máximo de linhas de código adicionado a um arquivo; número médio adicionado; linhas de código excluídas; máximo excluído e média de linhas(MALHOTRA, 2016). A granularidade dessa medida pode ser no nível de elemento, classe, arquivo, módulo ou sistema.

Essas medidas são úteis para complementar algumas situações quando o conhecimento de código é medido por meio de contribuições por *commits*. Como, por exemplo, o *commit*

resultado da alteração de endentação do código, assim imputando erroneamente o conhecimento do desenvolvedor que o corrigiu(SANTOS; GUERRERO; FIGUEIREDO, 2015). Para Girba et al. (2005), entre os desenvolvedores, aquele que possuir o maior percentual de linhas modificadas é considerado o especialista, seja do arquivo ou projeto.

2.4.5 Contribuições baseadas no modelo grau de conhecimento

O modelo de contribuições *Degree Of Knowledge*(Grau de conhecimento — DOK) proposto por Fritz et al. (2014), busca identificar especialistas, valorando a combinação dos componentes de autoria, chamado grau de autoria (*Degree Of Authorship*- DOA), e a componente de interação sobre o código grau de interesse (*Degree Of Interest* — DOI).

Os dados do DOA são extraídos dos VCS, enquanto os dados do DOI representam o conhecimento adquirido pelos desenvolvedores decorrentes das mudanças por eles realizadas sobre os elementos de código. DOA aumenta quando se cria elementos (*First Authorship* — FA) e quando se entrega modificações do elemento criado (*Deliveries* - DL), e diminui quando se aceita modificações de outros desenvolvedores sobre um elemento criado (*Acceptances* - AC). O DOI aumenta quando o desenvolvedor interage com o elemento e diminui quando outros elementos sofrem interações. DOK consiste em modelar o fluxo e o contra-fluxo de múltiplos desenvolvedores modificando um mesmo elemento. O DOK é calculado da seguinte forma:

- Equação DOK:

$$DOK = \alpha_{F_A} \cdot FA + \alpha_{D_L} \cdot DL + \alpha_{A_C} + \beta \cdot DOI \tag{1}$$

- onde: α e β, são pesos para os fatores de *DOA* e de *DOI* respectivamente.

2.4.6 Contribuições por vocabulário de software

A métrica proposta por Santos e colegas(SANTOS; GUERRERO; FIGUEIREDO, 2015) em seu trabalho usa, para identificar quem são os especialistas de código uma abordagem que utiliza como modelo de conhecimento, a similaridade entre o vocabulário presente nas entidades de código e o vocabulário manipulado por cada um daqueles que ao longo da evolução do projeto participou como integrante da equipe de desenvolvimento.

A abordagem é composta por três grandes fases:

1. Coletar as contribuições para cada um dos desenvolvedores sobre o vocabulário. Consiste em percorrer todo o repositório alimentando o modelo de conhecimento, com base no vocabulário dos desenvolvedores, com os dados recuperados a cada *commit*;

2. Extrair o vocabulário das entidades de código-fonte; extrair seu vocabulário, e granulariza por entidade de design(classes, métodos, arquivos);

3. Por último, para uma dada entidade computar a similaridade do seu vocabulário com o vocabulário de cada um dos desenvolvedores. Aquele com o maior valor de similaridade é considerado o desenvolvedor mais adequado para manipular a entidade, ou seja, o especialista do código da entidade.

2.5 Trabalhos Relacionados

Nesta seção, são apresentados 8 trabalhos relacionados ao tema em estudo, seguindo uma ordem cronológica de publicação do mais antigo para o mais recente. Os estudos fornecem uma visão do estado da arte e o potencial da aplicabilidade da necessidade de medição dos especialistas de código-fonte encontrada na literatura.

Meng et al. (2013) desenvolveram em seu estudo uma investigação dos repositórios independente do tipo do VCS utilizado para a construção do repositório de código, mas sim baseando-se em grafos, denominado VCS abstrato. No modelo criado, os nós dos grafos representam os *commits* realizados e as arestas o desenvolvimento de dependências entre as modificações. Propôs para encontrar a autoria do código-fonte em repositórios *git* dois conceitos para identificação de autoria: estrutural e ponderada. A autoria estrutural representa a história do desenvolvimento da linha de código e a autoria ponderada é um vetor de pesos das contribuições do autor originado da autoria estrutural da linha. O peso de um autor é definido por uma medida de mudança de código entre os *commits*. Este trabalho traz as seguintes contribuições: 1) O modelo de autoria estrutural e ponderada permitiu mostrar um nível de detalhe diferente de outras técnicas existentes, onde autoria ponderada tem a atribuição de autoria ao nível de linha, apresentados através de um estudo em projetos de código-fonte; 2) A criação da ferramenta *git-author*, sendo uma nova ferramenta embutida no git e implementa a autoria estrutural e a ponderada modelo de autoria.

Fritz et al. (2014) realizaram um estudo empírico e coletaram os dados de autoria FA, DL, AC e os dados de interação, para 40 elementos de código, escolhidos arbitrariamente, de um dado projeto. Construíram também um *survey* sobre 7 desenvolvedores que faziam parte do projeto, com isso obtiveram o grau de conhecimento, o valor de DOK, que cada um deles estabeleceu ter sobre os elementos. Por fim, para determinar o grau de conhecimento, os dados foram submetidos a uma regressão multilinear de DOK em função de DOA e DOI, ($DOK \sim DOA + DOI$). A predição expressa 25% (Coeficiente de Determinação, R2 = 0,25) da variabilidade de valores do DOK. Embora que o *p-value* para todo modelo e dos *p-values* para as variáveis preditoras FA, DL, AC e DOI preconizam uma significância estatística. O valor de R2 demonstra que ainda faltam 75% da variabilidade de DOK para ser explicado(esse percentual não explicado no trabalho(FRITZ et al., 2014)). Ou seja, há outros fatores que precisariam ser avaliados e considerados para indicar, com uma maior precisão, quem é o especialista do código-fonte.

Santos, Guerrero e Figueiredo (2015) propuseram uma abordagem para identificação dos especialistas nas entidades de código-fonte em Java, com base na semelhança entre o vocabulário dos desenvolvedores e o vocabulário do software do sistema. Em linhas gerais sua solução foi, percorrer o repositório de código-fonte e a cada modificação encontrada acumula ao vocabulário as contribuições realizadas por cada um desenvolvedor. Extraí o seu vocabulário de código-fonte, ao nível de entidade de código, classes e interfaces. Para identificar o especialista consiste em encontrar o desenvolvedor cujo vocabulário tem a maior similaridade com o vocabulário de uma dada entidade sobre a qual uma tarefa de codificação foi previamente localizada.

No trabalho desenvolvido por Zhang et al. (2017) apresenta uma abordagem de construção de perfis de autoria baseados em um modelo lógico orientado n-grama de nível de palavra contínua e n-grama(na área da linguística computacional, um n-grama[1] de nível de palavra discreto, e um modelo de contexto de vários níveis sobre operações, *loops*, matrizes e métodos. Seu trabalho utilizou a técnica do classificador de otimização sequencial mínima SMO(*Social Media Optimization*) para treinamento SVM(*Support Vector Machine*), empregado para determinar a autoria dos códigos-fonte. As principais contribuições deste trabalho realizado foram: a construção de perfis de autoria explícito e implícito com base nas características de programação de diferentes programadores sobre o uso de palavras-chave, identificadores, operadores, instruções, métodos e classes e padrões de agrupamento de programação entre essas diferentes granularidades de componentes de linguagens de programação.

A proposta do trabalho Santos et al. (2018) 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 as apresentadas no *GitHub*. Eles também fizeram uma pesquisa com estudantes de uma universidade para verificar se o método de recomendação proposto é valido. Segundo os autores, os resultados mostram que 89% dos desenvolvedores do *GitHub* selecionados relatam suas habilidades em sites de redes sociais como o *LinkedIn*.

No estudo de Avelino et al. (2019) foi resultante de uma continuação de outro artigo anterior que investigou autoria de código investigando o *kernel Linux*. Neste estudo ampliou a análise dessa vez para 118 projetos *open-source* do *GitHub*, no qual comparou e constatou a autoria calculados anteriormente para o *kernel Linux*. A medida de autoria utilizada para identificação dos especialistas de código foi o modelo DOA(FRITZ et al., 2014). Como resultado, coletou dados de autoria para cada arquivo, a fim de revelar a carga de trabalho dos desenvolvedores do Linux (75% deles são autores de no máximo 10 arquivos) e seu grau de especialização (mais

[1] é uma sequência contígua de n-itens de uma amostra de texto.

de 71% dos autores são especialistas em um único subsistema). Acompanhado a evolução das mudanças em 66 versões do Linux, identificou que o número de arquivos de autoria dos 10 principais desenvolvedores sofreu uma grande diminuição (de 43% dos arquivos do sistema para 14%). A principal contribuição do trabalho foi a validação e investigação de especialistas de códigos utilizando repositórios abertos de código-fonte em 118 projetos no *GitHub*, e também o uso do modelo DOA como medida de autoria de código, porém não foi descrito o uso do modelo DOA no trabalho.

Hammad et al. (2020) produziram em seu estudo a identificação de especialista com base nas mensagens de textos contidos nos *commits*. O objetivo do trabalho foi recuperar as palavras-chave frequentes para ajudar na compreensão do especialista. Assim, desenvolveram uma ferramenta para automatizar e ajudar a identificar a contribuição de desenvolvedores em termos de palavras-chave que marcam sua experiência. A técnica proposta analisa apenas o conteúdo textual dos componentes. Com base nessa análise, a experiência dos desenvolvedores é determinada pelos termos que aparecem em todos os *commits* durante o período específico. Termos de uso frequente que aparecem em *commits*, podem ser utilizados para refletir a experiência de desenvolvedores. As principais contribuições de pesquisa deste trabalho podem ser resumidas como: uma técnica leve, baseada na análise textual dos *commits*, para ajudar na identificação de experiência em desenvolvedores. Aumento na velocidade de identificação do possível especialista. Três definições para três tipos diferentes de experiência dos desenvolvedores. Um estudo de caso em três projetos de código aberto para ilustrar a técnica proposta.

Sviridov, Evtikhiev e Kovalenko (2021) com intuito de investigar códigos abertos, desenvolveram uma ferramenta(Tool for Mining of Socio-Technical-TNM) para minerar dados sociotécnicos(dados da interação entre as pessoas e a tecnologia nos locais de trabalho) em repositórios git. TNM funciona com repositórios git locais. Em sua construção implementou técnicas de mineração baseadas no grau de propriedade de arquivos com base em conhecimento(FRITZ et al., 2014), e aplicou o algoritmo *PageRank* para a análise de um histórico de *commit* para avaliação de influência de *commit*. A principal contribuição deste trabalho é a ferramenta TNM, que pode ser usada para extrair vários tipos de dados sociotécnicos de repositórios *git*. Uma ferramenta autônoma para minerar os dados. O valor da ferramenta é na redução do esforço de desenvolvimento para extração de dados sociotécnicos para análises posteriores.

Tabela 2 – Codificação dos artigos mapeados

Código	**Artigo**
MRAA	Mining software repositories for accurate authorship (MENG et al., 2013)
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; GUERRERO; FIGUEIREDO, 2015)
AIS	Authorship identification of source code (ZHANG et al., 2017)
MCA	Measuring and analyzing code authorship in 1+ 118 open source project (AVELINO et al., 2019)
MEDS	Mining expertise of developers from software repositories (HAMMAD et al., 2020)
TSR	Tool for mining of socio-technical data from git repositorie (SVIRIDOV; EVTIKHIEV; KOVALENKO, 2021)

Fonte: autoria própria.

Levantamos algumas questões para comparar os trabalhos relacionados a nossa pesquisa, essas questões ajudaram a esclarecer as semelhanças existentes nos modelos apresentados pelos artigos, bem como as discrepâncias. A Tabela 3 mostra as questões levantadas para a identificação de especialista em código-fonte, com cada artigo codificado com a legenda apresentada na Tabela 2.

Tabela 3 – Questões levantadas para a identificação de especialistas

	MRAA	**DOK**	**DCS**	**AIS**	**MCA**	**MEDS**	**TSR**
Investiga histórico de commit?	X			X	X	X	
Analisa commit de código?		X	X		X		X
Analisa o conteúdo dos arquivos alterados por commit?		X	X		X		X
Considera perda de conhecimento?		X				X	
Considera que o tempo é um fator importante para medir os conhecimentos?				X		X	
Considera a autoria de código um fator de conhecimento?	X		X		X		
Utiliza técnicas de AI/ Machine Learning?	X			X		X	X

Fonte: autoria própria.

A partir dessas questões, percebemos que existe uma similaridade entre as abordagens apresentadas em cada artigo em relação a cada uma das questões apresentadas. Classificamos essas semelhanças e as relacionamos com cada tipo de dado específico para cada métrica. Este conjunto de dados classifica as várias abordagens apresentadas nesses artigos para identificar especialistas em desenvolvimento de software.

Considerando o entendimento da relação entre os artigos e as métricas, foram mapeados oito dados comuns com base nos critérios acima. Esses dados são usados em pelo menos um dos artigos. Cada artigo analisado aborda pelo menos um dado em suas métricas. São esses:

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

Fritz et al. (2014) apresenta um percentual não explicado pelo DOK, e conclui que é devido à influência sobre o modelo do processo de desenvolvimento (fase do projeto, relação de propriedade entre desenvolvedores e os elementos de código), quanto do sistema (tipo, tamanho, complexidade).

Portanto, esse fato corrobora com uma conclusão comum a que chegam os estudos e pesquisas que buscam identificar especialistas: é preciso avançar e contemplar outros aspectos não utilizados em suas respectivas abordagens (MOCKUS; HERBSLEB, 2002; GIRBA et al.,

2005; AVELINO et al., 2019; SVIRIDOV; EVTIKHIEV; KOVALENKO, 2021). E, independentemente das medidas de contribuições ou do tipo da abordagem empregada, a informação sobre especialistas deve ser de fácil acessibilidade a desenvolvedores, testadores, e gerentes, para ajudá-los a rapidamente localizar os melhores indivíduos para executar cada tarefa de codificação(SANTOS; GUERRERO; FIGUEIREDO, 2015).

Nossa abordagem proposta apresenta alguns aspectos: primeiro, propusemos a combinação de três métricas diferentes para determinar a especialidade do desenvolvedor para ajudar no processo de atribuição de tarefas. Em segundo lugar, nossa análise é focada em identificar conhecimento de código dos repositórios de *software* sem a necessidade de outros recursos, como rastreamento de *bugs* no repositório ou interações sociais através dos comentários inseridos nos *commits*. Terceiro, a proposta não requer nenhum procedimento avançado de processamento de texto ou qualquer aprendizado de máquina, ou IA(Inteligencia artificial). Nosso trabalho usa da reunião das principiais contrições das abordagens propostas anteriormente para construção da nova proposta.

3 METODOLOGIA

Neste capítulo, apresenta-se a metodologia de pesquisa selecionada neste trabalho. Para isso, inicia-se com a classificação da metodologia, os procedimentos de pesquisas, e as técnicas de coletas de dados adotadas.

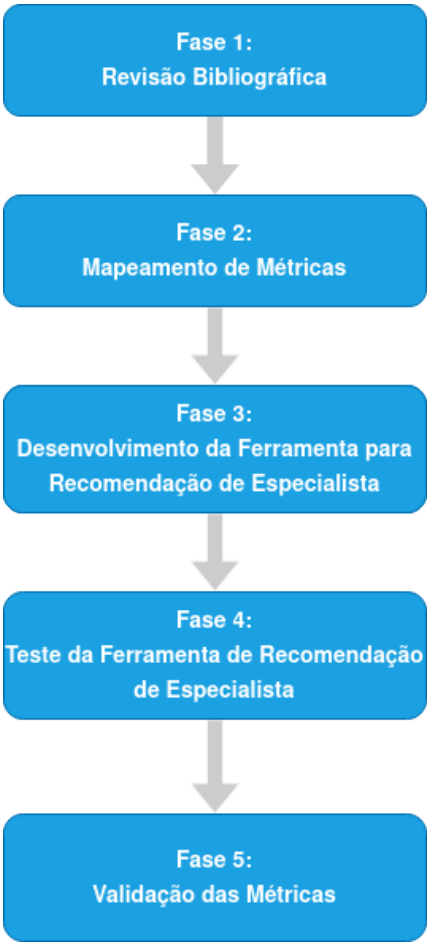
3.1 Metodologia

A metodologia aplicada neste trabalho foi uma adaptação do método *Design Science Research* (DSR) desenvolvida por (DRESCH; LACERDA; JUNIOR, 2015). O uso do DSR se dá por ter um objetivo mais amplo: gerar conhecimento que seja aplicável e útil para a solução de problemas, melhoria de sistemas existentes e criação de novas soluções e/ou artefato.

O processo metodológico deste trabalho foi dividido em 5 etapas: (1) Revisão bibliográfica, (2) Mapeamento de métricas, (3) Desenvolvimento da ferramenta para recomendação de especialistas em código-fonte, (4) Teste da ferramenta de recomendação de especialistas, (5) Avaliação das métricas.

Na Figura 5 apresenta as etapas metodológicas envolvidas para construção e desenvolvimento desta pesquisa.

Figura 5 – Etapas metodológicas do trabalho



Fonte: autoria propria.

3.1.0.1 Classificação da metodologia de pesquisa

A metodologia de pesquisa adotada neste trabalho foi classificada quanto à natureza, à abordagem, à tipologia, aos procedimentos técnicos e as técnicas de coleta de dados.

Quanto à natureza, esta pesquisa é baseada nos métodos contidos na engenharia de software empírica (ESE). Em suma, a ESE, é uma área de pesquisa que enfatiza o uso de métodos empíricos no campo da engenharia de *software*. Através de procedimentos para avaliar, prever, monitorar e controlar os artefatos existentes no desenvolvimento de *software*(MALHOTRA, 2016).

Quanto à abordagem, foi desenvolvida uma pesquisa do tipo empírica descritiva, na qual, visar descrever as características do sistema proposto, bem como analisar estatisticamente dos dados coletados através deste, ou seja, a pesquisa terá uma abordagem quantitativa(ZHANG et al., 2018).

Quanto aos fins, nesta pesquisa detalhamos as características em relação aos dados de autoria do código-fonte para identificação de especialista em código, além disso, são apresen-

tadas métricas e um método de medição para identificação, algoritmo de medição, regras de coleta(RITSCHEL; ZINK, 2018).

Quanto aos procedimentos técnicos de coleta de dados, os meios de investigação foram: pesquisa bibliográfica e análise de dados contidos em repositórios reais de código-fonte aberto no Github.

3.1.0.2 Plano metodológico adotado

Para Malhotra (2016), o processo de pesquisa inclui: planejamento; coleta de dados com análise e interpretação correspondentes e, em seguida, registrando o resultado. E todas essas etapas principais podem ser faseadas e mais detalhadas, resultando em sistemas bem diferentes. Esta seção descreve as etapas seguidas na condução deste estudo e as ferramentas usadas em cada etapa.

O processo utilizado neste estudo consiste nas seguintes etapas: (1) projeto; (2) coleta de dados; (3) analisar e interpretar os dados e (4) registrar os resultados. Partindo da metodologia de pesquisa adotada e definição das etapas que a compõem, desenvolveu-se o plano metodológico e etapas de pesquisa, utilizando procedimentos e técnicas de dados.

1. **Planejamento do trabalho**: nesta etapa, definimos o escopo do trabalho definindo a questão de pesquisa, o objetivo geral e os objetivos específicos. Essas definições são dadas na introdução deste trabalho no Capítulo 1

2. **Coleta de dados**: os procedimentos de pesquisa utilizados para coleta de dados foram:

    - Levantamento bibliográfico: a revisão de literatura(ABRANTES; TRAVASSOS, 2007) foi feita visando artigos científicos que embasassem as obtenções das métricas de autoria de código-fonte por um sistema de versionamento. Com isso foram pesquisados artigos relacionados a métricas de autoria de código-fonte, sistema de versionamento, uso de métrica de autoria e abordagens de medição.
    - Estudo de caso: De acordo com Yin(YIN, 2015), um estudo de caso é um método científico empírico de pesquisa que investiga dados em um contexto real através de análise aprofundada de um ou mais objetos de análise. A pesquisa de estudo de caso permite que as indústrias de software avaliem uma ferramenta, método ou processo(KITCHENHAM; PICKARD; PFLEEGER, 1995). Com isso, desenvolveu-se um estudo de caso com repositórios abertos de código-fonte disponíveis no Github para validar a proposta de identificação de especialista de código-fonte.

3. **Redação dos resultados**: A redação dos resultados desta pesquisa é constituída pelo referencial teórico deste trabalho, obtido por meio da pesquisa bibliográfica; a criação e validação da abordagem de identificação de especialistas por meio da convergência de métricas de autoria de código-fonte. Estes resultados são apresentados em forma de

capítulos nesta dissertação de mestrado, incluindo a conclusão sugestão de trabalhos futuros.

3.1.1 Revisão bibliográfica

Para a melhor compreensão do cenário a ser estudado, foi necessário um entendimento sobre os trabalhos relacionados e revisão da bibliografia, assim realizando uma revisão *quasi*-Sistemática da Literatura baseado no trabalho de Abrantes e Travassos (2007).

Nessa etapa, ao invés de produzir uma RSL(Revisão Sistemática da Literatura) nos moldes proposto por Nakagawa et al. (2017)[1], muito embora se entenda que seja a forma mais adequada para obtenção do panorama completo acerca do problema e das possíveis soluções, é sugerido a busca por trabalhos recentes relacionados e entender a bibliografia acerca do tema ou a busca por RSL recentes na literatura.

Uma vez não encontrando uma RSL recente, outro meio, são buscas nas bases de dados bibliografa, assim realizamos uma busca direta por trabalhos relacionados por meio de palavras-chave. Para tanto, utilizaremos como principais fontes de pesquisa as seguintes bases indexadas: *IEEEXplore Digital Library*, *ACM Digital Library*, *Scopus*, *Science Direct*. A partir desses trabalhos encontrados, buscamos citações e referências a eles até chegar a uma abrangência considerada satisfatória que cobria os trabalhos referentes a conhecimento de código-fonte de software, sistemas de recomendação de especialista em código, repositórios sociais de código-fonte.

No Anexo A está descrito o protocolo de pesquisa criado durante processo de desenvolvimento dessa pesquisa.

3.1.2 Mapeamento de métricas

Esta fase teve por objetivo realizar o mapeamento das métricas de identificação de especialista em código-fonte. O mapeamento foi produzido a partir da análise dos artigos selecionados na etapa de revisão bibliográfica elaborado na etapa anterior. 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 em código-fonte. Para podermos entender melhor estas semelhanças e divergências, mapeamos algumas perguntas que poderiam nos auxiliar a entender melhor quais dados mais utilizadas por cada uma das métricas e como elas poderiam ser agrupadas.

3.1.3 Construção da ferramenta

Desenvolvemos uma ferramenta capaz de extrai commits de projetos originados dos repositórios de código-fonte gerenciados pelo VCS Git, calcula as métricas de número de *commit*,

[1] Revisão Sistemática da Literatura (RSL), ou simplesmente Revisão Sistemática (RS), é um dos principais meios para sumarizar evidências de pesquisa.

número de linhas e grau de conhecimento para identificar o especialista no código-fonte baseado na convergência dessas métricas. A ferramenta foi desenvolvida em Python e pode extrair commits em projetos escritos em Java, PHP, JavaScript ou Python.

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 indicação de especialistas em código-fonte de software apresentado nesse projeto está baseado na análise das três métricas apresentadas anteriormente. A partir dessas métricas, e exatamente como elas foram definidas pelos autores, a ferramenta apresenta uma convergência dessas indicações para os desenvolvedores extraídos para cada uma delas.

Neste primeiro momento, o objetivo da ferramenta foi de apenas identifica especialistas em código-fonte com base em cada uma dessas métricas(número de *commit*, número de linhas e grau de conhecimento) e sua convergência.

3.1.4 Testa da ferramenta

Em paralelo ao desenvolvimento da ferramenta de identificação de especialista de código-fonte, iniciou-se o processo de testes. Para os testes, foi desenvolvido outra ferramenta denominada de gerador de repositório de código-fonte fictício(descrito o funcionamento no Apêndice A), para que permitisse controlar e simular o desenvolvimento de um projeto de software fictício e assim validar se as interações dos desenvolvedores com o código estavam sendo computadas corretamente pela ferramenta a fim de gerar os resultados corretamente.

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. 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 foram previamente cobertos e quando todas as métricas foram validadas manualmente.

A fim de melhor a legibilidade do documento relatamos e detalhamos os testes realizados no Apêndice B desde trabalho.

3.1.5 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 Fintech[2], disponibilizou um de seus projetos para que fosse validada a ferramenta, assim como também disponibilizou um dos seus times de desenvolvimento para responder a um questionários para entendimento e definição do oraculo.

Com isso, o primeiro passo para definição do oráculo foi entender quais entidades tinham mais relevância no projeto em questão, para ser possível validar a recomendação por meio da ferramenta. A escolha das entidades está descrito na seção 4.0.3.

Após definidas as entidades, foi enviado um questionário para os desenvolvedores responderem (em uma escala de 0 a 10) com o conhecimento que cada desenvolvedor considera que têm em relação às entidades. As características dos respondentes são: 7 desenvolvedores, destes 4 desenvolvedores sêniores, 2 plenos e 1 júnior. Com estas respostas computadas, nós definimos o oráculo para cada entidade e assim comparamos com os resultados de cada métrica extraída pela ferramenta. Todo esse processo avaliativo está detalhado no Capítulo 4 deste trabalho.

[2] Fintech é um termo que surgiu da união das palavras *financial* e *technology*, se tratando da tecnologia e inovação aplicadas na solução de serviços financeiros.

A empresa atua no mercado brasileiro, com mais de mil funcionários, na qual utiliza as tecnologias Java e Kotlin como linguagens de programação base nos seus projetos.

4 PROTÓTIPO DA FERRAMENTA

Neste Capítulo, é apresentada uma visão geral do protótipo da ferramenta desenvolvida[1] para a identificação de especialista de código-fonte.

Inicialmente, faz-se necessário o entendimento do escopo do protótipo da ferramenta definido em: implementação de uma ferramenta capaz de extrair dados contidos em repositórios de código-fonte, onde a ferramenta construída auxilie na identificação dos possíveis especialistas de código-fonte em sistemas. Este protótipo considera o repositório do ponto de vista do sistema de controle de versão (Git).

O protótipo da ferramenta desenvolvida deve conseguir extrair dados de projetos com foco em dados sobre os *commits* do projeto. A extração dos dados será feita de projetos de código armazenados em repositórios Git, sejam eles privados ou *open-source*. 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 dos dados para a identificação de especialistas de um 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.

4.0.1 Processo para extração dos dados

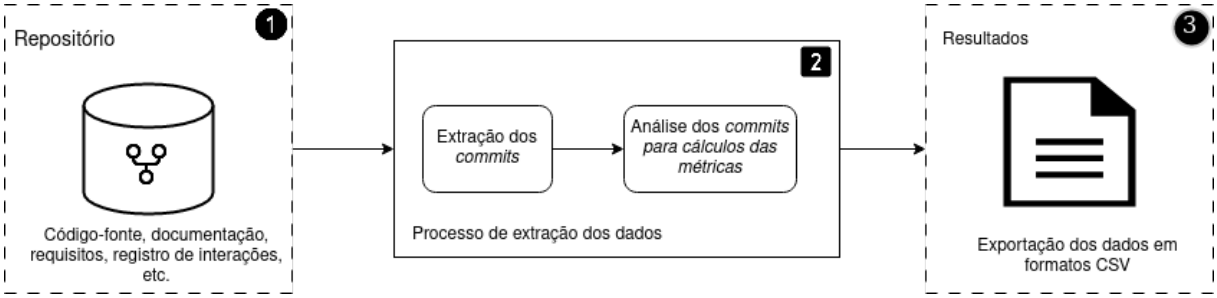
Para ser possível processar as medidas baseadas nas métricas por números de *commits*, por números de linhas modificadas e por grau de conhecimento, e extrair dados a partir delas, definimos um processo para tal. Este processo permite recuperar os *commits* e extração dos dados de interesse em repositórios Git. O processo segue o fluxo de escrita do código-fonte e da estrutura de armazenamento das informações nos repositórios. A representação deste processo pode ser visto na Figura 6.

A abordagem usada na implementação da ferramenta gera medidas baseadas nas métricas por números de *commits*, por números de linhas modificadas e por grau de conhecimento, cada uma delas podendo recomendar diferentes especialistas de código *software*.

Na Figura 6 é possível observar de forma esquematizada o processo de execução da ferramenta, logo não reflete toda a dinâmica pela qual os dados são processados na ferramenta. O exemplo mostra o processo de coleta de dados de extração de um repositório. A primeira etapa é a seleção do repositório, na segunda etapa coleta de dados, que consiste em extrai os

[1] https://github.com/WemersonThayne/expert-extractor

Figura 6 – Processo de execução da ferramenta.



Fonte: autoria própria.

valores contabilizados pelas métricas de, números de *commits*, números de linhas e grau de autoria, apresentada na seção 4.0.1.1. A terceira etapa envolve a sumarização de informações das contribuições para a granularidade de entidade, arquivo ou projeto contidos no repositório de controle de origem. Por fim, o relatório contendo as métricas de conhecimento de código é gerado para análises manuais.

4.0.1.1 Extrator das métricas

O Extrator consiste num conjunto de artefatos de software para extração a partir do código-fonte, que realiza os cálculos das métricas de autoria e computa da convergência entre as métricas. É composto de duas principais aplicações, o extrator de contribuições e o gerador das métricas; e definido um formato de armazenamento em CSV para as informações das indicações do possível especialista indicado por cada métrica. A Figura 7 apresenta uma visão geral da interação dos artefatos de *software*.

Figura 7 – Etapas de funcionamento da ferramenta de extração e coleta.

Fonte: autoria própria.

- Etapa 1) Clonagem das informações dos repositórios remotos para uma pasta local;
- Etapa 2) O processo faz uma busca por todos os *commits* para identificar aqueles que realizaram alguma interação com o código-fonte. A partir dessa identificação, são salvos os dados em uma tupla, que será posteriormente utilizada para o cálculo das métricas. A Tabela 4 contém as informações que a tupla armazena.

  No processo de extração dos contribuidores é necessário fazer uma normalização dos contribuidores do projeto para validar os desenvolvedores do projeto. Pois, é comum um mesmo desenvolvedor utilizar mais de um identificador para registrar suas respectivas contribuições sobre o código-fonte. Na ferramenta, consideramos o e-mail utilizado como chave das credenciais dos integrantes do projeto, por exemplo, um contribuidor que utilize jose.aldo e depois mudou o nome para aldo.jose, mas o e-mail continuar o mesmo terá suas contribuições identificadas por jose.aldo@email.com;
- Etapa 3) Nesta etapa, com a tupla definida, e com todos os *commits* registrados, é acionado o extrator que inicia o processo de extração dos especialistas por meio das métricas escolhidas para análise do projeto, no qual usam os componentes para gerar como saída uma matriz de frequência de contribuições por arquivos do sistema para cada tipo de medida;
- Etapa 4) Exporta os resultados do extrator para um arquivo de extensão CSV;
- Etapa 5) Identificação dos possíveis especialistas manualmente, comparando visualmente o valor de cada uma das métricas para cada uma das entidades por cada tipo de abordagem.

Tabela 4 – Dados extraídos apresentados resumidamente

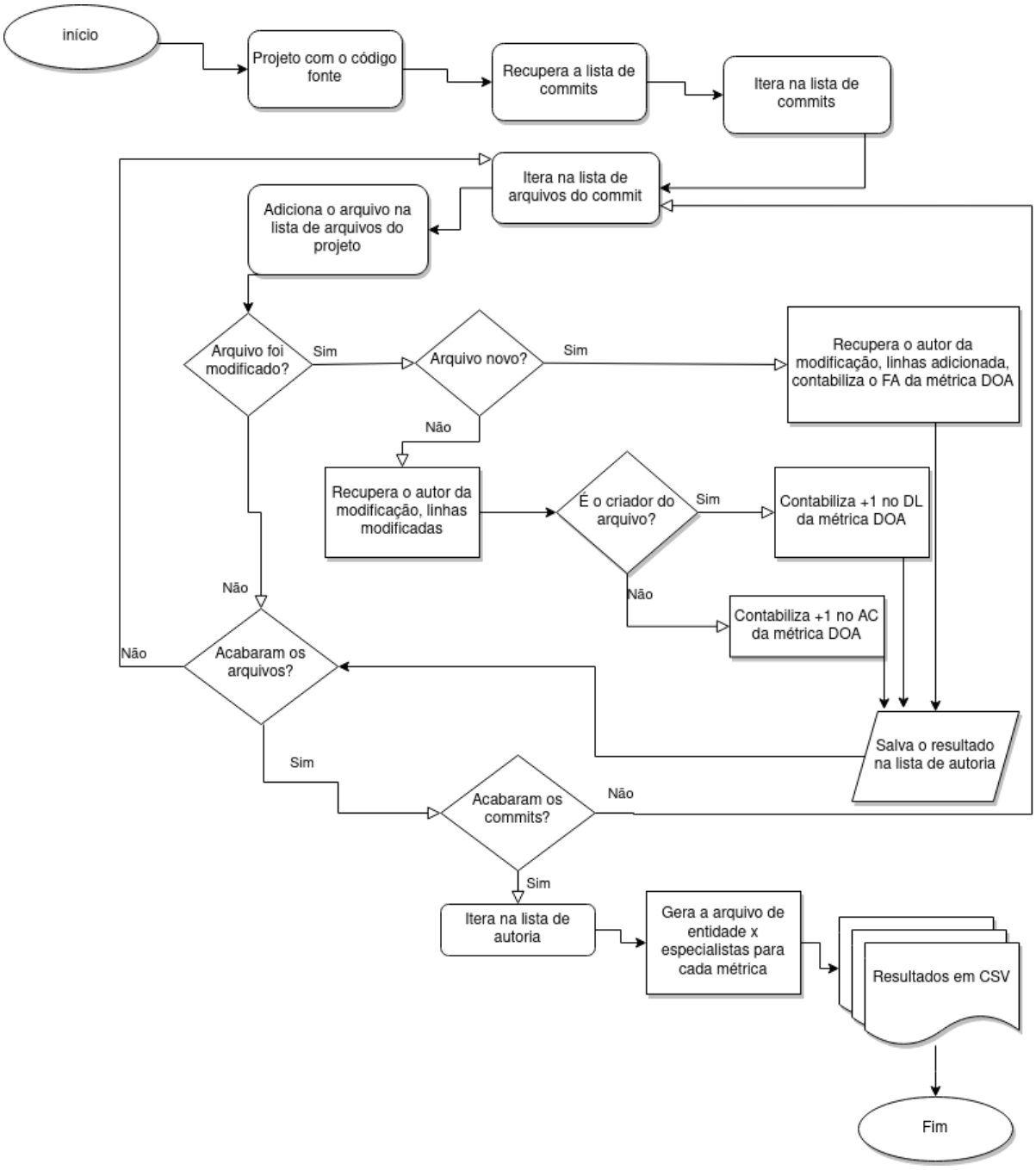
Dados extraídos	**Descrição**
Desenvolvedor	O nome do desenvolvedor que realizou o commit
E-mail	E-mail do desenvolvedor que realizou o commit
Commit	O identificador único hash correspondente do commit
Tipo do commit	Identificação do tipo de modificação do commit, determinando se é um commit de adição de linhas, remoção de linhas ou deleção de arquivo
Quantidade de linhas adicionadas	A quantidade de linha adicionadas pelo desenvolvedor no commit
Quantidade de linhas removidas	A quantidade de linha removidas pelo desenvolvedor no commit
Arquivos modificados no commit	Lista de arquivos modificados ou criados no commit
Data do commit	Data e hora em que o commit foi realizado

Fonte: autoria própria.

4.0.2 Descrição do protótipo da ferramenta

O algoritmo que representa o funcionamento da ferramenta de extração é representado pela Figura 8.

Figura 8 – Fluxograma do algoritmo da ferramenta.



Fonte: autoria própria.

A descrição das etapas do algoritmo:

- **Projeto com o código-fonte:** É o local onde está localizado o projeto com o Git no computador.

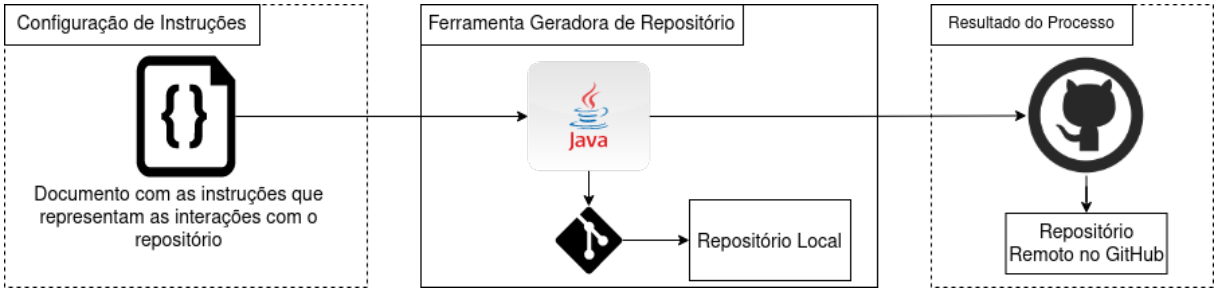
- **Recupera a lista de *commits*:** Usando o *git log* que é um comando do Git, responsável por recuperar o histórico de *commits* de um repositório.
- **Itera na lista de *commits*:** Logo após se obter a lista de *commits*, é possível criar um laço de repetição para percorrer a lista do elemento *commit* de forma cronológica, ou seja, do mais velho, até os mais recentes.
- **Itera na lista de arquivos do *commit*:** Ao iterar na lista de *commits*, se obtém um *commit* por vez. Dentro dos *commits* existem arquivos, que devem ser percorridos para as informações serem armazenadas.
- **Condicional: O arquivo foi adicionado?** Caso o arquivo seja novo, caso contrário, indica que o arquivo foi removido, logo não há contabilização das métricas para aquele *commit*.
- **Condicional: Arquivo novo?** Caso o arquivo seja novo, ou seja, criado naquele *commit*, recupera o autor da modificação, linhas adicionadas e contabiliza o FA da métrica DOA. Caso contrário, recupera o autor e linhas modificadas e encaminha para uma nova condicional.
- **Condicional: É o criador do arquivo?** Neste passo o algoritmo consulta a lista de autorias, e recupera quem foi o primeiro autor no arquivo, caso, sim, ele contabiliza "+1"no fator DL da métrica DOA. Caso não contabiliza um AC para o autor da autoria para o arquivo em questão.
- **Condicional: Acabaram os arquivos?** Caso já tenha passado por todos os arquivos do commit, o laço irá para o próximo commit caso ele existir. *commit* **Condicional: Acabaram os *commits*?** Caso já tenha passado por todos os *commits*, o laço de repetição irá se encerrar.
- **Iterar na lista de autoria:** Essa iteração ocorre sobre a lista de autoria de todos os *commits*, que contém as informações de cada autoria realizada nos arquivos que existem no projeto, ela armazena a quantidade de autoria dos autores para cada uma das métricas.
- **Resultados:** Os resultados consistem na geração para cada métrica a matriz de entidade especialista(apresentado na seção 5.2.1) e posteriormente exportado para um arquivo da extensão CSV.

4.0.2.1 Gerador de repositório fictícios

A princípio se faz necessário pontuar sobre a importância e necessidade de um repositório fictício na validação de ferramentas que investigam autoria de código. Ele permite inicialmente testar a ferramenta em questão, e também permite a simular a dinâmica em um repositório real de código. Como objetivo específico desta pesquisa, observou a necessidade de criar uma ferramenta geradora de repositórios sintéticos para *benchmarking* dos métodos e ferramentas

que investigam autoria de código. O gerador de repositório fictícios é composto por artefatos de software que manipulam o sistema de versionamento *git*, gerando arquivos e interações no código-fonte a partir de instruções fornecidas por meio de um arquivo de configuração, assim simulando a realização de *commits* no repositório gerado. A figura 9 mostra o funcionamento da ferramenta.

Figura 9 – Processo de geração de repositório fictícios.



Fonte: autoria própria.

O funcionamento da ferramenta inicia-se pela leitura das informações contidas no arquivo de instruções como, por exemplo, nome do projeto, pasta, usuário, as simulações de modificações propriamente ditas. Apresentamos como mais detalhes a construção do arquivo no apêndice A. Após a leitura, as instruções são transformadas em objetos java e manipulados pela biblioteca *processbuilder*[2]. para operar os comandos do VCS *git* e assim gerando localmente o histórico de alterações e interações dos desenvolvedores fictícios com os códigos fonte. Por fim, usando a API *Rest(Representational State Transfer)* da plataforma GitHub criamos o repositório remoto para armazenar todo o repositório gerado a partir do arquivo e em seguida é realizado o envio de todo o histórico e arquivos para o repositório criado.

Abaixo é apresentado a sequência de etapas que o gerador é executado:

1. Ler o arquivo de instruções com todas as informações e simulações dos *commits*;
2. Manipula as instruções transformando em objetos java e gerando os *commits* e mudanças nos arquivos, simulando assim uma interação real com os arquivos de código-fonte;
3. Gera o repositório local usando os recurso do próprio git através da biblioteca *processbuilder*;
4. Cria o repositório na plataforma GitHub;
5. Realiza o envio do repositório local para o criado na fase anterior.

[2] Página da biblioteca: <https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html>. Acesso em 3 janeiro 2023

4.0.3 Definição do oráculo para validação

Para podermos validar as métricas calculadas pela ferramenta, comparando os resultados por ela apresentados com a realidade, tivemos de definir um oráculo para validação. No qual entramos em contato com a gestão de um time de desenvolvimento, e em conversar via e-mail foi permitido a construção do oráculo com auxílio do time de desenvolvedores na condição de nenhuma informação sensíveis ao negócio fosse exposta, para isso usaremos de um ofuscamentos nos dados de: nome de entidades, desenvolvedores e projeto.

A fim de chegar à definição de um oráculo, visamos entender junto aos desenvolvedores do projeto PROJETO_1(Nome fictício solicitado pela empresa) de uma empresa brasileira qual era o nível de conhecimento do time de desenvolvimento sobre o 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 C.

O time de desenvolvimento é composto por 9 pessoas(destas 7 são desenvolvedores), porém as medidas de conhecimentos só foram computadas para os desenvolvedores de software, já que apenas estes podem ser avaliados por meio da ferramenta de indicação de especialistas de código-fonte, a partir dos dados extraídos do Git. Para cada membro do time, pedimos para eles quantificarem o seu conhecimento para a dada entidade que compõe o projeto em uma escala de zero a dez. Os desenvolvedores deste projeto foram nomeados de Desenvolvedor_1 a Desenvolvedor_7. Cada resposta foi única e não teve nenhuma interferência dos outros desenvolvedores sobre cada uma das notas dadas. Estas respostas podem ser visualizada na Tabela 5.

A seleção da amostra de entidades do projeto iniciou na quantificação do total de entidades de código-fonte que compõe o PROJETO_1. Foi contabilizado 133 entidades, destas selecionamos 20 entidades, representando 15% do total. Para determinar a amostra das entidades utilizamos o método de classificação de *quartis* baseados na quantidade de modificações recebidas, assim classificando em 4 grupos: as entidades com um percentual de 0% a 25%, de 26% a 50%, de 51% a 75% e maior de 75%. Depois da classificação das entidades pelos *quartis*, selecionamos de forma aleatória 5 entidades de cada grupo a serem submetidas ao *survey*.

Tabela 5 – Respostas dos desenvolvedores para definição do oráculo

ENTIDADE	DEV_1	DEV_2	DEV_3	DEV_4	DEV_5	DEV_6	DEV_7
ENTIDADE_1	0	3	1	8	9	8	6
ENTIDADE_2	0	10	0	0	7	7	6
ENTIDADE_3	0	6	0	6	10	8	2
ENTIDADE_4	10	7	7	5	7	7	4
ENTIDADE_5	7	9	0	8	8	3	5
ENTIDADE_6	3	6	0	9	7	8	8
ENTIDADE_7	8	3	0	7	9	7	6
ENTIDADE_8	5	5	5	5	9	7	0
ENTIDADE_9	1	7	5	5	10	6	7
ENTIDADE_10	7	6	2	9	4	6	7
ENTIDADE_11	5	10	9	7	8	9	3
ENTIDADE_12	10	6	7	9	8	10	5
ENTIDADE_13	8	3	2	2	7	7	1
ENTIDADE_14	8	7	5	5	6	8	5
ENTIDADE_15	8	7	7	6	9	9	1
ENTIDADE_16	7	9	1	3	7	6	8
ENTIDADE_17	9	7	7	6	6	9	6
ENTIDADE_18	8	9	4	4	7	7	0
ENTIDADE_19	7	10	7	7	8	8	3
ENTIDADE_20	8	3	0	9	9	7	3

Fonte: autoria própria.

A partir das notas atribuídas por cada um dos desenvolvedores as entidades, criamos um ranking de notas para cada entidade e para determinar o desenvolvedor especialista com base na maior nota atribuída. Desta forma, o oráculo definido para o projeto é um ranking apresentado na Tabela 6.

Tabela 6 – Oráculo do PROJETO_1

ENTIDADE	DESENVOLVEDOR
ENTIDADE_1	DEV_5
ENTIDADE_2	DEV_2
ENTIDADE_3	DEV_5
ENTIDADE_4	DEV_1
ENTIDADE_5	DEV_2
ENTIDADE_6	DEV_4
ENTIDADE_7	DEV_5
ENTIDADE_8	DEV_5
ENTIDADE_9	DEV_5
ENTIDADE_10	DEV_4

ENTIDADE	DESENVOLVEDOR
ENTIDADE_11	DEV_2
ENTIDADE_12	DEV_1
ENTIDADE_13	DEV_1
ENTIDADE_14	DEV_1
ENTIDADE_15	DEV_2
ENTIDADE_16	DEV_1
ENTIDADE_17	DEV_1
ENTIDADE_18	DEV_2
ENTIDADE_19	DEV_2
ENTIDADE_20	DEV_4

Fonte: autoria própria.

4.0.4 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 o protótipo computou os resultados com boa acurácia.

Os resultados de cada métrica são exibidos na Tabela 7. 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.

Tabela 7 – Resultados da execução do protótipo para o PROJETO_1

Entidade	**Métrica *commit***	**Métrica LOC**	**Métrica DOA**
ENTIDADE_1	DEV_5	DEV_5	DEV_5
ENTIDADE_2	DEV_2	DEV_2	DEV_2
ENTIDADE_3	DEV_5	DEV_5	DEV_5
ENTIDADE_4	DEV_1	DEV_1	DEV_1
ENTIDADE_5	DEV_2	DEV_2	DEV_2
ENTIDADE_6	DEV_6	DEV_6	DEV_6
ENTIDADE_7	DEV_5	DEV_5	DEV_4
ENTIDADE_8	DEV_5	DEV_5	DEV_5
ENTIDADE_9	DEV_5	DEV_5	DEV_7
ENTIDADE_10	DEV_5	DEV_5	DEV_5
ENTIDADE_11	DEV_1	DEV_2	DEV_1
ENTIDADE_12	DEV_1	DEV_1	DEV_1
ENTIDADE_13	DEV_1	DEV_1	DEV_1
ENTIDADE_14	DEV_1	DEV_1	DEV_1
ENTIDADE_15	DEV_2	DEV_7	DEV_1
ENTIDADE_16	DEV_1	DEV_5	DEV_2
ENTIDADE_17	DEV_1	DEV_2	DEV_1
ENTIDADE_18	DEV_1	DEV_2	DEV_1
ENTIDADE_19	DEV_1	DEV_2	DEV_7
ENTIDADE_20	DEV_5	DEV_6	DEV_6

Fonte: autoria própria.

A assertividade do protótipo em relação ao oráculo é razoavelmente alta. A precisão dos *commits* foi de 65%, o que significa que o protótipo foi bastante preciso ao prever os *commits*. A precisão do LOC foi de 55%, o que significa que o protótipo foi possivelmente preciso ao prever o LOC. Por fim, a precisão da DOA foi de 70%, o que significa que o protótipo foi bastante

preciso ao prever a DOA. No entanto, é importante notar que o protótipo divergiu do oráculo para as três métricas, sendo 35%, 45% e 30% para os *commits*, LOC e DOA, respectivamente. Isso mostra que, embora o protótipo tenha sido possivelmente preciso, ainda há um certo grau de divergência entre o protótipo e o oráculo.

Pode-se destacar que o projeto tem 133 entidades, destas apenas 20 foram selecionados para construção do oráculo. Com base nisto é importante ressaltar, que a construção de um oráculo ele tem um alto custo de tempo e esforço, uma vez que é necessário que os envolvidos no projeto pare suas atividades e respondam o *survey* para criação do oráculo.

Essa dificuldade se mostra em relação à falta de representatividade de algumas entidades na construção do oráculo. Porque se tratavam de várias entidades. O projeto teve como um dos principais objetivos a criação de um modelo que pudesse ser aplicado na verificação da taxa de assertividade do protótipo. Portanto, ainda que os resultados não tenham sido totalmente satisfatórios, as conclusões obtidas servirão como base para a criação de um modelo de oráculo mais abrangente em trabalhos futuros.

5 AVALIAÇÃO DAS MÉTRICAS PARA IDENTIFICAÇÃO DE ESPECIALISTAS

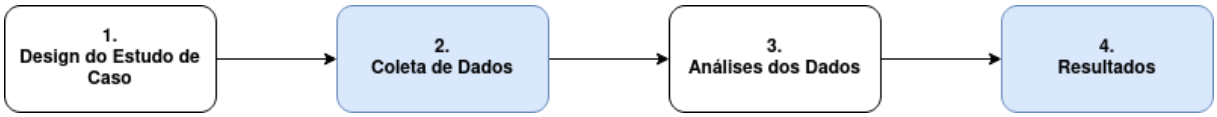
Neste capítulo, é apresentado o processo de avaliação das métricas para a identificação de especialistas. Esse processo avaliativo foi realizado usando o protótipo da ferramenta desenvolvida. Para tanto, foram analisadas projetos de software aberto disponíveis no GitHub.

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

5.1 Design da avaliação das métricas

As fases incluídas para a avaliação das métricas escolhidas neste trabalho são apresentadas a seguir, com uma adaptação dos passos definidos por Malhotra (2016). A Figura 10 apresenta as quarto fases que foi estruturado o estudo.

Figura 10 – Fases da avaliação das métricas a partir dos repositórios *open source*.



Fonte: autoria própria.

1. Design do estudo de caso: Definido as questões de pesquisas, apresentadas na seção 1.3, com intuito de guiar as investigações que serão realizadas nos repositórios de código. No apêndice D contém o nome de todos os repositórios analisados.

2. Coleta de dados: Estruturação dos dados alvos a serem coletados, detalhado na seção 5.1.2. Nesta etapa são selecionados os projetos de códigos abertos, os critérios utilizados na escolha dos projetos está descrito na seção 5.1.2. Conforme observou Zhang et al. (2018) que o código aberto mais popular estão nas plataformas Apache, GitHub e SourceForge. São bastante utilizados nas abordagens e ferramentas de apoio para coleta/processamento/análise de dados no ESE.

3. Análises dos dados: Nesta etapa são realizadas as extrações das contribuições confirmadas através dos registros de *commits* e identificação dos desenvolvedores dos *commits* extraído, computando a eles os valores identificados por cada métrica(*Commit* , LoC e DOA), conforme na seção 5.2.

4. Resultados: Geração de uma tabela de convergência entre as métricas com cada indicação do possível especialista da entidade de código. Disponíveis na seção 5.3.

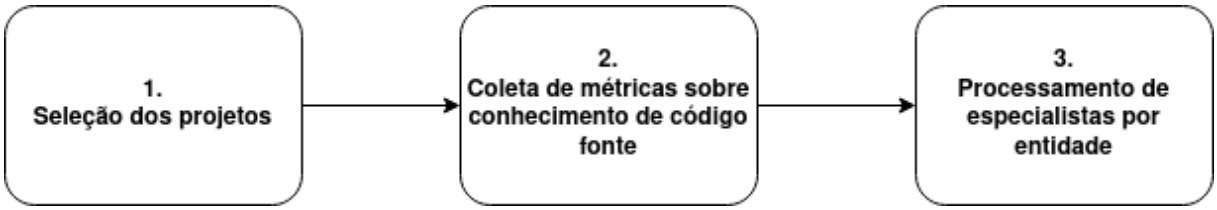
5.1.1 Questões de Pesquisa

Todas as medidas indiretas são úteis para complementar algumas situações em que uma abordagem que modela conhecimento não conseguem capturar(SANTOS; GUERRERO; FIGUEIREDO, 2015). Há também situações onde as abordagens convergem para o mesmo especialista. Existe ainda a possibilidade de que trechos de código sejam órfãos (sem especialistas). Para tentar quantificar essas situações, definimos as seguintes Questões de Pesquisa (QP) apresentadas na seção 1.3:

5.1.2 Coleta de dados

Para responder às QPs realizamos uma avaliação em 100 projetos Java *open source* do repositório Github, divididos em três etapas apresentados na Figura 11.

Figura 11 – Etapas da coleta de dados



Fonte: autoria própria.

1. Seleção dos projetos:
   Para a seleção desses projetos, foram utilizados os seguintes critérios:

   a) Projetos. Que tinham popularidade na plataforma GitHub através da ferramenta *trending*.

   b) Linguagem dos projetos. Para essa avaliação, nós usamos apenas projetos em Java, embora a ferramenta possa ser aplicada para qualquer linguagem.

   c) Quantidade de desenvolvedores ativos. Foram considerados para a análise projetos que tenham mais de 5 desenvolvedores contribuindo ativamente com o projeto.

   d) Número de commits. Consideraram-se apenas projetos com no mínimo de 50 *commits*.

   e) A lista dos projetos pode ser visualizada no apêndice D

2. Coleta de métricas sobre conhecimento de código-fonte:
   Utilizamos o ferramental próprio descrito na seção 4.0.1.1, para realizar o processo de contagem que ocorre por meio das interações com o código, definido como *commit*. O extrator recupera todos os *commits* mantido no versionamento Git e processa cada um deles. Por fim, gera os valores das métricas (por *commit*, por LoC, e por DOA) para cada um dos contribuidores;

3. Processamento de especialistas por entidade:
   Para cada um dos projetos, as abordagens indicam qual desenvolvedor é o especialista para cada uma dos arquivos. A partir das métricas foram feitas definições operacionais de métrica, pois é importante garantir que as métricas sejam coletadas de uma forma consistente para serem analisadas e esses dados se mantenham consistentes(ROCHA; SOUZA; BARCELLOS, 2012).

5.2 Análise dos dados

5.2.1 Matriz de especialidade

Após o processo de extrações das contribuições, foi gerado um arquivo contendo os possíveis especialistas de código-fonte para cada entidade, representado por uma Matriz de Entidade Especialista(MEE), sendo uma matriz de **M** linhas onde cada linha representa uma Entidade por **N** Especialistas, onde o conteúdo da célula é o valor da medida para m-ésima Entidade dado um n-ésimo desenvolvedor, artefato gerado durante o desenvolvimento do trabalho. Importante destacar para cada métrica gera-se uma tabela MEE, conforme apresenta a Tabela 8 um exemplo de extração.

A primeira coluna **Entidade** representa um arquivo-fonte, o nome do arquivo físico. As demais colunas da primeira linha, apresentam os desenvolvedores(possíveis **especialistas**), que realizaram alguma alteração no código e submeteram ela para o repositório assim, contabilizando um commit a cada interação. Exemplo **Especialista 1**, **Especialista 2** e **Especialista N**.

Tabela 8 – Tabela entidade especialista(MEE).

Entidade	**Especialista 1**	**Especialista 2**	**Especialista N**
entidadeA	X	Z	N
entidadeB	Y	Y	N
entidadeC	Z	X	N

Fonte: autoria própria.

De modo a descrever a Tabela 8, dispomos de um cenário fictício de extração da métrica de *commits*, contido na Tabela 9. Um dado projeto tem 3 desenvolvedores(**João, Ana e Pedro**) que contribuíram com as entidades(**A.java, B.java e C.java**). Interpretando a saída da tabela de entidade x especialista, detalhemos a interação de cada entidade de código.

Tomando como ponto de partida a entidade **A.java**, recebeu um total de 10 *commits* ao longo do tempo, deste observamos que o desenvolvedor **João** realizou 6 *commits*, **Ana** no que lhe concerne, fez 3 e **Pedro** apenas 1 do total de 10. Logo, com base na abordagem por número de *commits*, o especialista é **João**.

Semelhantemente, a entidade **B.java**, obteve um total de 7 *commits*, observando os valores de cada desenvolvedor, teríamos a indicação do desenvolvedor **Pedro** como especialista

por produzir 4 alterações no código. Já a entidade **C.java** indica a desenvolvedora **Ana** como especialista.

Tabela 9 – Exemplo de uma extração da métrica de *commits*.

Entidade	**João**	**Ana**	**Pedro**
A.java	6	3	1
B.java	2	1	4
C.java	1	3	0

Fonte: autoria própria.

5.2.2 Convergência indicativa de especialista

Com a intenção de avaliarmos as abordagens e a ferramenta de extração de métricas geradas, se fez necessário uma avaliação das medidas extraídas para entender o comportamento das abordagens em alguns cenários consoantes ao desenvolvimento de *software*. Para que pudéssemos responder à motivação do trabalho em: quantas entidades em um projeto não possuem especialistas, quantas entidades tem indicadas diferentes desenvolvedores pelas diferentes abordagens, realizamos uma análise de convergência entre as abordagens.

A fim de verificarmos a ocorrência de múltiplas indicações do mesmo especialista na entidade para cada uma das abordagens, pode auxiliar na identificação do especialista de código, uma vez que cada técnica observa um aspecto diferente na interação com o código.

A abordagem de *commit* apenas contabiliza a mudança submetida ao repositório independente de qual seja, LoC determina o número de linhas modificadas sendo mais precisa na medição, porém se discriminar se realmente é uma mudança no conteúdo da linha ou uma alteração de organização de código, por exemplo, espaçamento entre os caracteres. Já a DOA combina a interações entre código e desenvolvedores, com objetivo de cobrir os deficits das outras abordagens. Então a convergência pode nos trazer uma nova visão sobre o entendimento do código.

Importante definir semanticamente o que é divergência e convergência no contexto desta pesquisa, divergência é quando todas as métricas apontam diferentes especialistas, por sua vez convergência é quando todas as métricas apontam um único especialista, existe também uma convergência parcial, é quando duas quaisquer métricas concordam entre si com o mesmo especialista.

Tabela 10 – Convergência entre as Abordagens

Entidade	**Métrica Commit**	**Métrica LoC**	**Métrica DOA**	**Conv. de 2 para 3**	**Conv. de 3 para 3**	**Divergência**
entidadeA	especialista 1	especialista 2	especialista 1	X	O	N
entidadeB	especialista 2	especialista 2	especialista 1	X	O	N
entidadeC	especialista 1	especialista 1	especialista 1	O	X	N
entidadeD	especialista 1	especialista 2	especialista 3	O	O	S
			Total	2	1	1

Fonte: autoria própria.

A saída da convergência indicativas de especialista e exibida na Tabela 10. Descrevendo os campos que compõe a tabela:

- **Entidade:** Nome do arquivo;
- **Abordagem Commit:** Desenvolvedor indicado como especialista pela métrica de *commit*;
- **Abordagem LoC:** Desenvolvedor indicado como especialista pela métrica de LoC;
- **Abordagem DOA:** Desenvolvedor indicado como especialista pela métrica de DOA;
- **Convergência de 2 para 3:** Convergência de duas abordagens indicando o mesmo especialista, isto significa que, das 3 métricas duas delas apontam para o mesmo desenvolvedor;
- **Convergência de 3 para 3:** Convergência das três abordagens indicando o mesmo especialista, logo, apontam para o mesmo desenvolvedor em todas as indicações;
- **Divergência:** As 3 métricas apontaram desenvolvedores diferentes como especialista, assim cada abordagem identificou um desenvolvedor especialista da entidade em questão.

Cenários possíveis que podem ocorrer na convergência para cada entidade: (1) as 3 métricas indicarem o mesmo especialista, determinando assim uma convergência total; (2) apenas duas métricas apontam para o mesmo especialista, uma convergência parcial;(3) uma total divergência, cada medida recomendam diferentes especialistas em código. Exemplificando os cenários acima geremos uma extração fictícia em um projeto, ver a Tabela 11.

Tabela 11 – Convergência indicativa fictícia

Entidade	**Métrica Commit**	**Métrica LoC**	**Métrica DOA**	**Conv. de 2 para 3**	**Conv. de 3 para 3**	**Divergência**
A.java	Tereza	Tereza	Tereza	O	X	N
B.java	Ester	Bruce	Ester	X	O	N
C.java	Bruce	Ester	Tereza	O	O	S
			Total	1	1	1

Fonte: autoria própria.

5.3 Resultados

5.3.1 Sumarização dos dados

No nosso estudo de caso foram analisados mais de 500 mil de *commits* nos dos projetos, quase 23 mil desenvolvedores indicificados como contribuidores, e 256 mil entidades, e mais 172 milhões de linhas modificadas, adicionadas ou removidas. A Tabela 12 sumariza todos os dados extraídos e analisado pelo Identificador de Especialista de código-fonte. Estes dados fornecem um panorama amplo dos projetos.

Tabela 12 – Sumário dos dados extraídos e analisados

Número de Desenvolvedores	22.443
Número de *Commits*	665.257
Número de Entidades	256.174
Número de Linhas Modificadas	172.633.557
Número de Linhas Adicionadas	106.567.619
Número de Linhas Removidas	66.065.938

Fonte: autoria própria.

Os resultados apontam para a existência de uma abundância de desenvolvedores ativos, com boa desempenho na realização de *commits* e estabilidade dos projetos. A média de *commits* por desenvolvedor foi de aproximadamente 4,4 por mês e a média de *commits* por mês foi de cerca de 5 mil. A média de linhas de código modificadas foi de aproximadamente 170 milhões.

Além disso, foi possível observar a existência de um grupo de desenvolvedores com alto nível de atividade, responsáveis por grandes partes dos *commits*. Estes desenvolvedores tiveram um papel significativo na manutenção dos projetos, uma vez que realizaram grande parte das modificações nos códigos-fonte. Em suma, é possível afirmar que os projetos se encontram bem estabelecidos, com desenvolvedores ativos e uma contribuição significativa dos desenvolvedores experientes. Estes dados fornecem uma base para a realização do estudo.

As extensões permitidas foram: **.java, .py .html .css .js .xml .c .clj .cpp .sql .ymal yml .sh**. Do total de 256.174 entidades, 62.315 foram descartadas, restando 193.859, representando **75,67%** das entidades dos projetos identificadas como entidades de código-fonte. No apêndice E estão detalhadas as quantidades das entidades de código-fonte para cada projeto.

No geral, as entidades descartadas foram aquelas que não eram relevantes para o estudo e que estavam fora do escopo dos projetos. Por exemplo, algumas entidades de código-fonte estavam em arquivos de configuração ou arquivos de teste, e outras eram entidades de terceiros, como bibliotecas ou ferramentas externas. Além disso, algumas entidades não tinham conteúdo relevante, como arquivos vazios e linhas em branco.

5.3.2 Extrações da matriz de especialidade por métrica

A extração da métrica gera para cada uma delas uma MEE conforme apresentada na seção 5.2.1. A fim de exemplificar como o extrator da MME constrói a relação entre entidade e especialista em um cenário de extração real, usaremos como base o projeto **HikariCP**(listado no apêndice D). Do total de 124 entidades com algum *commit*, foram consideradas 112 como entidades de código-fonte, representando cerca de 90% do projeto. Selecionamos dez entidades que tiveram o maior percentual de mudanças por meio do método de classificação de *quartis*, para sintetizar o resultado da extração por *commit* na Tabela 13. No Anexo B contém todo o *datasheet* para construir a Tabela 13.

Tabela 13 – Sintetize da matriz de especialidade do projeto HikariCP pela abordagem de *Commit*

Entidade	Brett Wooldridge	Guillaume Smet	Mikhail Mazurskiy	Nitin	Johno Crawford	Nerses	Jungtaek Lim	nothing	Guido Medina	Arthur D	Stephan Schroevers	Will Vuong	Yanming Zhou
DriverDataSource.java	15	1	1	1	-	-	-	-	-	-	-	-	-
UtilityElf.java	14	-	1	8	1	1	-	1	-	-	-	-	-
PoolEntry.java	15	-	1	22	-	-	1	1	-	-	-	-	-
TestConnections.java	18	-	-	19	-	-	-	-	1	-	-	-	-
ProxyConnection.java	28	-	-	22	2	-	-	-	-	-	-	-	-
HikariDataSource.java	53	2	4	1	-	-	-	-	1	1	1	-	-
ConcurrentBag.java	74	-	3	8	-	-	-	-	-	-	-	-	-
PoolBase.java	41	-	-	56	-	-	1	-	-	-	1	2	1
HikariConfig.java	124	1	1	52	-	1	-	1	1	1	1	-	-
HikariPool.java	167	-	4	85	2	1	1	3	1	1	1	2	4

Fonte: autoria própria.

Tabela 13 mostra algumas das entidades com o maior percentual de mudanças e como elas estão relacionadas aos especialistas. O resultado da abordagem por *commit* para cada uma das entidades, linhas, para cada um dos desenvolvedores contribuintes, colunas. As células com fundo cinza indicam o maior valor, e por consequência é considerado o especialista pela abordagem por *commit*, importante destacar que os desenvolvedores que estão sem valores nas colunas indica que ele não realizou nenhuma interação com a entidade.

O resultado da extração da MME indica que Brett Wooldridge é o especialista para as entidades DriverDataSource, UtilityElf, ProxyConnection e HikariDataSource, ConcurentBag, HikariConfig e HikariPool, enquanto Nitin é o especialista para as entidades PoolEntry, Test-Connection PoolBase. Com base na extração acima, podemos afirmar que Brett Wooldridge é o especialista da maior parte das entidades contidas na tabela pela abordagem de *commit*, também podemos observar um alto percentual de desenvolvedores com uma baixo percentual de mudanças nas entidades, isso implica que em projetos *open-sources* existe uma vasta quantidade de desenvolvedores sazonais nas entidades.

Esta extração nos permite identificar quem é o especialista em cada entidade e, assim, possibilita ao projeto melhorar o conhecimento sobre os seus desenvolvedores e como eles contribuem. Isso também pode servir para identificar erros mais rápido e identificar quem é o melhor especialista para determinadas tarefas.

5.3.3 Análise sobre a convergência das abordagens

A convergência das três abordagens indicando o mesmo especialista foi de 50,15%, isso para todos os projetos considerando as entidades ditas como válidas. Em números absolutos, representam, das 193.859 entidades, 97.220 apontam que o apenas um desenvolvedor é o especialista dentre os demais autores do código-fonte do projeto. No repositório[1] apresenta um *datasheet* com todas as extrações dos 100 projetos.

Durante o processo de análise foi observado que existem algumas entidades com indicações diferentes de desenvolvedores como especialista para a mesma entidade, então uma

[1] https://github.com/WemersonThayne/datasheet-convergence-entities-100-project-open-source

divergência entre as métricas. No que lhe concerne a divergência de uma das três métricas indicando diferentes especialistas foi de 49,85%, das 193.859 entidades, 96.639 indicaram diferentes desenvolvedores como possível especialista dentre os demais participantes do projeto, desta forma convergência parcial e total de especialista. Com isso, responde-se à questão de pesquisa, **"QP1) Qual o percentual da divergência entre as métricas de autoria de código-fonte?"**. A Tabela 14 reúne os valores de cada análise realizada. No Apêndice F dispõem da relação de cada projeto com a convergência das entidades.

Tabela 14 – Convergência e divergência de especialista dos projetos

Descrição	**Valor absoluto**	**Porcentagem**
Convergência	97.220	50,15%
Divergência	96.639	49,85%
	193.859	100%

Fonte:Autoria própria.

A convergência das três abordagens foi média, mas isso pode ser explicado pela natureza dos projetos estudados. Como já mencionado, são projetos complexos e muito desenvolvedores, com inúmeras contribuições, o que dificulta estabelecer um padrão de desenvolvimento para as entidades. Além disso, os desenvolvedores podem contribuir para os projetos de várias maneiras diferentes. Alguns podem contribuir com apenas pequenas correções ou otimizações, enquanto outros podem contribuir com grandes mudanças ou implementações nos projetos. Isso pode explicar por que as três abordagens não conseguiram fornecer resultados, uma alta convergência de indicações dos especialistas.

Visando observar os 49,85% de divergências dos especialistas para as entidades de todos os projetos em um nível de número de projetos, categorizamos os projetos por faixa percentual de divergência. A Tabela 15 nos mostra que dos 100 projetos analisados, 27 projetos tem um percentual menor que 25% de divergência entre as abordagens, na faixa de 25% a 49,99% há 38 projetos, e que 35 projetos foram contabilizados com mais de 50% de divergência de especialista pelas 3 abordagens.

Tabela 15 – Categorização de divergência de especialistas nos projetos

Percentual de divergência de entidades	**# Projetos**
De 0 a 25% de Divergência	27 Projetos
Entre 25 a 49,99% de Divergência	38 Projetos
Entre 50 a 69,99% de Divergência	19 Projetos
Mais 70% de Divergência	16 Projetos

Esses resultados mostram que existe uma grande variedade de projetos, com diferentes níveis de divergência entre as abordagens. Isso significa que dependendo do projeto, algumas abordagens podem ser mais adequadas que outras. Desta forma, os resultados desta análise podem auxiliar os responsáveis por projetos *open-source* a selecionar e distribuir melhor as tarefas entre os desenvolvedores para melhorar o desempenho global do projeto.

Assim, é possível concluir que os projetos *open-source* possuem uma grande diversidade de especialistas para as entidades, sendo que o nível de divergência para um mesmo projeto pode variar conforme a abordagem utilizada para identificação do mesmo. Além disso, existe uma quantidade de desenvolvedores que apenas realizaram apenas mudanças pontuais.

Para responder as outras duas questões pesquisas, **"QP2) Convergência entre pelo menos duas métricas aumenta a assertividade da indicação de especialista?"** e **"QP3) Convergência total das três métricas aumenta a assertividade da indicação de especialista?"**, verificamos que: ao usar a convergência parcial(duas métricas indicam o mesmo especialista) o percentual de assertividade é 35% na indicação dos especialistas em comparação com o oráculo, por outro lado, usando a convergência total, ou seja, as 3 métricas apontando o mesmo desenvolvedor, aumentamos em 55% na assertividade das indicações. Portanto, o uso da convergência total aumenta significativamente a assertividade na seleção de especialistas.

5.4 Ameaças à validade

Diante dos resultados discutidos, vale ressaltar que ainda são necessários alguns avanços para a melhoria dos resultados na precisão das extrações em projetos com menores níveis de interações entre o desenvolvedor e o código-fonte. Toda a pesquisa foi realizada utilizando apenas três abordagens(*Commit*, LoC e DOA), envolvendo um projeto mantido por uma empresa e outros 100 projetos *open-source* 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.

Uma possibilidade de aperfeiçoamento seria o experimento com novas abordagens, como a análise de classes, métodos, comentários, documentações, etc. Além disso, seria interessante realizar o experimento com outros tipos de projetos, como sistemas embarcados, bibliotecas, plataformas, *frameworks*, etc. Isso permitiria avaliar a eficiência do protótipo para projetos, que possuem diferentes níveis de interação entre o desenvolvedor e o código-fonte.

Outra ameaça à validade é o oráculo utilizado. É sabido que o oráculo apresentado é limitado 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 do projeto e mudanças na dinâmica no projeto. Porém, devido à dificuldade de resposta por parte dos desenvolvedores, optamos por validar o protótipo da ferramenta por meio deste oráculo reduzido, tendo em vista o tempo e esforço do time para responder o questionário de 133 perguntas, tornado inviável a construção do mesmo, limitamos o escopo em apenas 20 perguntas sobre as entidades.

6 CONSIDERAÇÕES FINAIS

Neste trabalho, exploramos a teoria e prática de abordagens para a reconhecimento de especialistas em código-fonte de software a partir de repositórios de software. Para isso, visamos identificar as principais métricas a partir de um levantamento bibliográfico sobre identificação de especialista e áreas co-relatas, permitindo o reconhecimento das técnicas e métricas utilizadas pelos autores dos trabalhos relacionados e avaliamos a utilização destas métricas para reconhecer especialistas em código-fonte. Após a análise dos trabalhos, identificamos as principais métricas utilizadas para identificar especialistas. Essas métricas foram: (1) métricas de número de *commit*, (2) métricas de números de linhas modificadas (3) métricas baseadas em grau de conhecimento(DOA). Estas métricas foram utilizadas para identificar os especialistas em diferentes estudos apresentados na seção 2.5.

Diante da diversidade de métricas mapeadas para determinação de conhecimento em código-fonte, concluímos que elas são bastante variadas, usam diversas medidas diretas e indiretas, mas que não pode se afirmar um consenso sobre a melhor métrica para as diversas situações de determinação de conhecimento dos desenvolvedores. Uma métrica pode ser mais adequada para determinadas aplicações, enquanto outras podem ser mais adequadas para outras aplicações. Assim, é importante que os desenvolvedores e avaliadores de código-fonte entendam os princípios básicos das métricas e como elas são usadas para mensurar o conhecimento em código-fonte. Além disso, é importante que as organizações tenham uma visão geral de todos os tipos de métricas disponíveis antes de selecionar uma métrica específica para uso em seu projeto.

Esse trabalho recorre à convergência entre as abordagens para se avaliar a especialidade de código dos desenvolvedores de um projeto com base nas métricas de *commit*, LoC e DOA. O protótipo de ferramenta para Indicação de Especialista por abordagem foi desenvolvido para extração das medidas, e assim contabilizando o especialista de código conforme abordado na seção de Resultado(5.3). Para avaliarmos a ferramenta de extração criamos um gerador de repositórios fictícios(apresentado na seção 4.0.2.1) como parte dos artefatos produzidos durante o desenvolvimento desta pesquisa, no Apêndice B descrevemos o processo de validação do protótipo. Com isso, nós analisamos a especialidade dos desenvolvedores de cada projeto a partir das métricas de commit, LoC e DOA. Estes dados foram usados para aferir os resultados obtidos pelo protótipo de Indicação de Especialista. Os resultados obtidos foram positivos, ao serem confrontados com os dados reais, demonstrando que a ferramenta de extração de dados conseguiu identificar os desenvolvedores especializados em cada projeto.

Diante do exposto, percebe-se que o processo de identificação de especialistas em código-fonte 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 dos especialistas, embora observem cenários diferentes. Em suma, a abordagem proposta conseguiu identificar os desenvolvedores especializados de um projeto com base nas métricas de commit, LoC e DOA. No entanto, acredita-se que a abordagem adotada neste trabalho é uma boa prática para análise de projetos em cenários reais.

Os resultados apresentados são passíveis de melhoria, porém apontam para a viabilidade de identificar especialistas em código a partir de métricas de conhecimento de código-fonte. Embora esta área demande ainda uma exploração mais sistemática para avaliação do potencial de outras métricas e fontes de dados como, por exemplo, métrica baseadas no vocabulário do software e repositórios sociais como determinantes na identificação de especialistas, 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 código-fonte de software.

6.1 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 de quais métricas seriam utilizadas para construção do protótipo. Com inúmeros métodos para identificar especialistas das entidades, tais como: tamanho da entidade, número de contribuições, experiência no projeto, tempo de participação, número de *issues* abertas e tempo para resolução destas, vocabulário do software. Diante disso, optou-se, adotarmos apenas três delas para construção do protótipo, isso pode limitar a análise de especialista em código-fonte.

Uma limitação específica da métrica escolhida(Métrica de número de *Commit*) se um desenvolvedor retirar todo o código do projeto e realizar um *commit*, logo após ele recolocar todo o projeto novamente. Esse desenvolvedor irá deter 100% da autoria do código, retirando o código inteiro e colocando ele novamente, será como se o projeto começasse daquele ponto. Pois no *commit* anterior não existiam linhas, ou seja, o histórico foi apagado.

Outra limitação encontrada foi a abrangência da avaliação. Para cada uma das métricas, foram utilizadas projetos extraídos do GitHub. O GitHub possui um conjunto de projetos bastante abrangente, com vários tipos de linguagens de programação, mas, pelo tempo disponível para a avaliação, o estudo analisou apenas projetos na linguagem Java, sendo assim, não foi viável avaliar toda a sua base de dados.

A limitação na construção do oráculo. O oráculo foi construído com uma população com baixo número na representatividade de entidades, para uma melhor análise da assertividade do protótipo seria necessário ter um número maior de entidades, porém como destacado na seção 4.0.4 seria inviável tendo em vista o esforço do time de desenvolvimento na construção do mesmo.

Da utilização do protótipo. O protótipo usa o software de versionamento Git, se caso o projeto de software for versionado em outro sistema de versionamento não é possível utilizar o protótipo da ferramenta nesse projeto.

6.2 Trabalhos futuros

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

1. Quanto à aplicação desta ferramenta, alguns estudos futuros podem ser realizados para testar a eficácia e a precisão do protótipo de Identificação de Especialista, bem como para verificar como a ferramenta pode ser aplicada a repositórios reais e utilizada no dia-a-dia dos projetos;

2. Adicionar diferentes abordagens para a avaliação da especialidade de código dos desenvolvedores, em vez de apenas as abordagens relacionadas a métricas de *commit*, LoC e DOA;

3. Analisar outras fontes além do código-fonte para auxiliar na identificação de especialistas, como, por exemplo, repositório de *bugs*, gerenciadores de atividades, listas de discussão e documentos de especificação. Essas fontes podem ajudar a identificar quem tem mais experiência no código-fonte, portanto, quem pode ser considerado um especialista.

4. Evolução do oráculo construído neste trabalho;

5. Propor novas métricas a partir do aprendizado deste trabalho;

REFERÊNCIAS BIBLIOGRÁFICAS

ABBASI, M. U. et al. Software maintenance cost: A systematic literature review. *Information and Software Technology*, Elsevier, v. 85, p. 87–102, 2017. Citado na página 15.

ABRANTES, J.; TRAVASSOS, G. Revisão quasi-sistemática da literatura: Caracterização de métodos Ágeis de desenvolvimento de software. 01 2007. Citado 3 vezes nas páginas 18, 36 e 37.

ALALI, A.; KAGDI, H.; MALETIC, J. I. What's a typical commit? a characterization of open source software repositories. In: IEEE. *2008 16th IEEE international conference on program comprehension*. [S.l.], 2008. p. 182–191. Citado 3 vezes nas páginas 23, 25 e 26.

AVELINO, G. et al. Measuring and analyzing code authorship in 1+ 118 open source projects. *Science of Computer Programming*, Elsevier, v. 176, p. 14–32, 2019. Citado 4 vezes nas páginas 29, 31, 32 e 33.

BANAEI, M.; GRAY, D. O que os gestores de projetos de software precisam saber sobre a produtividade em engenharia de software. *Gestão de projetos: teoria e prática*, Universidade Federal de Santa Catarina, v. 8, n. 2, p. 61–75, 2014. Citado na página 16.

BENNETT, K.; RAJLICH, V. Software maintenance and evolution: a roadmap. In: . [S.l.: s.n.], 2000. p. 73–87. Citado na página 15.

BHAT, P.; ROBILLARD, M. P. Software understanding: How much time do developers spend understanding code? *Proceedings of the 38th International Conference on Software Engineering*, IEEE Press, p. 739–749, 2016. Citado na página 15.

CORLEY, C. S.; KAMMER, E. A.; KRAFT, N. A. Modeling the ownership of source code topics. In: IEEE. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. [S.l.], 2012. p. 173–182. Citado 2 vezes nas páginas 16 e 25.

CROWSTON, K.; HOWISON, J.; ANNABI, H. Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice*, Wiley Online Library, v. 11, n. 2, p. 123–148, 2006. Citado na página 26.

DANTAS, A. L.; MEIRA, S.; MEDEIROS, C. Determining Software Developers Specialization Level. *Information and Software Technology*, v. 29, n. 4, p. 541–556, jul. 2016. ISSN 0950-5849. Disponível em: <https://www.researchgate.net/publication/307716208_Determining_Software_Developers_Specialization_Level>. Citado na página 24.

DRESCH, A.; LACERDA, D. P.; JUNIOR, J. A. V. A. *Design science research: método de pesquisa para avanço da ciência e tecnologia*. [S.l.]: Bookman Editora, 2015. Citado na página 34.

FRITZ, T. et al. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM New York, NY, USA, v. 23, n. 2, p. 1–42, 2014. Citado 6 vezes nas páginas 16, 27, 28, 29, 30 e 32.

FRITZ, T. et al. A degree-of-knowledge model to capture source code familiarity. In: *2010 ACM/IEEE 32nd International Conference on Software Engineering*. [S.l.: s.n.], 2010. v. 1, p. 385–394. Citado 4 vezes nas páginas 15, 16, 26 e 31.

GAO, Y. et al. Cost of software maintenance: A systematic literature review. *Information and Software Technology*, Elsevier, v. 117, p. 106095, 2020. Citado na página 15.

GIRBA, T. et al. How developers drive software evolution. In: IEEE. *Eighth international workshop on principles of software evolution (IWPSE'05)*. [S.l.], 2005. p. 113–122. Citado 4 vezes nas páginas 20, 27, 32 e 33.

GUO, Y. et al. Identifying expert developers from open-source projects using metrics. *IEEE Transactions on Software Engineering*, IEEE, v. 44, n. 10, p. 939–954, 2018. Citado na página 17.

HAMMAD, M. et al. Mining expertise of developers from software repositories. *International Journal of Computer Applications in Technology*, Inderscience Publishers (IEL), v. 62, n. 3, p. 227–239, 2020. Citado 2 vezes nas páginas 30 e 31.

HATTORI, L. P.; LANZA, M. On the nature of commits. In: IEEE. *2008 23rd IEEE/ACM International Conference on Automated Software Engineering-Workshops*. [S.l.], 2008. p. 63–71. Citado na página 23.

HOSSAIN, M. et al. Understanding the cost of software maintenance. *Journal of Software Engineering and Applications*, HINDAWI, v. 11, n. 07, p. 260, 2018. Citado na página 15.

KITCHENHAM, B.; PICKARD, L.; PFLEEGER, S. L. Case studies for method and tool evaluation. *IEEE software*, IEEE, v. 12, n. 4, p. 52–62, 1995. Citado na página 36.

KORKMAZ, F. A framework for version control of source code. *International Journal of Software Engineering and Knowledge Engineering*, World Scientific, v. 28, n. 3, p. 393–410, 2018. Citado na página 21.

KUMAR, S.; SINGH, A.; NEVES, R. A comprehensive guide to software development expertise. *Communications of the ACM*, ACM, v. 63, n. 3, p. 76–84, 2020. Citado na página 15.

LIN, C.-C.; WENG, C.-H. Influência da curva de aprendizado nos resultados do desenvolvimento de software. *International Journal of Software Engineering and Its Applications*, InderScience Publishers, v. 11, n. 4, p. 67–80, 2017. Citado na página 16.

MAJUMDAR, R. et al. Source code management using version control system. In: IEEE. *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*. [S.l.], 2017. p. 278–281. Citado na página 23.

MALHOTRA, R. *Empirical research in software engineering: concepts, analysis, and applications*. [S.l.]: CRC press, 2016. Citado 6 vezes nas páginas 17, 21, 26, 35, 36 e 50.

MCDONALD, D. W.; ACKERMAN, M. S. Expertise recommender: a flexible recommendation system and architecture. In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. [S.l.: s.n.], 2000. p. 231–240. Citado na página 25.

MEIRELLES, P. et al. A study of the relationships between source code metrics and attractiveness in free software projects. In: IEEE. *2010 Brazilian Symposium on Software Engineering*. [S.l.], 2010. p. 11–20. Citado na página 26.

MENG, X. et al. Mining software repositories for accurate authorship. In: IEEE. *2013 IEEE international conference on software maintenance*. [S.l.], 2013. p. 250–259. Citado 2 vezes nas páginas 28 e 31.

MOCKUS, A.; HERBSLEB, J. D. Expertise browser: a quantitative approach to identifying expertise. In: IEEE. *Proceedings of the 24th international conference on software engineering. icse 2002*. [S.l.], 2002. p. 503–512. Citado 4 vezes nas páginas 25, 26, 32 e 33.

NAKAGAWA, E. Y. et al. Revisão sistemática da literatura em engenharia de software: teoria e prática. Elsevier Brasil, 2017. Citado na página 37.

OU, J. *Computing degree-of-knowledge values for a developer's workspace*. Tese (Doutorado) — University of British Columbia, 2009. Citado na página 16.

PERERA, C. Uma abordagem para a identificação de desenvolvedores de software. *International Journal of Advanced Computer Science and Applications*, IGI Global, v. 8, n. 7, p. 217–224, 2017. Citado na página 15.

RASTOGI, A.; ARASU, S.; ROBILLARD, M. P. Software understanding: How much time do developers spend understanding code? *Proceedings of the 40th International Conference on Software Engineering*, ACM, p. 90–101, 2018. Citado na página 15.

RIEHLE, D.; LAGO, P. Version control systems: An empirical study. In: IEEE. *Proceedings of the 2007 International Conference on Software Engineering*. [S.l.], 2007. p. 661–670. Citado na página 21.

RITSCHEL, A.; ZINK, J. *International case study research: Theories, methods and practice*. [S.l.]: Springer International Publishing, 2018. Citado na página 36.

ROCHA, A. d.; SOUZA, G. d. S.; BARCELLOS, M. P. Medição de software e controle estatístico de processos. *PBQP Software, Brasília*, 2012. Citado na página 52.

ROCHKIND, M. J. The source code control system. *IEEE transactions on Software Engineering*, IEEE, n. 4, p. 364–370, 1975. Citado na página 21.

ROGER, S. P.; BRUCE, R. M. *Software engineering: a practitioner's approach*. [S.l.]: McGraw-Hill Education, 2015. Citado na página 15.

SANTOS, A. et al. Mining software repositories to identify library experts. In: *Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse*. [S.l.: s.n.], 2018. p. 83–91. Citado na página 29.

SANTOS, K. d. F.; GUERRERO, D. D.; FIGUEIREDO, J. C. D. Using developers contributions on software vocabularies to identify experts. In: IEEE. *2015 12th International Conference on Information Technology-New Generations*. [S.l.], 2015. p. 451–456. Citado 10 vezes nas páginas 16, 17, 20, 24, 25, 27, 29, 31, 33 e 51.

SILVA, J. e. a. da. Identificação de especialistas em desenvolvimento de software. *International Journal of Software Engineering and Applications*, IGI Global, v. 3, n. 2, p. 105–108, 2012. Citado na página 15.

SOMMERVILLE, I. *Engenharia de Software*. [S.l.]: Engenharia de Software. Uma Abordagem Profissional, 2013. v. 9. Citado 2 vezes nas páginas 20 e 23.

SREEKANTH, V. S.; RUBAN, S. Desenvolvimento de software ágil como um método de produtividade para equipes de desenvolvimento de software. *International Journal of Advanced Research in Computer Science and Software Engineering*, International Journal of Advanced Research in Computer Science and Software Engineering, v. 5, n. 1, p. 7–13, 2015. Citado na página 16.

STEEB, M. K.; BISSYANDÉ, T. F. A systematic review of methods to identify code-level software experts. *Journal of Systems and Software*, Elsevier, v. 129, p. 93–106, 2017. Citado na página 16.

SVIRIDOV, N.; EVTIKHIEV, M.; KOVALENKO, V. Tnm: A tool for mining of socio-technical data from git repositories. In: IEEE. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. [S.l.], 2021. p. 295–299. Citado 5 vezes nas páginas 16, 30, 31, 32 e 33.

TORKAR, R.; PARNAS, D. L.; LANO, K. Software development as a collaborative activity. In: ____. *The Human Side of Software Engineering*. [S.l.]: Springer, 2019. p. 195–225. Citado na página 16.

UDDIN, M. M.; RAHMAN, M. R.; AL-HADDAD, S. A review of version control systems for software development. *Information*, Multidisciplinary Digital Publishing Institute, v. 11, n. 7, p. 1–19, 2020. Citado na página 21.

VELASQUEZ, A. T.; SILVA, V. F. da. Coding habits: a study of software developers. In: IEEE. *Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN)*. [S.l.], 2016. p. 2480–2487. Citado na página 16.

WALKER, M. Software development expertise: Strategies for finding the right talent. *Communications of the ACM*, ACM, v. 62, n. 2, p. 74–83, 2019. Citado na página 15.

WANG, Y.; GAO, Y. Identifying code-level software experts based on source code repository. In: IEEE. *2019 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. [S.l.], 2019. p. 576–580. Citado na página 16.

WANG, Y. et al. Expertise identification in open source software development using structural metrics. *IEEE Transactions on Software Engineering*, IEEE, v. 45, n. 2, p. 176–193, 2019. Citado na página 17.

YIN, R. K. *Estudo de Caso-: Planejamento e métodos*. [S.l.]: Bookman editora, 2015. Citado na página 36.

YOON, J. et al. The cost of unknown code: An empirical study of learning during software maintenance. *IEEE Transactions on Software Engineering*, v. 42, p. 490–513, 2016. Citado na página 15.

ZHANG, C. et al. Authorship identification of source codes. In: SPRINGER. *Asia-Pacific Web (APWeb) and Web-Age Information Management (WAIM) Joint Conference on Web and Big Data*. [S.l.], 2017. p. 282–296. Citado 2 vezes nas páginas 29 e 31.

ZHANG, L. et al. Empirical research in software engineering—a literature survey. *Journal of Computer Science and Technology*, Springer, v. 33, n. 5, p. 876–899, 2018. Citado 2 vezes nas páginas 35 e 50.

ZHOU, Y. et al. Identifying expert developers from open-source projects using metrics. *Information and Software Technology*, Elsevier, v. 113, p. 1–20, 2020. Citado na página 17.

ZHU, W. et al. Exploring the potential of structural metrics for expertise identification in open source software development. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 25, n. 3, p. 1–36, 2016. Citado na página 17.

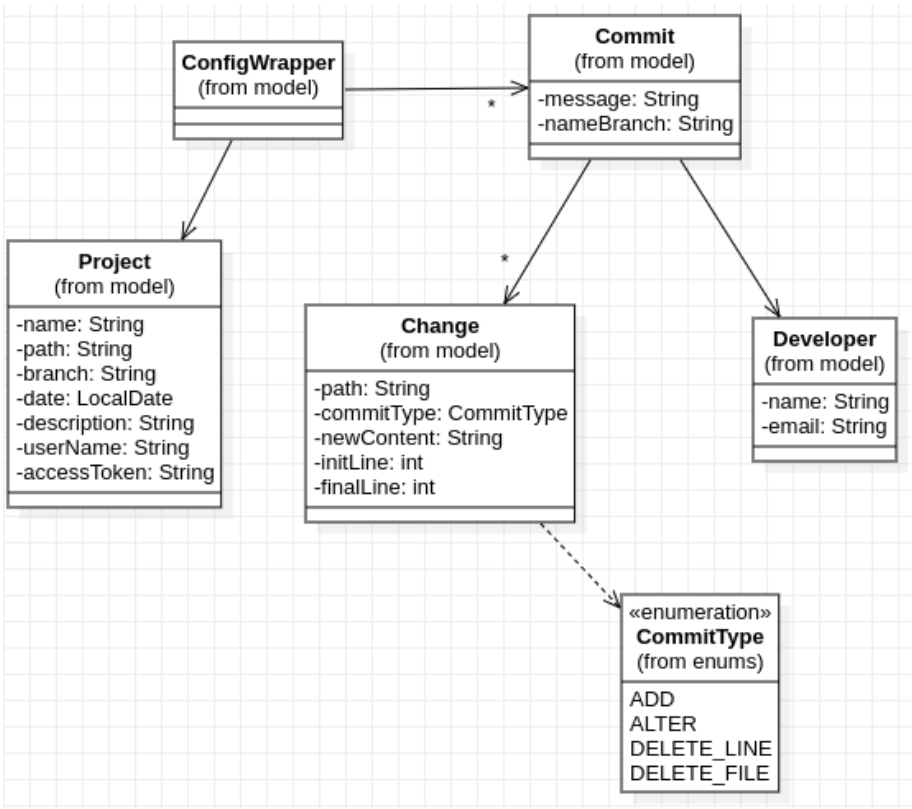
Apêndices

APÊNDICE A – GERADOR DE REPOSITÓRIOS FICTÍCIOS PARA BENCHMAK

O gerador de repositório fictícios é composto por artefatos de software que manipulam o sistema de versionamento git, foi desenvolvido usando a linguagem Java. O princípio de funcionamento do gerador se dá a partir das instruções contidas no arquivo de extensão **.ymal**, o tipo de arquivo YMAL são muito utilizados para arquivos de configurações, a sigla YAML pode significar em inglês *yet another markup language* (mais uma linguagem de marcação), é voltado para os dados, e não documentos.

As instruções fornecidas por meio de um arquivo de configuração, representam as interações que acontecem nas entidades do projeto fictício, assim simulando a realização de commits no repositório gerado. A Figura 12 mostra uma representação das entidades que contem o compõe o arquivo de configuração.

Figura 12 – Diagrama UML das entidades do gerador fictício.



Fonte: autoria própria.

Detalhando cada item preciso para construir o arquivo de configuração:

- **Project**: Entidade que armazena as informações do projeto, como nome, local de criação

do repositório, nome da *branch* padrão, data, descrição do projeto, usuário do GitHub e *token* de acesso para realizar o envio do projeto ao final da criação dos *commits* localmente.

- **Commit:** contém a mensagem que será registrada no *commit*, a *branch* de destino e a lista de mudanças;
- **Changes**: Lista de modificações do *commit*, com arquivo que está sendo alterado ou criado, tipo do *commit*(Adição, Alteração, Remoção de linha e deleção de arquivo), novo conteúdo para adição ou alteração, seguindo do início e final das linhas que serão modificadas;
- **Developer**: armazena as informações de quem está realizando a modificação, ou seja, o autor do *commit*.

Abaixo está um modelo do arquivo lido pela ferramenta:

Listing A.1 – Modelo do arquivo de construção de um repositório fictício

```
project:
  name:
  path:
  branch:
  description:
  userName:
  accessToken:
  date:
commits:
  - message:
    developer:
       name:
       email:
    changes:
      - path:
        commitType:
        newContent:
        initLine:
        finalLine:
```

Para executar a ferramenta basta preencher o arquivo com as configurações do repositório e com os commits que serão gerados. Após o preenchimento basta executar o seguinte comando em uma interface de linha de comando: *java -jar gerador-repositorio.jar "path-config.yaml"*. O path-config.yaml é o caminho do arquivo de configuração.

APÊNDICE B – VALIDAÇÃO DO COLETOR DE MÉTRICAS E INDICADOR DE ESPECIALISTA POR ABORDAGEM

Esta seção apresenta a criação de um repositório fictício para validação do Indicador de Especialista por Abordagem. Descreveremos os cenários específicos para os testes, nos quais usaremos o gerador de repositórios fictícios(apresentado no apêndice A) para simular as modificações, versionando cada uma, e posterior submetendo a ferramenta de Indicador de Especialista por Abordagem.

Resumo dos cenários que serão usados na criação do repositório fictício:

- Número de *commits* totais: 7
- Número de desenvolvedores: 3
- Número de entidades válidas: 3
- Número de linhas: 18 Adicionadas e 4 linhas removidas

O caso de aplicação da ferramenta começa com um arquivo com 3 linhas, seguido de um *commit*, o *commit* inicial.

- Nome do arquivo: File.java;
- Conteúdo do arquivo:

```
1  public class File {
2      // criando o arquivo file com 3 linhas
3  }
```

- Número do *commit*: 1;
- Autor do *commit*: Author1;
- Branch do *commit*: master;
- Mensagem do *commit*: Author1 criou o File.java e adicionou 3 novas linhas;
- Linhas adicionadas: 3 linhas no arquivo File.java;
- Observação: Ocorreram apenas adição de linhas sem deleções de linhas ou arquivos.

No arquivo de configurações que será lido pelo Gerador de Repositório Fictícios ficará dessa maneira para adicionar as informações do primeiro *commit*:

Listing B.1 – Configuração inicial do arquivo

```
project:
  name: teste-ferramenta-extracao
  path: /tmp/
  branch: master
  description: teste da ferramenta de extracao de metricas de
     especialidade
  userName: wemerson
  accessToken: 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
date: 2022-03-21
```

Após a configuração inicial, adicionamos as informações do *commit*:

Listing B.2 – Primeiro commit

```
commits:
  - message: Author1 criou o File.java e adicionou 3 novas
     linhas
    developer:
      name: author1
      email: author1@email.com
    changes:
      - path: File.java
        commitType: ADD
        newContent: "public class File {\n// criando o arquivo
           file com 3 linhas\n}"
        initLine: 1
        finalLine: 3
```

No próximo *commit* do arquivo, são adicionadas mais 3 linhas ao final do arquivo.

- Nome do arquivo: File.java;
- Conteúdo do arquivo:

```
public class File {
    // criando o arquivo file com 3 linhas
}
/* Classe de teste da ferramenta de extração a partir de um
 repositório fictício*/
```

- Número do *commit*: 2;
- Autor do *commit*: Author1;
- Branch do *commit*: master;
- Mensagem do *commit*: Author1 modificou o File.java e adicionou 3 novas linhas;
- Linhas adicionadas: 3 linhas no arquivo File.java;
- Observação: Ocorreram apenas adição de linhas sem deleções de linhas ou arquivos

No arquivo de configuração fica da seguinte forma:

Listing B.3 – Segundo *Commit*

```
commits:
  - message: Author1 modificou o File.java e adicionou 3 novas
      linhas;
    developer:
      name: author1
      email: author1@email.com
    changes:
      - path: File.java
        commitType: ALTER
        newContent: "/* Classe de teste da ferramenta\n de
            extracao a partir de um\nrepositorio ficticio*/"
        initLine: 3
        finalLine: 6
```

No próximo estado do arquivo, um segundo autor adicionou, a partir terceira linha, duas novas linhas ao arquivo.

- Nome do arquivo: File.java;
- Conteúdo do arquivo:

```
public class File {
    // criando o arquivo file com 3 linhas
    public void static main(String args[]){
    }// fim main
}
/* Classe de teste da ferramenta
 de extracao a partir de um
 repositorio ficticio*/
```

- Número do *commit*: 3;
- Autor do *commit*: Author2;
- Branch do *commit*: master;
- Mensagem do *commit*: Author2 modificou o File.java e adicionou 2 novas linhas;
- Linhas adicionadas: 2 linhas no arquivo File.java;
- Observação: Ocorreram apenas adição de linhas sem deleções de linhas ou arquivos;

Para melhorar a dinâmica de leitura dos cenários de teste, neste documento será adicionado ao final de toda a construção o arquivo de configuração final que será utilizado para gerar o repositório fictício.

Agora, um terceiro autor adicionará dois novos arquivos ao projeto, chamados File2.java e File3.java com 2 linhas cada, no quarto *commit*.

- Nome do arquivo: File2.java;
- Conteúdo do arquivo:

```
1  public class File2 {
2  }
3  public class File3 {
4  }
```

- Número do commit: 4;
- Autor do *commit*: Author3;
- Branch do *commit*: master;
- Mensagem do *commit*: Author3 adicionou ao projeto o File2.java e File3.java, com 2 linhas cada;
- Linhas adicionadas: 2 linhas no arquivo File2.java e File3.java;

No quinto *commit*, o autor 1 adicionou ao File2.java duas novas linhas entre a primeira e última linha.

- Nome do arquivo: File2.java;
- Conteúdo do arquivo:

```
1              public class File2 {
2                  private int a;
3                  private int b;
4              }
```

- Número do *commit*: 5;
- Autor do *commit*: Author1;
- Branch do *commit*: master;
- Mensagem do *commit*: Author1 modificou File2 com duas novas linhas;
- Linhas adicionadas: 2 linhas no arquivo File2.java;
- Observação: O Author1 adicionou duas novas linhas no arquivo e não apagou nenhuma delas.

No sexto *commit*, o autor 2 removeu as 3 últimas linhas adicionada pelo autor 1 do arquivo File.java.

- Nome do arquivo: File.java;
- Conteúdo do arquivo:

```
1               public class File {
2                  // criando o arquivo file com 3 linhas
3                  public void static main(String args[]){
4                  }// fim main
5              }
```

- Número do *commit*: 6;
- Autor do *commit*: Author2;
- Branch do *commit*: master;
- Mensagem do *commit*: Author2 modificou File removendo 3 linhas;
- Linhas adicionadas: 0 linhas;
- Observação: O Author2 retirou linhas do arquivo File.java adicionado pelo Author1

No sétimo *commit* de teste no projeto, o autor 1 modifica novamente o File.java e adiciona 4 novas linhas ao arquivo, após a quarta linha.

- Nome do arquivo: File.java;
- Conteúdo do arquivo:

```
1                public class File {
2                   // criando o arquivo file com 3 linhas
3                   public void static main(String args[]){
4                   }// fim main
5                   // método que exibe uma mensagem no terminal
6                   public void showMsg(String msg){
7                       System.out.println(msg);
8                   }
9               }
```

- Número do *commit*: 7;
- Autor do *commit*: Author1;
- Branch do *commit*: master;
- Mensagem do *commit*: Author1 modificou File.java adicionando um novo método de 3 linhas, mais comentário;
- Linhas adicionadas: 4 linhas;

Arquivo final que será lido pela ferramenta:

Listing B.4 – Arquivo de construção de um repositório fictício

```
1  project:
2    name: teste-ferramenta-extracao
3    path: /home/teste
4    branch: master
5    description: teste da ferramenta
6    userName: userNameGithub #usuario do github
7    accessToken: acessTokenGithub #adicionar aqui o acessToken gerado no github
8    date: 2022-03-21
9  commits:
10   - message: Author1 criou o file e adicionou 3 novas linhas
11     developer:
12       name: author1
13       email: author1@email.com
14     changes:
15       - path: File.java
16         commitType: ADD
```

```
17        newContent: "public class File{\n// criando o arquivo file com 3
              linhas\n}"
18        initLine: 1
19        finalLine: 3
20    - message: Author1 modificou o File.java e adicionou 3 novas linhas
21      developer:
22        name: author1
23        email: author1@email.com
24      changes:
25        - path: File.java
26          commitType: ALTER
27          newContent: "/* Classe de teste da ferramenta\n de extracao a partir
              de um\nrepositorio ficticio*/"
28          initLine: 3
29          finalLine: 6
30    - message: Author2 modificou o File.java e adicionou 2 novas linhas
31      developer:
32        name: author2
33        email: author2@email.com
34      changes:
35        - path: File.java
36          commitType: ALTER
37          newContent: "public void static main(String args[]){\n}// fim main"
38          initLine: 3
39          finalLine: 5
40    - message: Author3 adicionou ao projeto o File2.java e File3.java, com 2
        linhas cada
41      developer:
42        name: author3
43        email: author3@email.com
44      changes:
45        - path: File2.java
46          commitType: ADD
47          newContent: "public class File2 {\n}"
48          initLine: 1
49          finalLine: 3
50        - path: File3.java
51          commitType: ADD
52          newContent: "public class File3 {\n}"
53          initLine: 1
54          finalLine: 3
55    - message: Author1 modificou File2 com duas novas linhas
```

```
56    developer:
57      name: author1
58      email: author1@email.com
59    changes:
60      - path: File2.java
61        commitType: ALTER
62        newContent: "private int a;\nprivate int b;\n}"
63        initLine: 2
64        finalLine: 4
65  - message: Author2 modificou File removendo 3 linhas
66    developer:
67      name: author2
68      email: author2@email.com
69    changes:
70      - path: File.java
71        commitType: DELETE
72        newContent: ""
73        initLine: 6
74        finalLine: 8
75  - message: Author1 modificou File.java adicionando um novo metodo de 3
      linhas, mais comentario.
76    developer:
77      name: author1
78      email: author1@email.com
79    changes:
80      - path: File.java
81        commitType: ALTER
82        newContent: "// metodo que exibe uma mensagem no terminal\n public
            void showMsg(String msg){\nSystem.out.println(msg);\n}"
83        initLine: 5
84        finalLine: 9
```

Validação do gerador de repositórios fictícios

Para validarmos a ferramenta de indicações de especialista por abordagem, descrevamos o processo de extração e indicação manualmente. Esse procedimento é chamado de Teste de Mesa, no qual um processo manual é utilizado para validar a lógica de um determinado algoritmo.

Portanto, a construção dos cenários esperados através do teste de mesa, representando assim cada interação com o código realizada pelo gerador de repositórios fictícios, permite uma maior compreensão das medições extraídas pelo indicador de especialistas por abordagem.

Simularemos o funcionamento das indicações por cada abordagem, iniciando pela métrica de *commits*. A Tabela 16 contém a execução em forma de teste de mesa de cada um dos *commits* descritos na seção anterior.

Tabela 16 – Teste de mesa sobre o cálculo da métrica para a Abordagem por *Commit*

MÉTRICA DE COMMIT			
# Commit	File.java	File2.java	File3.java
1	author1	-	-
2	author1	-	-
3	author2	-	-
4	-	author3	author3
5	-	author1	-
6	author2	-	-
7	author1	-	-

Fonte: Autoria própria.

Na Tabela 16, podemos visualizar cada interação com as entidades criadas em cada *commit*, logo conseguimos contabilizar que: A entidade **File.java** recebeu um total de 5 *commits*, já a entidade **File2.java** recebeu 2 *commits* e a **File3.java** 1 *commit*. Também é possível identificar que, para **File.java** o especialista indicado, pela métrica de número de *commits*, será o **author1**, pois ele realizou 3 dos 5 *commits* que a entidade obteve. Semelhantemente, a entidade **File2.java** ficou com a especialidade compartilhada, ou seja, o **author1** e **author3** efetuaram apenas uma contribuição cada. Conforme a Tabela 17 exibe os resultados esperados para extração da métrica de *commit*.

Tabela 17 – Resultado esperado da extração dos especialistas pela métrica de *commit*

TOTALIZADORES			
File.java	5 Commits	author1	3 Commits
File2.java	2 Commits	author2	2 Commits
File3.java	1 Commit	author3	1 Commit

RESULTADO ESPERADO	
Entidade	Especialista
File.java	**author1**
File2.java	**author1/author3**
File3.java	**author3**

Fonte: Autoria própria.

Pela abordagem LoC, construímos a Tabela 18, que contém o teste de mesa de cada um dos *commits* descritos na seção anterior para ela. Cada linha da tabela apresenta a quantidade de linhas adicionadas e removidas na entidade e seu respectivo autor da ação. A entidade **File.java** recebeu 12 linhas e apenas 3 linhas removidas nas 5 contribuições registrado na entidade. Para este cenário, conforme à abordagem por LoC, o especialista indicado é o **author1**.

A entidade **File2.java** obteve 4 linhas adicionadas e nenhuma removida nas duas contribuições, assim como na métrica de *commit* essa entidade tem uma especialidade compartilhada, no qual os desenvolvedores **author1** e **author3** adicionaram e removeram os mesmos valores em linhas. Para **File3.java** apenas duas linhas adicionadas, indicando o **author3** como especialista.

Tabela 18 – Teste de mesa sobre o cálculo da métrica para a abordagem por LoC

MÉTRICA DE LoC									
# Commit	File.java			File2.java			File3.java		
	Linhas adicionadas	Linhas removidas	Author	Linhas adicionadas	Linhas removidas	Author	Linhas adicionadas	Linhas removidas	Author
1	3	0	author1	-	-	-	-	-	-
2	3	0	author1	-	-	-	-	-	-
3	2	0	author2	-	-	-	-	-	-
4	-	-	-	2	0	author3	2	0	author3
5	-	-	-	2	0	author1	-	-	-
6	0	3	author2	-	-	-	-	-	-
7	4	0	author1	-	-	-	-	-	-
TOTAL	**12**	**3**	-	**4**	**0**	-	**2**	**0**	-

Fonte: Autoria própria.

Semelhantemente as métricas de LoC e de *Commit*, geramos a Tabela 19 para a abordagem DOA. Precisamos calcular a especialidade seguindo a Fórmula 2, que para isso é necessário computar os fatores FA, DL e AC para cada entidade. Importante ressaltar, pois, toda interação com o código os fatores são incrementados. Todas as linhas da Tabela mostram a contabilização dos fatores.

$$DOA(d,f) = 3,293 + 1,098 \cdot FA + 0,164 \cdot DL - 0,321 \cdot ln(1+AC) \quad (2)$$

Tabela 19 – Teste de mesa sobre o cálculo da métrica para a abordagem por DOA

MÉTRICA DOA						
# Commit	Author	File.java	Author	File2.java	Author	File2.java
1	author1	FA = 1, DL=1, AC = 0	-	-	-	-
2	author1	FA = 1,DL=2, AC = 0	-	-	-	-
3	author2	FA = 0, DL=0, AC = 1	-	-	-	-
4	-	-	author3	FA = 1, DL=1, AC = 0	author3	FA = 1, DL=1, AC = 0
5	-	-	author1	FA = 0, DL=0, AC =1	-	-
6	author2	FA = 0, DL=0, AC = 2	-	-	-	-
7	author1	FA = 1, DL=3, AC = 0	-	-	-	-

Fonte: Autoria própria.

Explicando os resultadas apresentado na Tabela 19, temos:

- **File.java**:
- Aplicando a fórmula 2, dispomos dos valores do DOA:

$$DOA(author1, File.java) = 3,293 + 1,098 \cdot 1 + 0,164 \cdot 2 - 0,321 \cdot ln(1+2) \approx \textbf{3,57} \quad (3)$$

$$DOA(author2, File.java) = 3,293 + 1,098 \cdot 0 + 0,164 \cdot 0 - 0,321 \cdot ln(1+2) \approx \mathbf{3.02} \quad (4)$$

- Logo o especialista é o **author1**.
- **File2.java**
- Aplicando a fórmula 2, dispomos dos valores do DOA:

$$DOA(author3, File.java) = 3,293 + 1,098 \cdot 1 + 0,164 \cdot 1 - 0,321 \cdot ln(1+1) \approx \mathbf{4.34} \quad (5)$$

$$DOA(author1, File.java) = 3,293 + 1,098 \cdot 0 + 0,164 \cdot 0 - 0,321 \cdot ln(1+1) \approx \mathbf{3.24} \quad (6)$$

- Logo o especialista é o **author3**
- **File3.java**
- Aplicando a fórmula 2, dispomos dos valores do DOA:

$$DOA(author3, File.java) = 3,293 + 1,098 \cdot 1 + 0,164 \cdot 1 - 0,321 \cdot ln(1+1) \approx \mathbf{4.55} \quad (7)$$

- Logo o especialista é o **author3**

Resultados das extrações da ferramenta mostram que, após execução dos cenários de teste, podemos observar que as indicações dos especialista condizem com o esperado. Conforme mostra a Tabela 20. Pelas métricas de *commit*, LoC e DOA o especialista esperado é o **author1** para entidade **File.java**, e **File2.java** apresentou especialidade compartilhada em duas medidas, isto significa que quando uma métrica indica dois desenvolvedores como especialistas para uma dada entidade pela métrica em questão, logo as abordagens de número de *commit* e LoC, os desenvolvedores **author1** e **author3**, porém a métrica DOA indica o **author3**.

Importante destacar que, neste caso de especialidade compartilhada, a ferramenta irá considerar o último desenvolvedor que realizou mudanças na entidade como especialista, pois que a indicação presente na Tabela 20 para a entidade **File2.java** aparece o **author1**, a entidade **File3.java** tem uma indicação esperada de que seja o mesmo especialista para todas as métricas, o desenvolvedor **author3**.

Tabela 20 – Resultado da matriz de especialidade entidade do projeto de teste

Entidade	**Abordagem por Commit**		**Abordagem por LoC**		**Abordagem por DOA**	
	Teste de Mesa	**Ferramenta**	**Teste de Mesa**	**Ferramenta**	**Teste de Mesa**	**Ferramenta**
File.java	author1	author1	author1	author1	author1	author1
File2.java	author1/author3	author1	author1/author3	author1	author3	author3
File3.java	author3	author3	author3	author3	author3	author3

Fonte: Autoria própria.

APÊNDICE C – QUESTIONÁRIO PARA APLICADO PARA CONSTRUÇÃO DO ORÁCULO

Estou construindo um oráculo para o protótipo de uma ferramenta que extraí métrica de código-fontes para identificar especialista de código-fonte que desenvolvi. O sistema está pronto e preciso validar se está recomendando corretamente, para tanto preciso do apoio do time para responderem a esse pequeno questionário abaixo, dando pontuações de 0-10 para cada entidade em questão, você atribuirá uma nota que você acha que conhece da entidade.

Ao final, farei uma comparação dos resultados que você me informou com os resultados extraídos da ferramenta.

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_1.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_2.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_3.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_4.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_5.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_6.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_7.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_8.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_9.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_10.java`?

`0 1 2 3 4 5 6 7 8 9 10`

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_11.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_12.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_13.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_14.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_15.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_16.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_17.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_18.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_19.java`?

0 1 2 3 4 5 6 7 8 9 10

Em uma escala de 0 a 10, quanto você acha que conhece da entidade `ENTIDADE_20.java`?

0 1 2 3 4 5 6 7 8 9 10

APÊNDICE D – LISTA DOS PROJETOS USADO NO ESTUDO DE CASO

Tabela 21 – Lista de nomes e link dos repositórios dos projetos analisados

Nome do projeto	Coluna 2
ADVANCED-JAVA	https://github.com/doocs/advanced-java.git
AKHQ	https://github.com/tchiotludo/akhq.git
ANDROID-CLEANARCHITECTURE	https://github.com/android10/Android-CleanArch...
ANDROID-PICKERVIEW	https://github.com/Bigkoo/Android-PickerView.git
ANDROID-UNIVERSAL-IMAGE-LOADER	https://github.com/nostra13/Android-Universal-...
ANDROIDUTILCODE	https://github.com/Blankj/AndroidUtilCode.git
APOLLO	https://github.com/ctripcorp/apollo.git
AROUTER	https://github.com/alibaba/ARouter.git
ARTHAS	https://github.com/alibaba/arthas.git
BASERECYCLERVIEWADAPTERHELPER	https://github.com/CymChad/BaseRecyclerViewAda...
BAZEL	https://github.com/bazelbuild/bazel.git
BUTTERKNIFE	https://github.com/JakeWharton/butterknife.git
CANAL	https://github.com/alibaba/canal.git
CAT	https://github.com/dianping/cat.git
CIRCLEIMAGEVIEW	https://github.com/hdodenhof/CircleImageView.git
CS-NOTES	https://github.com/CyC2018/CS-Notes.git
DBEAVER	https://github.com/dbeaver/dbeaver.git
DISRUPTOR	https://github.com/LMAX-Exchange/disruptor.git
DORAEMONKIT	https://github.com/didi/DoraemonKit.git
DRUID	https://github.com/alibaba/druid.git
DUBBO	https://github.com/apache/dubbo.git
EASYEXCEL	https://github.com/alibaba/easyexcel.git
ELADMIN	https://github.com/elunez/eladmin.git
ELASTICSEARCH	https://github.com/elastic/elasticsearch.git
EVENTBUS	https://github.com/greenrobot/EventBus.git
EXOPLAYER	https://github.com/google/ExoPlayer.git
FASTJSON	https://github.com/alibaba/fastjson.git
FIZZBUZZENTERPRISEEDITION	https://github.com/EnterpriseQualityCoding/Fiz...
FLINK	https://github.com/apache/flink.git
FRESCO	https://github.com/facebook/fresco.git
GHIDRA	https://github.com/NationalSecurityAgency/ghid...
GLIDE	https://github.com/bumptech/glide.git
GRAAL	https://github.com/oracle/graal.git
GSON	https://github.com/google/gson.git
GSYVIDEOPLAYER	https://github.com/CarGuo/GSYVideoPlayer.git
GUAVA	https://github.com/google/guava.git
HALO	https://github.com/halo-dev/halo.git
HELLO-ALGORITHM	https://github.com/geekxh/hello-algorithm.git
HIKARICP	https://github.com/brettwooldridge/HikariCP.git
HUTOOL	https://github.com/looly/hutool.git
HYSTRIX	https://github.com/Netflix/Hystrix.git
INTERVIEWS	https://github.com/kdn251/interviews.git
JADX	https://github.com/skylot/jadx.git
JAVA	https://github.com/TheAlgorithms/Java.git
JAVA-DESIGN-PATTERNS	https://github.com/iluwatar/java-design-patter...
JAVA8-TUTORIAL	https://github.com/winterbe/java8-tutorial.git

Continua na próxima página

Tabela 21 – *Continuação da tabela*

Nome do projeto	**Link do repositório**
JAVAGUIDE	https://github.com/Snailclimb/JavaGuide.git
JCSPROUT	https://github.com/crossoverJie/JCSprout.git
JEECG-BOOT	https://github.com/zhangdaiscott/jeecg-boot.git
JENKINS	https://github.com/jenkinsci/jenkins.git
KAFKA	https://github.com/apache/kafka.git
KEYCLOCK	https://github.com/keycloak/keycloak.git
LEETCODEANIMATION	https://github.com/MisterBooo/LeetCodeAnimatio...
LIBGDX	https://github.com/libgdx/libgdx.git
LITEMALL	https://github.com/linlinjava/litemall.git
LOGGER	https://github.com/orhanobut/logger.git
LOTTIE-ANDROID	https://github.com/airbnb/lottie-android.git
LOTTIE-REACT-NATIVE	https://github.com/lottie-react-native/lottie-...
LUBAN	https://github.com/Curzibn/Luban.git
MATERIAL-ANIMATIONS	https://github.com/lgvalle/Material-Animations...
MIAOSHA	https://github.com/qiurunze123/miaosha.git
MPANDROIDCHART	https://github.com/PhilJay/MPAndroidChart.git
MUSICBOT	https://github.com/jagrosh/MusicBot.git
MYBATIS-3	https://github.com/mybatis/mybatis-3.git
NACOS	https://github.com/alibaba/nacos.git
NETTY	https://github.com/netty/netty.git
NEWPIPE	https://github.com/TeamNewPipe/NewPipe.git
OKHTTP	https://github.com/square/okhttp.git
PHOTOVIEW	https://github.com/Baseflow/PhotoView.git
PICASSO	https://github.com/square/picasso.git
PROXYEE-DOWN	https://github.com/proxyee-down-org/proxyee-do...
QMUI_ANDROID	https://github.com/Tencent/QMUI_Android.git
REDISSON	https://github.com/redisson/redisson.git
RETROFIT	https://github.com/square/retrofit.git
ROCKETMQ	https://github.com/apache/rocketmq.git
RXANDROID	https://github.com/ReactiveX/RxAndroid.git
RXJAVA	https://github.com/ReactiveX/RxJava.git
SEATA	https://github.com/seata/seata.git
SELENIUM	https://github.com/SeleniumHQ/selenium.git
SENTINEL	https://github.com/alibaba/Sentinel.git
SHARDINGSPHERE	https://github.com/apache/shardingsphere.git
SIGNAL-ANDROID	https://github.com/signalapp/Signal-Android.git
SKYWALKING	https://github.com/apache/skywalking.git
SMARTREFRESHLAYOUT	https://github.com/scwang90/SmartRefreshLayout...
SPRING-BOOT	https://github.com/spring-projects/spring-boot...
SPRING-BOOT-DEMO	https://github.com/xkcoding/spring-boot-demo.git
SPRING-BOOT-EXAMPLES	https://github.com/ityouknow/spring-boot-examp...
SPRING-CLOUD-ALIBABA	https://github.com/alibaba/spring-cloud-alibab...
SPRING-FRAMEWORK	https://github.com/spring-projects/spring-fram...
SPRINGBOOT-LEARNING	https://github.com/dyc87112/SpringBoot-Learnin...
SPRINGBOOT-LEARNING-EXAMPLE	https://github.com/JeffLi1993/springboot-learn...
TERMUX-APP	https://github.com/termux/termux-app.git
TINKER	https://github.com/Tencent/tinker.git
TOBETOPJAVAER	https://github.com/hollischuang/toBeTopJavaer.git
TUTORIALS	https://github.com/eugenp/tutorials.git
VHR	https://github.com/lenve/vhr.git
XXL-JOB	https://github.com/xuxueli/xxl-job.git
ZHENG	https://github.com/shuzheng/zheng.git
ZIPKIN	https://github.com/openzipkin/zipkin.git
ZXING	https://github.com/zxing/zxing.git

Fim da lista de nomes e link dos repositórios dos projetos analisados

APÊNDICE E – RELAÇÃO DA QUANTIDADE DE ENTIDADES CONSIDERADAS EM CADA PROJETO.

Tabela 22 – Quantidade de entidades usada para identificação dos especialistas

Projeto	**# Entidades**	**#Entidades Consideradas**	**% Entidades Consideradas**
ADVANCED-JAVA	311	3	0.964630
AKHQ	521	330	63.339731
ANDROID-CLEANARCHITECTURE	121	96	79.338843
ANDROID-PICKERVIEW	111	77	69.369369
ANDROID-UNIVERSAL-IMAGE-LOADER	147	108	73.469388
ANDROIDUTILCODE	693	443	63.924964
APOLLO	1381	1036	75.018103
AROUTER	152	108	71.052632
ARTHAS	1766	741	41.959230
BASERECYCLERVIEWADAPTERHELPER	283	164	57.950530
BAZEL	9051	6828	75.439178
BUTTERKNIFE	243	187	76.954733
CANAL	1257	1135	90.294352
CAT	3719	3072	82.602850
CIRCLEIMAGEVIEW	30	10	33.333333
CS-NOTES	2558	584	22.830336
DBEAVER	6687	4631	69.253776
DISRUPTOR	246	212	86.178862
DORAEMONKIT	3499	1340	38.296656
DRUID	5781	5481	94.810586
DUBBO	2856	2673	93.592437
EASYEXCEL	471	415	88.110403
ELADMIN	308	288	93.506494
ELASTICSEARCH	21068	17323	82.224226
EVENTBUS	133	103	77.443609
EXOPLAYER	2731	1575	57.671183
FASTJSON	3110	3083	99.131833
FIZZBUZZENTERPRISEEDITION	102	92	90.196078
FLINK	16774	12607	75.157983
FRESCO	2736	2136	78.070175
GHIDRA	14074	11007	78.208043
GLIDE	1088	780	71.691176
GRAAL	12113	11152	92.066375
GSON	246	221	89.837398
GSYVIDEOPLAYER	562	399	70.996441
GUAVA	3036	3009	99.110672
HALO	589	539	91.511036
HELLO-ALGORITHM	1202	146	12.146423
HIKARICP	124	112	90.322581
HUTOOL	1618	1555	96.106304
HYSTRIX	470	418	88.936170
INTERVIEWS	554	517	93.321300
JADX	1225	1004	81.959184
JAVA	254	234	92.125984

JAVA-DESIGN-PATTERNS	2243	1610	71.778868
JAVA8-TUTORIAL	82	78	95.121951
JAVAGUIDE	431	21	4.872390
JCSPROUT	221	111	50.226244
JEECG-BOOT	1754	949	54.104903
JENKINS	11775	3060	25.987261
KAFKA	4201	3384	80.552249
KEYCLOAK	9708	8433	86.866502
LEETCODEANIMATION	368	17	4.619565
LIBGDX	4561	2991	65.577724
LITEMALL	1582	826	52.212389
LOGGER	54	23	42.592593
LOTTIE-ANDROID	1060	782	73.773585
LOTTIE-REACT-NATIVE	1181	34	2.878916
LUBAN	57	31	54.385965
MATERIAL-ANIMATIONS	87	47	54.022989
MIAOSHA	1372	800	58.309038
MPANDROIDCHART	386	282	73.056995
MUSICBOT	85	74	87.058824
MYBATIS-3	1852	1820	98.272138
NACOS	1305	1156	88.582375
NETTY	3123	2819	90.265770
NEWPIPE	1502	754	50.199734
OKHTTP	568	178	31.338028
PHOTOVIEW	72	45	62.500000
PICASSO	182	78	42.857143
PROXYEE-DOWN	104	69	66.346154
QMUI_ANDROID	1765	598	33.881020
REDISSON	1824	1782	97.697368
RETROFIT	379	295	77.836412
ROCKETMQ	1186	1064	89.713322
RXANDROID	42	21	50.000000
RXJAVA	1939	1881	97.008767
SEATA	1691	1551	91.720875
SELENIUM	6021	4846	80.484969
SENTINEL	1432	1331	92.946927
SHARDINGSPHERE	6391	5320	83.242059
SIGNAL-ANDROID	4332	3577	82.571560
SKYWALKING	5571	4657	83.593610
SMARTREFRESHLAYOUT	506	354	69.960474
SPRING-BOOT	7448	6430	86.331901
SPRING-BOOT-DEMO	890	718	80.674157
SPRING-BOOT-EXAMPLES	395	346	87.594937
SPRING-CLOUD-ALIBABA	633	521	82.306477
SPRING-FRAMEWORK	8813	8284	93.997504
SPRINGBOOT-LEARNING	823	578	70.230863
SPRINGBOOT-LEARNING-EXAMPLE	319	266	83.385580
TERMUX-APP	297	247	83.164983
TINKER	420	309	73.571429
TOBETOPJAVAER	211	2	0.947867
TUTORIALS	20784	17324	83.352579
VHR	242	175	72.314050
XXL-JOB	287	201	70.034843
ZHENG	1251	928	74.180655

ZIPKIN	732	561	76.639344
ZXING	3633	1246	34.296724

APÊNDICE F – CONVERGÊNCIA E DIVERGÊNCIA DAS ENTIDADES NOS 100 PROJETOS

Tabela 23 – Relação de convergência entre as entidades nos 100 projetos

Projeto	# Entidades Consideradas	Convergência	Divergência	Percentual Convergência	Percentual Divergência
advanced-java	7	6	1	85,71	14,29
akhq	330	186	144	56,36	43,64
Android-CleanArchitecture	96	93	3	96,88	3,13
Android-PickerView	77	35	42	45,45	54,55
Android-Universal-Image-Loader	108	4	104	3,7	96,30
AndroidUtilCode	443	313	130	70,65	29,35
apollo	1036	654	382	63,13	36,87
ARouter	108	79	29	73,15	26,85
arthas	741	588	153	79,35	20,65
BaseRecyclerViewAdapterHelper	164	101	63	61,59	38,41
bazel	6828	1632	5196	23,9	76,10
butterknife	187	137	50	73,26	26,74
canal	1135	801	334	70,57	29,43
cat	3072	1589	1483	51,73	48,27
CircleImageView	10	10	0	100	0,00
CS-Notes	584	584	0	100	0,00
dbeaver	4631	1382	3249	29,84	70,16
disruptor	212	119	93	56,13	43,87
DoraemonKit	1340	415	925	30,97	69,03
druid	5481	4695	786	85,66	14,34
dubbo	2673	1140	1533	42,65	57,35
easyexcel	415	381	34	91,81	8,19
eladmin	288	90	198	31,25	68,75
elasticsearch	17323	7647	9676	44,14	55,86
EventBus	103	5	98	4,85	95,15
ExoPlayer	1575	859	716	54,54	45,46
fastjson	3083	2750	333	89,2	10,80
FizzBuzzEnterpriseEdition	92	27	65	29,35	70,65
flink	12607	2417	10190	19,17	80,83
fresco	2136	806	1330	37,73	62,27
ghidra	11007	8225	2782	74,73	25,27
glide	780	515	265	66,03	33,97
graal	11152	5895	5257	52,86	47,14
gson	221	146	75	66,06	33,94
GSYVideoPlayer	399	232	167	58,15	41,85
guava	3009	1079	1930	35,86	64,14
halo	539	329	210	61,04	38,96
hello-algorithm	146	1	145	0,68	99,32
HikariCP	112	62	50	55,36	44,64
hutool	1555	1532	23	98,52	1,48
Hystrix	418	292	126	69,86	30,14
interviews	517	144	373	27,85	72,15
jadx	1004	689	315	68,63	31,37
java8-tutorial	78	10	68	12,82	87,18

Java	234	106	128	45,3	54,70
java-design-patterns	1610	387	1223	24,04	75,96
JavaGuide	21	19	2	90,48	9,52
JCSprout	111	106	5	95,5	4,50
jeecg-boot	949	639	310	67,33	32,67
jenkins	3060	1873	1187	61,21	38,79
kafka	3384	1785	1599	52,75	47,25
keycloak	8433	4466	3967	52,96	47,04
LeetCodeAnimation	17	17	0	100	0,00
libgdx	2991	2062	929	68,94	31,06
litemall	826	70	756	8,47	91,53
logger	23	17	6	73,91	26,09
lottie-android	782	230	552	29,41	70,59
lottie-react-native	34	9	25	26,47	73,53
Luban	31	29	2	93,55	6,45
Material-Animations	47	42	5	89,36	10,64
miaosha	800	730	70	91,25	8,75
MPAndroidChart	282	88	194	31,21	68,79
MusicBot	74	57	17	77,03	22,97
mybatis-3	1820	597	1223	32,8	67,20
nacos	1156	456	700	39,45	60,55
netty	2819	948	1871	33,63	66,37
NewPipe	754	298	456	39,52	60,48
okhttp	178	106	72	59,55	40,45
PhotoView	45	35	10	77,78	22,22
picasso	78	32	46	41,03	58,97
proxyee-down	69	56	13	81,16	18,84
QMUI_Android	598	409	189	68,39	31,61
redisson	1782	1493	289	83,78	16,22
retrofit	295	221	74	74,92	25,08
rocketmq	1064	861	203	80,92	19,08
RxAndroid	21	4	17	19,05	80,95
RxJava	1881	1556	325	82,72	17,28
seata	1551	699	852	45,07	54,93
selenium	4846	3113	1733	64,24	35,76
Sentinel	1331	1009	322	75,81	24,19
shardingsphere	5320	1352	3968	25,41	74,59
Signal-Android	3577	2128	1449	59,49	40,51
skywalking	4657	2480	2177	53,25	46,75
SmartRefreshLayout	354	67	287	18,93	81,07
spring-boot	6430	3142	3288	48,86	51,14
spring-boot-demo	718	461	257	64,21	35,79
spring-boot-examples	346	254	92	73,41	26,59
SpringBoot-Learning	578	571	7	98,79	1,21
springboot-learning-example	266	234	32	87,97	12,03
spring-cloud-alibaba	521	190	331	36,47	63,53
spring-framework	8284	3398	4886	41,02	58,98
termux-app	247	177	70	71,66	28,34
tinker	309	202	107	65,37	34,63
toBeTopJavaer	2	2	0	100	0,00
tutorials	17324	7013	10311	40,48	59,52
vhr	175	157	18	89,71	10,29
xxl-job	201	132	69	65,67	34,33
zheng	928	768	160	82,76	17,24

zipkin	561	359	202	63,99	36,01
zxing	1246	815	431	65,41	34,59

Anexos

ANEXO A – PROTOCOLO DE PESQUISA *QUASI*-SISTEMÁTICA

Objetivo do Trabalho:

Este protocolo de revisão sistemática pretende identificar, analisar e discutir os métodos e técnicas existentes, quais são as métricas de autoria em código-fonte existentes e ferramentas que usem esse tipo de métrica para identificar especialista em código-fonte, bem como sua aplicação em projetos de software. Não se cogita investigar características das métrica na sua especificidade, mas de uma maneira geral, identificar quais são ou características desses métodos.

Definição da questão de pesquisa: Quais são as métricas para identificação de especialistas em código-fonte?

Seleção de bases de dados:

As bases de dados relevantes, como a *IEEEXplore Digital Library*, *ACM Digital Library*, *Scopus*, *Science Direct*, serão pesquisadas para encontrar publicações relacionadas ao tema de métricas de autoria de código-fonte e identificação de especialistas em código-fonte.

As seguintes strings de busca base:

(metric NEAR0 authorship OR authorship NEAR1 metric) AND (source code NEAR0 expert OR expert NEAR1 source code) AND (sofware source code NEAR0 knowledge OR knowledge NEAR1 source code) AND (source code NEAR0 knowledge metrics OR knowledge metrics NEAR1 source code) AND (expertise NEAR0 source code OR source code NEAR1 expertise)

Seleção dos estudos

Os estudos selecionados para incluir na revisão sistemática deverão ser publicações em língua inglesa, abordando técnicas, métricas ou ferramentas para métricas de autoria de código-fonte e/ou identificação de especialistas em código-fonte. Deverão incluir as seguintes palavras-chave: Metric authorship in source code, Source code expert analysis; Software source code knowledge, Source code knowledge metrics; Expertise in source code. Além disso, os estudos deverão conter informações sobre a aplicação das métricas de autoria de código-fonte e/ou identificação de especialistas em código-fonte em projetos de software.

Inclusão de estudos:

Os critérios de inclusão de estudos para esta revisão sistemática incluirão artigos publicados entre os anos 2000 e 2022. Os estudos selecionados deverão conter informações relevantes sobre a aplicação das métricas de autoria de código-fonte e/ou identificação de especialistas em código-fonte em projetos de software. Publicações em língua inglesa, abordando técnicas, métri-

cas ou ferramentas para métricas de autoria de código-fonte e/ou identificação de especialistas em código-fonte.

Avaliação de qualidade:

Os estudos selecionados para a revisão sistemática serão avaliados quanto à qualidade conforme os critérios estabelecidos, de data, linguá produzida e com relação à temática da pesquisa. Estes critérios incluirão, mas não se limitarão a, a relevância do estudo para o tema da revisão, a qualidade do conteúdo do estudo, a relevância dos dados e informações apresentadas e a validade dos resultados apresentados. Os estudos que não atenderem aos critérios de qualidade estabelecidos serão excluídos da revisão sistemática.

Resultados:

Os resultados da revisão sistemática serão analisados e discutidos para identificar as técnicas e métricas mais comuns para métricas de autoria de código-fonte e identificação de especialistas em código-fonte, bem como sua aplicação em projetos de software.

Conclusões:

Após a análise e discussão dos resultados da revisão sistemática, serão elaboradas as conclusões finais do projeto, identificando quais técnicas e métricas são mais adequadas para métricas de autoria de código-fonte e identificação de especialistas em código-fonte, bem como sua aplicação em projetos de software. Essas conclusões deverão abordar os principais resultados da revisão sistemática, a qualidade das métricas e técnicas identificadas, a validade dos resultados apresentados e as possíveis limitações do projeto. Além disso, as conclusões deverão fornecer recomendações para futuras pesquisas na área.

ANEXO B – DATASHEET DA EXTRAÇÃO DO PROTÓTIPO DA MÉTRICA DE *COMMIT* PARA O PROJETO HIKARICP

DADOS
StubStatement.java, {'Brett Wooldridge': '8'}, {'Guillaume Smet': '1'}, {'nothing': '1'};
Wall-of-Fame.md, {'Brett Wooldridge': '10'};
ConnectionPoolSizeVsThreadsTest.java, {'Brett Wooldridge': '6'}, {'shendley': '2'}, {'Stephan Schroevers': '1'};
TestConnections.java, {'Brett Wooldridge': '18'}, {'Jungtaek Lim': '1'}, {'Nitin': '19'}, {'nothing': '1'};
LICENSE, {'brettwooldridge': '1'};
TestProxies.java, {'Brett Wooldridge': '3'}, {'Guido Medina': '1'}, {'Nitin': '2'}, {'puntogil': '1'};
MiscTest.java, {'Brett Wooldridge': '5'}, {'Guido Medina': '1'}, {'Nitin': '4'};
TomcatConcurrentBagLeakTest.java, {'Brett Wooldridge': '6'}, {'Gili Tzabari': '1'};
HikariCPCollector.java, {'Aleksandr Podkutin': '1'}, {'Brett Wooldridge': '1'}, {'cenk ak\xc4\xb1n': '1'}, {'Manabu Matsuzaki': '1'}, {'puntogil': '1'}, {'Tokuhiro Matsuno': '1'};
.settings/org.eclipse.pde.core.prefs, {'Brett Wooldridge': '1'};
TestConnectionCloseBlocking.java, {'Brett Wooldridge': '11'}, {'puntogil': '1'};
CodaHaleMetricsTracker.java, {'Brett Wooldridge': '5'}, {'Jack': '1'}, {'Manabu Matsuzaki': '1'}, {'Nitin': '1'}, {'shendley': '1'}, {'Stephan Schroevers': '1'}, {'Will Vuong': '1'};
PoolEntry.java, {'Brett Wooldridge': '15'}, {'Mikhail Mazurskiy': '1'}, {'Nitin': '22'}, {'nothing': '1'};
propfile3.properties, {'Brett Wooldridge': '2'};
IsolationTest.java, {'Brett Wooldridge': '4'}, {'puntogil': '1'};
HouseKeeperCleanupTest.java, {'Brett Wooldridge': '3'}, {'Fred Deschenes': '1'}, {'Martin St\xc5\x99\xc3\xad\xc5\xbe': '1'}, {'Stephan Schroevers': '1'};
PrometheusHistogramMetricsTracker.java, {'Vincent van Donselaar': '1'}, {'Vladislav Golub': '1'};
StatementTest.java, {'Brett Wooldridge': '3'}, {'Nitin': '4'}, {'puntogil': '1'}, {'stepbeekio': '1'};
TestPropertySetter.java, {'Brett Wooldridge': '3'}, {'puntogil': '1'};
PrometheusMetricsTrackerFactoryTest.java, {'Aleksandr Podkutin': '1'}, {'Brett Wooldridge': '1'}, {'Tom Hombergs': '1'};
HikariPoolMXBean.java, {'Brett Wooldridge': '6'};
PropertyElf.java, {'Brett Wooldridge': '11'}, {'fpirson': '1'}, {'freafrea': '1'}, {'Johno Crawford': '1'}, {'Nitin': '6'};
HikariJNDIFactory.java, {'Brett Wooldridge': '9'}, {'Mikhail Mazurskiy': '1'}, {'Nitin': '5'};
TestSealedConfig.java, {'Brett Wooldridge': '1'};
.gitignore, {'Andy Shulman': '1'}, {'Brett Wooldridge': '3'}, {'lburgazzoli': '1'};
install-jdk.sh, {'Brett Wooldridge': '1'};
MetricsTrackerTest.java, {'Brett Wooldridge': '1'}, {'Will Vuong': '1'};
StubPreparedStatement.java, {'Brett Wooldridge': '13'}, {'Guillaume Smet': '1'};
travis-toolchains.xml, {'Brett Wooldridge': '1'};
README.md, {'brettwooldridge': '85'}, {'Brett Wooldridge': '332'}, {'Kazuhiro Sera': '1'}, {'Kyle': '2'}, {'Nitin': '6'}, {'Pascal Schumacher': '1'}, {'Ruben Dijkstra': '1'}, {'Tomek Wa\xc5\x82kuski': '2'}, {'Uday Shankar': '1'};
hibernate.properties, {'Brett Wooldridge': '1'};
BasicPoolTest.java, {'Brett Wooldridge': '5'};
StubPoolStats.java, {'Aleksandr Podkutin': '1'};
TestSaturatedPool830.java, {'Brett Wooldridge': '1'};
JavassistProxyFactory.java, {'Brett Wooldridge': '10'}, {'Guido Medina': '1'}, {'Sergey Monichev': '1'}, {'Yanming Zhou': '1'};
StubConnection.java, {'Brett Wooldridge': '13'}, {'Guillaume Smet': '1'}, {'Nitin': '2'}, {'nothing': '1'};
TestIsRunning.java, {'cyleph': '1'};
UtilityElfTest.java, {'Brett Wooldridge': '1'}, {'Enes A\xc3\xa7\xc4\xb1ko\xc4\x9flu': '1'};

HikariConfig.java, {"Arthur D'Andr\xc3\xa9a Alemar": '1'}, {'Brett Wooldridge': '124'}, {'Fred Deschenes': '1'},
{'Guido Medina': '1'}, {'Guillaume Smet': '1'}, {'Mikhail Mazurskiy': '1'}, {'NersesAM': '1'}, {'Nitin': '52'}, {'nothing': '1'},
{'Sevket G\xc3\xb6kay': '1'}, {'Stephan Schroevers': '1'}, {'Uday Shankar': '1'};
UnwrapTest.java, {'Brett Wooldridge': '3'}, {'Guido Medina': '1'}, {'Nitin': '2'};
PrometheusHistogramMetricsTrackerFactoryTest.java, {'Vincent van Donselaar': '1'};
JdbcDriverTest.java, {'Brett Wooldridge': '3'};
UtilityElf.java, {'Brett Wooldridge': '14'}, {'Enes A\xc3\xa7\xc4\xb1ko\xc4\x9flu': '1'}, {'Johno Crawford': '1'},
{'Mikhail Mazurskiy': '1'}, {'NersesAM': '1'}, {'Nitin': '8'};
HikariCPCollectorTest.java, {'Aleksandr Podkutin': '1'}, {'Brett Wooldridge': '8'}, {'cenk ak\xc4\xb1n': '1'},
{'Lorenzo Lucherini': '1'}, {'Manabu Matsuzaki': '1'}, {'puntogil': '1'}, {'Tokuhiro Matsuno': '1'}, {'Tom Hombergs': '1'};
StubDriver.java, {'Brett Wooldridge': '3'}, {'Guillaume Smet': '1'};
ConcurrentCloseConnectionTest.java, {'Brett Wooldridge': '2'}, {'puntogil': '1'};
.github/workflows/ci.yml, {'sullis': '1'};
ClockSource.java, {'ams2990': '1'}, {'Brett Wooldridge': '7'}, {'Guido Medina': '1'}, {'Nitin': '2'};
ProxyLeakTask.java, {'Andreas Brenk': '1'}, {'Brett Wooldridge': '3'}, {'Nitin': '5'};
PrometheusMetricsTrackerTest.java, {'Aleksandr Podkutin': '1'}, {'Brett Wooldridge': '4'},
{'cenk ak\xc4\xb1n': '2'}, {'Lorenzo Lucherini': '1'}, {'puntogil': '1'}, {'Stanislav Savulchik': '1'},
{'Tokuhiro Matsuno': '1'}, {'Tom Hombergs': '1'}, {'Will Vuong': '1'};
ConcurrentBag.java, {'Brett Wooldridge': '74'}, {'Mikhail Mazurskiy': '3'}, {'Nitin': '8'},
{'shendley': '1'}, {'Stephan Schroevers': '1'};
TestMBean.java, {'Brett Wooldridge': '9'}, {'Yanming Zhou': '1'};
HikariConnectionProvider.java, {'Brett Wooldridge': '4'};
log4j2-test.xml, {'Brett Wooldridge': '2'};
PropertyElfTest.java, {'Brett Wooldridge': '1'}, {'fpirson': '1'}, {'freafrea': '1'};
PrometheusHistogramMetricsTrackerTest.java, {'Brett Wooldridge': '1'}, {'Vincent van Donselaar': '1'}, {'Vladislav Golub': '1'};
ProxyFactory.java, {'Brett Wooldridge': '6'}, {'Nitin': '3'};
.travis.yml, {'Brett Wooldridge': '34'}, {'Gili Tzabari': '1'}, {'Matteo Gazzetta': '1'};
ProxyPreparedStatement.java, {'Brett Wooldridge': '3'};
ProxyConnection.java, {'Brett Wooldridge': '28'}, {'Guido Medina': '1'}, {'Johno Crawford': '2'}, {'Nitin': '22'};
IMetricsTracker.java, {'Brett Wooldridge': '4'};
HikariConfigMXBean.java, {'Brett Wooldridge': '4'}, {'Nitin': '2'};
StubBaseConnection.java, {'Brett Wooldridge': '4'}, {'Guillaume Smet': '1'}, {'puntogil': '1'};
Welcome-To-The-Jungle.md, {'Brett Wooldridge': '19'}, {'Kazuhiro Sera': '1'};
TestElf.java, {'Brett Wooldridge': '11'}, {'Gili Tzabari': '1'};
TestHibernate.java, {'Brett Wooldridge': '2'}, {'puntogil': '1'};
IsolationLevel.java, {'Brett Wooldridge': '1'}, {'Enes A\xc3\xa7\xc4\xb1ko\xc4\x9flu': '1'};
pom.xml, {'Andy Shulman': '1'}, {'Brett Wooldridge': '361'}, {'checketts': '2'}, {'Gili Tzabari': '5'}, {'Guido Medina': '1'},
{'Johno Crawford': '1'}, {'lburgazzoli': '3'}, {'Matteo Gazzetta': '4'}, {'Pascal Schumacher': '2'}, {'Prashant Bhat': '1'},
{'Sergey Monichev': '1'}, {'sullis': '3'}, {'Tokuhiro Matsuno': '1'};
StubResultSet.java, {'Brett Wooldridge': '10'}, {'Guillaume Smet': '1'};
ProxyDatabaseMetaData.java, {'Brett Wooldridge': '3'};
FastList.java, {'Brett Wooldridge': '14'}, {'Johno Crawford': '1'};
ProxyStatement.java, {'Brett Wooldridge': '8'}, {'Guido Medina': '1'}, {'Nitin': '1'}, {'stepbeekio': '1'};

HikariConfigurationUtil.java, {’Brett Wooldridge’: ’5’};
SQLExceptionOverride.java, {’Brett Wooldridge’: ’1’};
MetricsTracker.java, {’Brett Wooldridge’: ’15’}, {’Mikhail Mazurskiy’: ’2’}, {’Nitin’: ’2’}, {’shendley’: ’1’},
{’Stephan Schroevers’: ’1’}, {’Will Vuong’: ’1’};
propfile1.properties, {’Brett Wooldridge’: ’5’};
TestConcurrentBag.java, {’Brett Wooldridge’: ’6’}, {’Guido Medina’: ’1’}, {’Nitin’: ’2’};
HikariDataSource.java, {"Arthur D’Andr\xc3\xa9a Alemar": ’1’}, {’Brett Wooldridge’: ’53’}, {’cyleph’: ’1’},
{’Fred Deschenes’: ’1’}, {’Guido Medina’: ’1’}, {’Guillaume Smet’: ’2’}, {’Mikhail Mazurskiy’: ’4’}, {’Nitin’: ’1’};
OSGiBundleTest.java, {’Brett Wooldridge’: ’4’}, {’Gili Tzabari’: ’1’};
CHANGES, {’Brett Wooldridge’: ’144’}, {’Nitin’: ’3’};
TestJavassistCodegen.java, {’Brett Wooldridge’: ’1’}, {’Yanming Zhou’: ’1’};
CodaHaleMetricsTrackerTest.java, {’Aleksandr Podkutin’: ’1’}, {’Brett Wooldridge’: ’3’}, {’Jack’: ’1’},
{’Manabu Matsuzaki’: ’1’}, {’shendley’: ’1’}, {’Stephan Schroevers’: ’1’};
.editorconfig, {’Mikhail Mazurskiy’: ’1’};
StubDataSource.java, {’Brett Wooldridge’: ’6’}, {’Guillaume Smet’: ’1’}, {’nothing’: ’1’},
{’shendley’: ’1’}, {’Stephan Schroevers’: ’1’};
MicrometerMetricsTracker.java, {’Brett Wooldridge’: ’5’}, {’checketts’: ’2’},
{’Manabu Matsuzaki’: ’1’}, {’Tommy Ludwig’: ’3’};
HikariPool.java, {’Andreas Brenk’: ’1’}, {"Arthur D’Andr\xc3\xa9a Alemar": ’1’}, {’Brett Wooldridge’: ’167’},
{’Guido Medina’: ’1’}, {’Johannes Herr’: ’1’}, {’Johno Crawford’: ’2’}, {’Jungtaek Lim’: ’1’}, {’Manabu Matsuzaki’: ’1’},
{’Martin St\xc5\x99\xc3\xad\xc5\xbe’: ’1’}, {’Mikhail Mazurskiy’: ’4’}, {’Muruganantham Mani’: ’3’}, {’NersesAM’: ’1’},
{’Nitin’: ’85’}, {’nothing’: ’1’}, {’shendley’: ’3’}, {’Stephan Schroevers’: ’1’}, {’Will Vuong’: ’2’}, {’yaojuncn’: ’4’};
PrometheusMetricsTracker.java, {’Aleksandr Podkutin’: ’1’}, {’Brett Wooldridge’: ’2’}, {’cenk ak\xc4\xb1n’: ’2’},
{’Lorenzo Lucherini’: ’1’}, {’puntogil’: ’1’}, {’shendley’: ’1’}, {’Stanislav Savulchik’: ’1’},
{’Tokuhiro Matsuno’: ’1’}, {’Tom Hombergs’: ’1’};
TestConnectionTimeoutRetry.java, {’Brett Wooldridge’: ’12’}, {’Nitin’: ’3’}, {’puntogil’: ’1’};
ClockSourceTest.java, {’ams2990’: ’1’}, {’Brett Wooldridge’: ’2’};
CodahaleHealthChecker.java, {’Brett Wooldridge’: ’3’};
ProxyLeakTaskFactory.java, {’Andreas Brenk’: ’1’};
TestMetrics.java, {’Brett Wooldridge’: ’6’}, {’NersesAM’: ’1’}, {’Ruben Dijkstra’: ’1’}, {’shendley’: ’1’};
.settings/org.eclipse.core.resources.prefs, {’Brett Wooldridge’: ’1’};
ConnectionStateTest.java, {’Brett Wooldridge’: ’5’}, {’puntogil’: ’1’};
TestValidation.java, {’Brett Wooldridge’: ’7’}, {’Nitin’: ’8’};
.settings/org.eclipse.m2e.core.prefs, {’Brett Wooldridge’: ’1’};
TestJNDI.java, {’Brett Wooldridge’: ’6’}, {’Nitin’: ’3’};
codecov.yml, {’Brett Wooldridge’: ’2’};
ProxyCallableStatement.java, {’Brett Wooldridge’: ’2’};
MockDataSource.java, {’Brett Wooldridge’: ’7’}, {’Guillaume Smet’: ’1’}, {’Kazuhiro Sera’: ’1’};
RampUpDown.java, {’Brett Wooldridge’: ’4’}, {’Nitin’: ’2’}, {’puntogil’: ’1’}, {’shendley’: ’1’}, {’Stephan Schroevers’: ’1’};
module-info.java, {’Brett Wooldridge’: ’1’}, {’Gili Tzabari’: ’3’}, {’Prashant Bhat’: ’1’};
TestFastList.java, {’Brett Wooldridge’: ’1’};
TestObject.java, {’fpirson’: ’1’}, {’freafrea’: ’1’};
.settings/org.eclipse.jdt.core.prefs, {’Brett Wooldridge’: ’1’};