Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Identificação de Habilidades no Código-Fonte

Mara de Lemos Gomes

JUIZ DE FORA
MARÇO, 2025

Identificação de Habilidades no Código-Fonte

Mara de Lemos Gomes

Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Gleiph Ghiotto Lima de Menezes

Coorientador: Alessandreia Marta de Oliveira Julio

JUIZ DE FORA

MARÇO, 2025

Identificação de Habilidades no Código-Fonte

Mara de Lemos Gomes

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Gleiph Ghiotto Lima de Menezes
Doutor em Computação - UFF

Alessandreia Marta de Oliveira Julio
Doutora em Computação - UFF

André Luiz de Oliveira
Doutor em Ciências da Computação e Matemática Computacional - USP

Leonardo Vieira dos Santos Reis
Doutor em Ciências da Computação - UFMG

JUIZ DE FORA
11 DE MARÇO, 2025

Resumo

A análise automatizada do código-fonte para identificar habilidades técnicas e comportamentais dos desenvolvedores surge como uma solução eficaz para superar as limitações da avaliação manual, que é pouco escalável e sujeita à subjetividade humana. Enquanto abordagens anteriores se concentram na análise de mensagens de *commit* ou na extração de dados do código sem considerar o contexto das modificações, esta monografia propõe avaliar as linhas adicionadas por cada desenvolvedor ao longo do histórico de desenvolvimento de projetos. As habilidades técnicas são determinadas pelas invocações de métodos presentes nessas linhas de código. As habilidades comportamentais são mensuradas de duas formas: o comprometimento é determinado pela quantidade de linhas inseridas ao longo do tempo, enquanto a colaboração é avaliada pela adição de linhas em arquivos previamente modificados por outros desenvolvedores. A avaliação da abordagem foi conduzida utilizando três repositórios populares: Guava, Gson e Guice. Esses repositórios foram escolhidos por serem predominantemente escritos em Java e utilizarem o Apache Maven, que são dependências essenciais para a abordagem proposta, além de apresentarem alta popularidade e possuírem um volume significativo de *commits*. Os resultados revelaram três categorias principais de conhecimento técnico: Java *Base*, Estruturas de Dados e Testes de *Software*. Além disso, os resultados demonstraram que o tempo das contribuições influencia diretamente a proficiência dos desenvolvedores. Os resultados também indicaram que 90,47% dos desenvolvedores contribuem para arquivos modificados por outros, refletindo um alto nível de colaboração. Por outro lado, o comprometimento foi evidenciado em uma parcela menor de desenvolvedores, apenas 32%.

Palavras-chave: Desenvolvedores de *Software*, Análise de Repositórios de Código-Fonte, Habilidades Técnicas, Habilidades Comportamentais.

Abstract

Automated source code analysis for identifying developers' hard and soft skills emerges as an effective solution to overcome the limitations of manual evaluation, which is not easily scalable and is subject to human subjectivity. While previous approaches have focused on analyzing commit messages or extracting code data without considering the context of modifications, this monograph proposes evaluating the lines added by each developer throughout the project's development history. Hard skills are determined based on the method invocations present in these lines of code. Soft skills are measured in two ways: commitment is determined by the number of lines inserted over time, whereas collaboration is assessed by the addition of lines to files previously modified by other developers. The evaluation of the approach was conducted using three popular repositories: Guava, Gson, and Guice. These repositories were chosen for being predominantly written in Java and using Apache Maven, which are essential dependencies for the proposed approach, as well as having high popularity and a significant volume of commits. The results revealed three main categories of technical knowledge: Java Basics, Data Structures, and Software Testing. Additionally, the results demonstrated that the timing of contributions directly influences developers' proficiency. The findings also indicated that 90.47% of developers contribute to files modified by others, reflecting a high level of collaboration. Conversely, commitment was observed in a smaller proportion of developers, accounting for only 32%.

Keywords: Software Developers, Analysis of Source Code Repositories, Hard Skills, Soft Skills.

Agradecimentos

Quero expressar minha profunda gratidão a todos que estiveram ao meu lado ao longo da minha trajetória acadêmica. Em primeiro lugar, agradeço a Deus por ter sido meu guia nas adversidades, iluminando meu caminho e me fortalecendo nos momentos mais difíceis.

Agradeço também aos membros da minha família: minha mãe, Neusa, pelo amor incondicional; meu pai, Braz, pelo sustento; e minhas irmãs, Thalita e Miriam, pelo exemplo. Sou igualmente grata às minhas primas, Aline, Carine, Marina e Vitória, e às minhas tias, Aparecida, Regina e Rosimar, assim como às amigas Ester, Laiza e Mayara, que sempre estiveram ao meu lado.

Na universidade fui abençoada com o apoio e o incentivo de pessoas verdadeiramente especiais. Um agradecimento especial vai para meus professores orientadores, Gleiph e Alessandreia, por me ajudarem a persistir, e para os amigos que fiz, Ketleen e Lucas.

Muitas outras pessoas contribuíram para minha jornada, e embora algumas não estejam mais presentes, suas palavras e gestos de encorajamento deixaram uma marca duradoura em meu crescimento pessoal e profissional.

“Quanto mais habilidades você desenvolve, mais você desenvolve a habilidade de desenvolver novas habilidades.”.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Lista de Abreviações

CE	Critérios de Exclusão
MME	Média Móvel Exponencial
QP	Questões de Pesquisa
SCV	Sistema de Controle de Versões
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação

1 Introdução

A identificação das habilidades dos desenvolvedores de *software* por meio da análise do código-fonte é um tema amplamente discutido na literatura (GREENE; FISCHER, 2016; JARUCHOTRATTANASAKUL et al., 2016; KINI; TOSUN, 2018; MONTANDON; SILVA; VALENTE, 2019; ALKHAZI et al., 2020; ATZBERGER et al., 2022; SONG et al., 2022; OLIVEIRA, 2022). Em um contexto em que a produção de *software* se torna cada vez mais complexa e dinâmica, entender as competências dos desenvolvedores é crucial para otimizar processos organizacionais, como recrutamento, formação de equipes e alocação de tarefas (GREENE; FISCHER, 2016; SONG et al., 2022; MONTANDON; SILVA; VALENTE, 2019; OLIVEIRA, 2022).

As habilidades podem ser divididas em duas categorias: técnicas e comportamentais. As habilidades técnicas envolvem competências adquiridas por meio de formação acadêmica, experiência profissional ou prática, como o domínio de uma linguagem de programação (MARTINS, 2017). Já as comportamentais são características pessoais e interpessoais, como o comprometimento e a colaboração (ALEX, 2009).

Parte dos estudos tem como objetivo a identificação de habilidades técnicas por meio de mensagens de *commit*, que podem ser inconsistentes ou até mesmo ausentes (ALKHAZI et al., 2020; GREENE; FISCHER, 2016; SONG et al., 2022). Alguns estudos que extraem dados diretamente do código-fonte, como bibliotecas e linhas adicionadas, apresentam uma abordagem promissora, mas ainda carecem de uma análise mais detalhada (ATZBERGER et al., 2022; OLIVEIRA, 2022; MONTANDON; SILVA; VALENTE, 2019). Quanto às habilidades comportamentais, poucos estudos as abordam (SONG et al., 2022; KINI; TOSUN, 2018), e os que o fazem limitam-se a analisar apenas a colaboração, sem expandir para outros aspectos, como o comprometimento.

Diante desse cenário, novas abordagens que aprimorem as metodologias existentes, integrando a análise de habilidades técnicas e comportamentais, podem complementar as práticas já consolidadas, proporcionando uma avaliação mais abrangente e precisa do perfil dos desenvolvedores.

1.1 Motivação

A identificação de habilidades por meio da análise do código-fonte tem aplicações práticas significativas. Para empresas de Tecnologia da Informação (TI), esse processo pode ser utilizado estrategicamente em processos seletivos, proporcionando uma avaliação mais precisa das habilidades dos candidatos (GREENE; FISCHER, 2016; SONG et al., 2022). Dessa forma, supera as limitações dos métodos tradicionais, como entrevistas e análise de currículos, que podem ser influenciados por subjetividade humana e informações imprecisas (CHIAVENATO, 2009; JARUCHOTRATTANASAKUL et al., 2016). Além disso, a análise do código-fonte contribui para uma gestão mais eficiente das equipes, permitindo a identificação de pontos fortes e áreas de melhoria dos profissionais. Isso facilita a redistribuição de tarefas conforme as especialidades de cada desenvolvedor, otimizando a produtividade e a colaboração (SONG et al., 2022; MONTANDON; SILVA; VALENTE, 2019; OLIVEIRA, 2022).

1.2 Descrição do Problema

A análise manual do código demanda tempo, esforço e conhecimento aprofundado das tecnologias utilizadas (ATZBERGER et al., 2022). Essa dependência de avaliação humana limita significativamente a escalabilidade do processo, tornando-o impraticável para projetos de grande porte ou organizações que precisam avaliar muitos desenvolvedores. Além disso, a subjetividade inerente à análise humana pode levar a variações nos critérios de avaliação, comprometendo a padronização e a confiabilidade dos resultados (CHIAVENATO, 1999). Diante desse cenário, a adoção de uma abordagem automatizada se torna essencial, pois permite a execução de tarefas repetitivas com maior consistência e eficiência, reduzindo o impacto de vieses humanos e aumentando a precisão dos resultados.

1.3 Objetivos

Este Trabalho de Conclusão de Curso (TCC) visa identificar, por meio da análise de código-fonte, as habilidades técnicas e comportamentais dos desenvolvedores de forma

automatizada, proporcionando uma abordagem objetiva e baseada em evidências concretas extraídas do próprio processo de desenvolvimento.

No que se refere à identificação das habilidades técnicas, o foco está na análise de invocações de métodos de classes de bibliotecas externas utilizadas pelos desenvolvedores ao longo do histórico de desenvolvimento de *softwares*. Essas classes são utilizadas como indicadores do conhecimento técnico do desenvolvedor. Por exemplo, ao examinar o código-fonte, é possível verificar o uso de classes específicas, como `java.sql.Connection` e `java.sql.ResultSet`, que estão relacionadas a Banco de Dados e indicam o conhecimento do desenvolvedor nessa área.

Em relação à análise das habilidades comportamentais, adota-se uma abordagem quantitativa com foco em duas habilidades: comprometimento e colaboração. O comprometimento do desenvolvedor com o projeto é medido através da quantidade de linhas de código adicionadas ao repositório por ele, representando seu engajamento com o projeto. A colaboração, entendida como a interação do desenvolvedor com outros membros da equipe, é avaliada pela quantidade de linhas adicionadas em arquivos alterados por outros desenvolvedores.

1.4 Questões de Pesquisa e Resultados Preliminares

Para coletar as habilidades dos desenvolvedores, foi implementada uma prova de conceito, que foi avaliada utilizando três repositórios do time de desenvolvimento do Google — Guava, Gson e Guice. Esses repositórios foram escolhidos por serem predominantemente escritos em Java e utilizarem o Apache Maven, que são dependências essenciais para a abordagem proposta, além de apresentarem alta popularidade e possuírem um volume significativo de *commits*. A seleção desses projetos visou responder às seguintes questões de pesquisa:

- **QP1: Quais as habilidades técnicas predominantes nos repositórios analisados, com base nas classes externas mais utilizadas, e quais desenvolvedores se destacam nessas áreas?**
- **QP2: Como os desenvolvedores colaboram nos repositórios analisados e**

quais se destacam nesse aspecto?

- **QP3: Como o comprometimento dos desenvolvedores evolui ao longo do tempo e quais se destacam pela dedicação nas contribuições?**

A análise dos três repositórios revelou que as principais competências técnicas dos desenvolvedores estão relacionadas às categorias de Java *Base*, Estruturas de Dados e Teste de *Software*. Em termos de colaboração, observou-se que 90,47% dos desenvolvedores interagem com outros membros da equipe ao adicionar linhas a arquivos previamente modificados por eles, demonstrando um alto nível de colaboração nos projetos. Quanto ao comprometimento, foi identificado que 68% dos desenvolvedores contribuíram nos repositórios uma única vez, indicando um baixo engajamento.

1.5 Organização da Monografia

Esta monografia segue uma estrutura composta por cinco capítulos, além desta introdução. No Capítulo 2, é fornecida a fundamentação teórica necessária para compreender o contexto e as bases sobre as quais o trabalho é construído. O capítulo aborda as habilidades técnicas e comportamentais dos desenvolvedores, sistemas de controle de versões, a compilação de código-fonte e a automação do processo de construção. O Capítulo 3 apresenta a revisão da literatura, o protocolo de pesquisa adotado, a execução da pesquisa, resultados, discussão e ameaças à validade. O Capítulo 4 apresenta a abordagem proposta para a identificação de habilidades no código-fonte. Além disso, são discutidos os métodos para identificar habilidades técnicas e comportamentais, bem como a implementação e persistência dos dados extraídos. O Capítulo 5 é dedicado à avaliação experimental. Ele apresenta as questões de pesquisa que orientam a avaliação experimental, descreve a seleção dos repositórios utilizados para a análise e o processo de coleta de dados, expõe e discute os resultados obtidos e, por fim, aborda as ameaças à validade. Finalmente, o Capítulo 6 oferece a conclusão desta monografia e sugestões para trabalhos futuros. Ele sintetiza os principais resultados da pesquisa e propõe direções para trabalhos futuros na área.

2 Fundamentação Teórica

Este capítulo apresenta conceitos fundamentais para a compreensão desta monografia. A Seção 2.1 define as Habilidades Técnicas e Comportamentais, destacando sua importância para o crescimento profissional. Para identificar essas habilidades, este trabalho utiliza ferramentas como sistemas de controle de versões, que registram a evolução do código e permitem a análise das contribuições dos desenvolvedores. Nesse contexto, a Seção 2.2 explora os Sistemas de Controle de Versões, com ênfase no Git. A Seção 2.3 aborda a Compilação de Código-Fonte, desde a análise léxica até a geração de código de máquina, proporcionando uma visão mais profunda dos processos envolvidos na análise do código-fonte. Em complemento, a Seção 2.4 apresenta o Maven, uma ferramenta de automação essencial para a construção de *softwares*, simplificando a gestão de dependências e o processo de compilação. Por fim, a Seção 2.5 reúne as considerações finais deste capítulo.

2.1 Habilidades Técnicas e Comportamentais

As habilidades de um indivíduo se referem à capacidade de executar tarefas específicas com eficácia e realizar um trabalho de alta qualidade (PATACSIL; TABLATIN, 2017). Elas podem ser classificadas em duas categorias principais: Técnicas e Comportamentais.

As Habilidades Técnicas são competências adquiridas por meio de formação acadêmica, profissional ou experiência prática (MARTINS, 2017). Sgobbi e Zanquim (2020) destacam que essas habilidades são quantificáveis e de aprendizado simplificado. Isso significa que elas podem ser avaliadas por meio de testes, exames ou desempenho em tarefas específicas, e que o processo de aprendizado é estruturado e direto, envolvendo cursos, treinamentos e prática.

Em um ambiente empresarial, as Habilidades Técnicas são fundamentais para o bom desempenho de funções (REIS et al., 2022). Por exemplo, em uma empresa de tecnologia, a proficiência em uma linguagem como Java[1] é crucial para desenvolvedores de

[1] ⟨https://www.oracle.com/br/java/⟩

software. Essa habilidade permite que eles escrevam códigos eficientes e integrem novas funcionalidades às aplicações, contribuindo para o sucesso dos projetos e da empresa.

As Habilidades Comportamentais são características pessoais e interpessoais que desempenham um papel crucial no ambiente de trabalho. Elas incluem habilidades como comunicação, trabalho em equipe, criatividade, tomada de decisão e gestão efetiva do tempo (ALEX, 2009).

Empregadores valorizam as Habilidades Comportamentais por sua importância no desempenho geral dos funcionários e na sua capacidade de interagir e colaborar com colegas e *stakeholders* (MAJID et al., 2012). Entre essas habilidades, este trabalho destaca duas em particular: o comprometimento e a colaboração.

Gagné (2014) descreve o comprometimento como a dedicação e o envolvimento dos funcionários em suas tarefas e objetivos organizacionais. Esse comprometimento é fortemente influenciado pela motivação intrínseca, na qual os indivíduos se dedicam ao trabalho por encontrar significado e satisfação nas tarefas, além de cumprir obrigações. Envolve uma conexão emocional e psicológica com o trabalho, refletindo-se em maior desempenho e satisfação. Profissionais comprometidos demonstram responsabilidade e determinação, contribuindo ativamente para o sucesso da organização e do time.

A colaboração é a habilidade de indivíduos trabalharem ativamente em conjunto para atingir um objetivo comum. Durante esse processo, os participantes constroem conhecimento e significado por meio de diálogo e negociação, resultando em um produto final que reflete suas ações conjuntas e interdependentes. Em vez de simplesmente atuar ao lado dos outros, a colaboração envolve interação dinâmica e a integração das contribuições de cada membro, culminando em um resultado que é fruto da cooperação e das decisões compartilhadas (ROBBINS; JUDGE; JUDGE, 2019).

Em suma, enquanto as Habilidades Técnicas são essenciais para a execução eficiente das tarefas, as Habilidades Comportamentais, como comprometimento e colaboração, são fundamentais para a dinâmica de trabalho e o sucesso das equipes. O equilíbrio entre essas duas dimensões é crucial para o desenvolvimento profissional e para alcançar os objetivos da organização.

2.2 Sistemas de Controle de Versões

Os Sistemas de Controle de Versões (SCV) são ferramentas importantes no processo de desenvolvimento de projetos, pois armazenam e gerenciam as mudanças feitas em artefatos de *software* ao longo do tempo. Essas mudanças são realizadas em arquivos e diretórios, podendo ser classificadas em adições, exclusões e reorganizações (PILATO; COLLINS-SUSSMAN; FITZPATRICK, 2008). Esse histórico permite que os desenvolvedores rastreiem e revertam mudanças quando necessário, como em casos de exclusão acidental de arquivos.

Existem dois tipos principais de SCV: centralizados e distribuídos. Nos SCVs centralizados, como o Subversion[2], há um único repositório central onde todos os arquivos e suas alterações são armazenados. Quando um SCV centralizado está sendo utilizando, os desenvolvedores realizam um fluxo de trabalho baseado em *checkout, commit* e *update* como o ilustrado na Figura 2.2. O comando *checkout* copia os arquivos do repositório central para suas máquinas locais e, após realizar modificações, os desenvolvedores usam o comando *commit* para registrar essas mudanças de volta no repositório central. O comando *update* é utilizado para garantir que a área de trabalho local esteja sincronizada com a versão mais recente do repositório central, já que outros desenvolvedores podem ter feito *commits* depois da cópia (MENEZES, 2016).

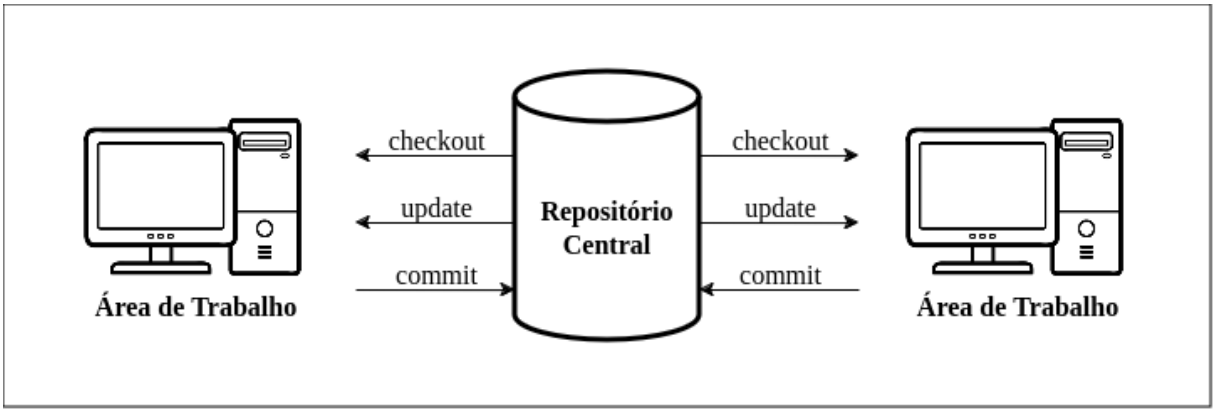

Figura 2.1: Fluxo de trabalho de um SCV centralizado

Por outro lado, os SCV distribuídos, como o Git[3], foram projetados para gerenciar múltiplos repositórios. Em vez de um único repositório central, cada desenvolvedor

[2] ⟨https://subversion.apache.org/⟩
[3] ⟨https://git-scm.com/⟩

mantém uma cópia completa do repositório em sua máquina local, com todo o histórico de mudanças (CHACON; STRAUB, 2014). Isso significa que não há um repositório central obrigatório; os desenvolvedores frequentemente utilizam um repositório principal como referência para compartilhar suas modificações, mas isso não impede que eles realizem e compartilhem modificações diretamente entre si.

A Figura 2.2 ilustra o fluxo de trabalho típico em um SCV distribuído. O processo envolve o clone de um repositório para a máquina local do desenvolvedor, onde ele realiza suas alterações e as registra com o comando *commit*. Em seguida, o desenvolvedor envia suas alterações para o repositório remoto usando o comando *push* e atualiza sua cópia local com as mudanças feitas por outros desenvolvedores usando o comando *pull*.

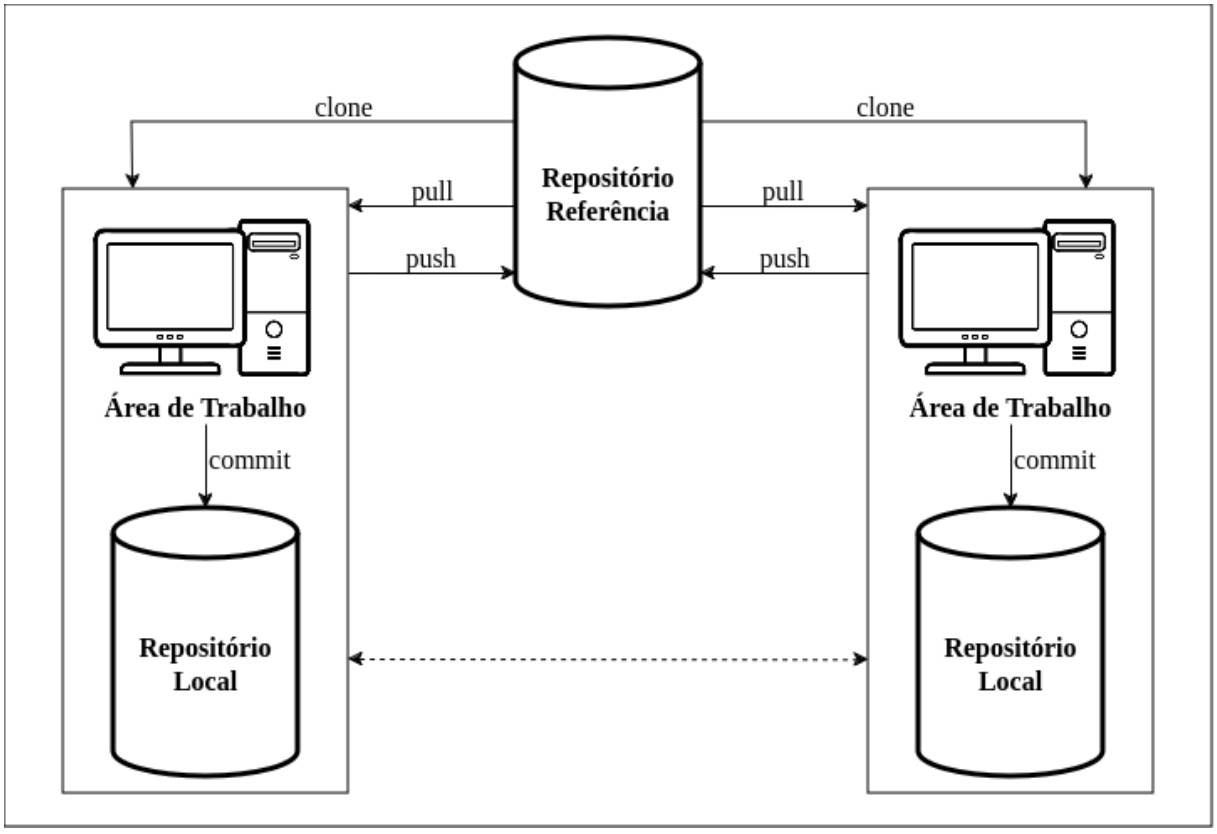

Figura 2.2: Fluxo de trabalho de um SCV distribuído

Um *commit*, além de registrar as modificações realizadas nos arquivos, armazena metadados como o autor que realizou as modificações, e-mail do autor, data e hora e uma mensagem fornecida pelo autor explicando as mudanças (CHACON; STRAUB, 2014). Esses metadados são cruciais para rastrear quem fez quais alterações em determinado ponto do tempo e para entender o motivo das mudanças implementadas.

O Git é um SCV distribuído desenvolvido por Linus Torvalds em 2005, o mesmo

criador do sistema operacional Linux, projetado para fornecer um gerenciamento robusto e eficiente de artefatos de *software*, especialmente em projetos de grande escala e com colaboração entre múltiplos desenvolvedores (CHACON; STRAUB, 2014). Essa seção apresenta um exemplo que ilustra alguns dos principais comandos do Git e suas respectivas funções.

Neste exemplo, suponha que Maria seja uma desenvolvedora e esteja trabalhando em um projeto Java. Com o comando `git log`, que exibe o histórico de *commits* de um repositório, ela verifica que o repositório do projeto contém inicialmente a versão `AAA`. O comando apresenta o identificador da versão (`AAA`), além de outras informações, como o nome (João) e o e-mail do desenvolvedor (joao@mail.com), a data do *commit* (3 de fevereiro de 2024) e a mensagem associada ("feat: cria arquivo Exemplo.java que faz a soma de inteiros"), conforme mostrado na Listagem 2.1.

```
commit AAA
Author: Joao <joao@mail.com>
Date:   Sat Feb 3 09:54:52 2024 -0700

    feat: cria arquivo Exemplo.java que faz a soma de inteiros
```

Listagem 2.1: Exemplo de saída do comando git log

A versão `AAA` possui o arquivo `Exemplo.java`, que realiza a soma de dois números inteiros, `a` e `b`, e atribui o resultado a uma variável chamada `soma` conforme Listagem 2.2.

```
1 public class Exemplo {
2     public static void main(String[] args) {
3         int a = 5, b = 3;
4         int soma = a + b;
5     }
6 }
```

Listagem 2.2: Arquivo `Exemplo.java` na versão `AAA`

A desenvolvedora adiciona uma instrução no arquivo `Exemplo.java` para imprimir o valor de `soma` e altera o valor de `b`, conforme Listagem 2.3. Após realizar essas mudanças, ela usa o comando `git commit` para salvar as alterações, gerando uma nova versão `BBB`.

```
1 public class Exemplo {
2     public static void main(String[] args) {
3         int a = 5, b = 2;
4         int soma = a + b;
5         System.out.println(soma);
6     }
7 }
```

Listagem 2.3: Arquivo `Exemplo.java` na versão `BBB`

Para visualizar as diferenças entre as versões `AAA` e `BBB` do arquivo `Exemplo.java`, Maria utiliza o comando `git diff AAA BBB -- Exemplo.java`. O resultado deste comando é apresentado na Listagem 2.4. As linhas adicionadas são precedidas com o símbolo de adição "+", enquanto as linhas removidas são precedidas com o símbolo de subtração "-". Linhas modificadas são interpretadas como remoções seguidas de adições.

```
1  public class Exemplo {
2      public static void main(String[] args) {
3 -        int a = 5, b = 3;
3 +        int a = 5, b = 2;
4          int soma = a + b;
5 +        System.out.println(soma);
6      }
7  }
```

Listagem 2.4: Saída do comando `git diff AAA BBB -- Exemplo.java`

Utilizando o comando `git blame BBB -- Exemplo.java`, ela pode verificar que as linhas 1, 2, 4, 6 e 7 estão registradas como desenvolvidas por João, enquanto as linhas 3 e 5 estão como de sua autoria na versão `BBB`. O comando `git blame` mostra, linha por linha, quem foi o último autor a modificar cada linha de um arquivo.

Maria então começa a fazer novas alterações no arquivo. No entanto, ela se arrepende das modificações e decide revertê-las. Ela então usa o comando `git checkout BBB` para voltar o arquivo `Exemplo.java` para o estado apresentado na Listagem 2.3. Com o comando `git checkout`, é possível navegar entre os diferentes *commits* do repositório. Ele restaura o estado do projeto para aquele ponto específico no tempo, descartando quaisquer alterações que ainda não foram confirmadas, ou seja, sem um *commit*.

Para explorar uma gama mais ampla de comandos e casos de uso, a documentação oficial do Git[4] pode ser consultada.

[4] ⟨https://git-scm.com/doc⟩

2.3 Compilação de Código-Fonte

Compreender o funcionamento básico de um compilador é fundamental para entender os procedimentos envolvidos em uma análise detalhada de código-fonte. As principais etapas no processo de compilação são: análise léxica, análise sintática, análise semântica, geração de código intermediário, otimização e geração de código final. Esta seção foi baseada principalmente nas ideias e conceitos apresentados por Aho et al. (2008).

Considere como exemplo o código Java apresentado na Listagem 2.5. Ele apresenta a implementação de uma classe denominada `Soma`, que possui a declaração de dois métodos: `imprimeSoma` e `main`. O método `imprimeSoma` recebe como parâmetros dois números inteiros (`a` e `b`) e imprime na tela o resultado da soma desses dois números. O método `main` é o método principal da classe e invoca o método `imprimeSoma`, passando como parâmetros os números 3 e 5.

```
1 public class Soma {
2     public static void imprimeSoma(int a, int b) {
3         System.out.println(a + b);
4     }
5
6     public static void main(String[] args) {
7         imprimeSoma(3, 5);
8     }
9 }
```

Listagem 2.5: Classe Soma em Java

Na análise léxica, o código-fonte é transformado em uma sequência de unidades de linguagem conhecidas como *tokens*. Esses *tokens* representam as menores partes reconhecíveis da linguagem, como: palavras reservadas (`public`, `class`, `static`, `void`, `int`), identificadores (`Soma`, `main`, `args`, `imprimeSoma`, `System`, `out`, `println`, `a`, `b`), operadores (`+`), e símbolos especiais (`(`, `)`, `;`, `.`, `[`, `]`).

Na análise sintática, o objetivo é verificar se os *tokens* gerados durante a análise léxica estão organizados de acordo com as regras gramaticais da linguagem de programação. Durante essa fase, o compilador estrutura os *tokens* em uma árvore hierárquica, conhecida como Árvore de Análise Sintática ou Árvore de Derivação. Esta árvore representa a organização e a estrutura sintática do código, refletindo como as diferentes partes do código se relacionam e se agrupam conforme as regras da gramática da linguagem.

A análise sintática garante que o código-fonte esteja corretamente estruturado e siga a sintaxe esperada, preparando-o para as etapas subsequentes de análise e compilação.

A Listagem 2.6 apresenta um trecho da gramática da linguagem Java[5], relacionado à invocação de métodos, que consiste em utilizar um método definido em uma classe. O termo `methodInvocation` descreve a invocação de métodos e funciona como um rótulo para essa regra. O símbolo `:` separa o nome da regra de sua definição, enquanto cada linha separada por "`|`" representa formas diferentes para criação de uma invocação de método.

```
1 methodInvocation
2  : methodName '(' argumentList? ')'
3  | typeName '.' typeArguments? identifier '(' argumentList? ')'
4  | expressionName '.' typeArguments? identifier '('
   argumentList? ')'
5  | primary '.' typeArguments? identifier '(' argumentList? ')'
6  | 'super' '.' typeArguments? identifier '(' argumentList? ')'
7  | typeName '.' 'super' '.' typeArguments? identifier '('
   argumentList? ')'
8  ;
```

Listagem 2.6: Trecho da Gramática da Linguagem Java

Na segunda linha, o trecho `methodName '(' argumentList? ')'` descreve a forma mais básica de invocação de método, na qual um método é invocado pelo seu nome seguido por argumentos entre parênteses. O `argumentList` é opcional, indicando a possibilidade de zero ou mais argumentos passados para o método. A invocação de método `imprimeSoma` na linha 8 da Listagem 2.5 exemplifica essa forma básica. A Figura 2.3 apresenta o trecho da Árvore de Análise Sintática para a mesma invocação de método.

[5] ⟨https://github.com/antlr/grammars-v4/tree/master/java/java9⟩

Figura 2.3: Exemplo de Árvore de Análise Sintática

Após a análise sintática, o compilador realiza uma análise semântica, responsável por verificar se o código respeita as regras e restrições da linguagem, garantindo a coerência e a integridade do programa. Nesta fase, o compilador realiza verificações como a verificação de tipos, que confirma se as operações e funções estão sendo usadas com os tipos de dados apropriados. Por exemplo, é nessa fase que o compilador verifica se os parâmetros 3 e 5 passados na invocação do método `imprimeSoma` são do tipo inteiro esperados pela assinatura do método.

Após a análise semântica, o compilador gera um código intermediário. O código intermediário é uma representação de baixo nível do programa, que não é específico de nenhuma arquitetura de *hardware* e é mais fácil de otimizar e traduzir para o código final.

Em seguida, a etapa de otimização visa melhorar o desempenho do código. Ela modifica o código intermediário para aumentar a eficiência sem alterar o comportamento do programa.

Finalmente, a geração de código final traduz o código intermediário otimizado em código de máquina específico da plataforma. O resultado é um programa que pode ser executado diretamente pelo processador.

Em resumo, o processo de compilação transforma o código-fonte em um formato que pode ser executado pela máquina, passando por diversas etapas que garantem a correção, eficiência e adequação do código para a execução.

2.4 Automação do Processo de Construção com Apache Maven

O processo de construção de *softwares* inclui etapas como a compilação do código, o gerenciamento de dependências, a execução de testes e o empacotamento do *software* para distribuição. A automação desse processo é fundamental para minimizar o trabalho manual dos desenvolvedores, aumentando a produtividade e reduzindo significativamente a ocorrência de erros humanos ao longo do ciclo de desenvolvimento (PRAKASH, 2022).

Por exemplo, ao compilar código Java, o desenvolvedor utiliza o comando `javac` para compilar cada arquivo `.java` e gerar os arquivos `.class` em *bytecode*. Quando uma classe possui dependências, ou seja, outras classes importadas no código-fonte, é necessário organizar a compilação de forma que todas as dependências sejam resolvidas antes. Isso implica que todas as classes das quais a classe principal depende precisam estar previamente compiladas e disponíveis na mesma pasta de saída.

Com um grande número de dependências, o processo manual torna-se custoso e propenso a erros. Nesse cenário, ferramentas de automação de construção se tornam essenciais para otimizar e garantir a eficiência da compilação.

O Apache Maven[6] é um exemplo de ferramenta de automação de construção, popular em projetos Java, desenvolvida pela Apache Software Foundation. O Maven utiliza um arquivo de configuração denominado "pom.xml" (*Project Object Model*) que centraliza as informações do projeto, como nome, versão e organização, além de gerenciar dependências e outras configurações[7].

O ciclo de construção do Maven é composto por várias fases, incluindo *validate* (valida se o projeto está correto e se todas as informações necessárias estão disponíveis), *compile* (compila o código-fonte), *test* (executa os testes), *package* (faz o empacotamento do código-fonte em um artefato, como JAR ou WAR), *install* (faz a instalação do artefato no repositório local) e *deploy* (implanta o artefato em um repositório remoto)[8].

O Maven opera através da execução de *plugins*, que adicionam funcionalidades específicas ao processo de construção. Esses *plugins* podem realizar diversas tarefas, como compilar o código-fonte, gerar documentação, realizar análise de código e muito mais[9]. Essa capacidade de adicionar e personalizar funcionalidades permite adaptar o processo de construção às necessidades específicas de cada projeto.

O Maven adota várias convenções como boas práticas para organização dos projetos. Por exemplo, ele estabelece uma estrutura de diretórios padrão, com `src/main/java` para o código-fonte Java, `src/test/java` para o código de teste, e `target` para os arquivos gerados durante o processo de construção, incluindo arquivos compilados e artefatos empacotados. O arquivo de configuração principal deve ser nomeado "pom.xml" e estar localizado na raiz do projeto[10]. As dependências devem ser declaradas no "pom.xml", permitindo que o Maven as baixe de repositórios de bibliotecas, como o Maven Central[11].

Em resumo, o Maven define um padrão eficiente para o processo de construção, automatizando tarefas e reduzindo o esforço manual dos desenvolvedores.

[6] ⟨https://maven.apache.org/⟩
[7] ⟨https://maven.apache.org/pom.html⟩
[8] ⟨https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html⟩
[9] ⟨https://maven.apache.org/plugins/⟩
[10] ⟨https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html⟩
[11] ⟨https://central.sonatype.com/⟩

2.5 Considerações Finais

Este capítulo destacou a importância do equilíbrio entre Habilidades Técnicas e Comportamentais. As Habilidades Técnicas são essenciais para a execução de tarefas e a resolução de problemas técnicos, enquanto as Habilidades Comportamentais, como comprometimento e colaboração, são vitais para manter um ambiente de trabalho produtivo e agradável.

Foram apresentados os Sistemas de Controle de Versões como uma solução eficiente para trabalhos colaborativos, dando destaque ao Git, um SCV distribuído. Foram discutidos alguns dos principais comandos do Git com um exemplo prático.

Na seção sobre Compilação de Código-Fonte, foram apresentadas as etapas do processo de compilação, desde a análise léxica até a geração do código final. Compreender essas etapas é crucial para entender a análise de código-fonte discutida posteriormente nesta monografia.

Finalmente, o Maven foi apresentado como uma ferramenta que facilita o processo de construção, automatizando o ciclo de vida da construção e o gerenciamento de dependências, promovendo consistência e eficiência no desenvolvimento de *software*.

Em resumo, o capítulo oferece uma visão abrangente dos requisitos e ferramentas essenciais para o desenvolvimento de *software*.

3 Revisão da Literatura

Para investigar as metodologias e técnicas relacionadas à análise de habilidades nos repositórios de código-fonte, foi realizada uma revisão da literatura. Embora não siga estritamente o formalismo de uma revisão sistemática, este trabalho adota uma abordagem parcialmente sistemática, com o objetivo de garantir uma pesquisa abrangente e estruturada. A Seção 3.1 descreve o protocolo de pesquisa adotado para a revisão. A Seção 3.2 aborda o processo de execução da revisão. A Seção 3.3 apresenta os resultados obtidos. A Seção 3.4 inclui uma discussão sobre os achados da revisão. A Seção 3.4 aborda as ameaças à validade. Finalmente, a Seção 3.6 discute as considerações finais.

3.1 Definição do Protocolo de Pesquisa

Conforme descrito anteriormente, na Seção 1.3 do Capítulo 1, o objetivo desta pesquisa é identificar as habilidades dos desenvolvedores de *software* por meio da análise automatizada dos repositórios de código-fonte. Para isso, esta revisão tem como foco identificar como os estudos definem o conceito de habilidade e compreender as técnicas utilizadas para extração de habilidades nos repositórios de código-fonte.

A fim de obter as informações desejadas, esse estudo possui as seguintes questões de pesquisa (QP):

QP1: Como as habilidades técnicas são identificadas em repositórios de código-fonte?

Essa pergunta busca esclarecer quais tipos de dados são utilizados para definir as habilidades técnicas dos desenvolvedores presentes em repositórios de código-fonte. Por exemplo, as classes empregadas pelos desenvolvedores podem ser indicativas de suas habilidades técnicas.

QP2: Como as habilidades comportamentais são identificadas em repositórios de código-fonte?

Essa pergunta busca esclarecer quais tipos de dados são utilizados para definir as habilidades comportamentais dos desenvolvedores presentes em repositórios de código-fonte. Por exemplo, a quantidade de linhas adicionadas por um desenvolvedor pode ser um indicativo de seu comprometimento.

QP3: Quais são as técnicas e ferramentas utilizadas?

Essa pergunta explora as metodologias, ferramentas e técnicas específicas aplicadas para analisar repositórios de código-fonte e extrair as habilidades dos desenvolvedores. Por exemplo, o uso de ferramentas como Git para análise de histórico de *commits*.

Duas bibliotecas digitais foram escolhidas para a busca de trabalhos que respondam as questões de pesquisa: *Scopus*[12] e *IEEE Xplore*[13]. Essas fontes foram selecionadas devido ao acesso facilitado pela instituição dos autores, à ampla disponibilidade de artigos em inglês e português — idiomas mais familiares aos autores — e por possuírem escopo na área da Ciência da Computação (QUINTELA et al., 2024).

Para definir a *string* de busca utilizada nas bibliotecas digitais, foi aplicado o acrônimo PICO (*Population*, *Intervention*, *Comparison*, and *Outcome*), um método comumente empregado em revisões ou mapeamentos sistemáticos da literatura com esse propósito (FRANDSEN et al., 2020). A Tabela 3.1 apresenta a aplicação do acrônimo PICO no contexto desta revisão.

[12] ⟨https://www.scopus.com/⟩
[13] ⟨https://ieeexplore.ieee.org/⟩

Tabela 3.1: Aplicação do acrônimo PICO

Letra	**Significado**	**Aplicação**
P	População	Pesquisas que visam identificar as habilidades dos desenvolvedores de *software*.
I	Intervenção	Técnicas que utilizam a análise de repositórios de código-fonte.
C	Comparação	Não se aplica pois não há o objetivo de se comparar abordagens.
O	*Outcome*/Saída	Técnicas, Abordagens, Métodos, Metodologias ou Ferramentas que utilizam a análise de código-fonte para identificação de habilidades.

A partir da definição do PICO, foram determinadas as palavras-chave que representariam a população, a intervenção e a saída. Para ajustar e aprimorar essas palavras-chave, os artigos de Greene e Fischer (2016) e Atzberger et al. (2022) foram escolhidos como ponto de controle. Estes foram previamente analisados e considerados relevantes para os aspectos desta pesquisa, e estão presentes nas duas bibliotecas digitais utilizadas.

Após a realização de testes para verificar a eficácia das palavras-chave em recuperar os artigos de controle, as expressões foram definidas conforme a Tabela 3.2.

Tabela 3.2: Palavras-Chave

Significado	**Palavras-Chave**
População	(*"developers"* OR *"programmers"* OR *"software engineers"* OR *"IT professionals"* OR *"development teams"* OR *"coders"*)
Intervenção	(*"code repository analysis"* OR *"source code analysis"* OR *"code review"* OR *"source code evaluation"* OR *"mining software"* OR *"source code mining"* OR *"software repository mining"* OR *"source code inspection"*)
Outcome/Saída	(*"skills identification"* OR *"skill assessment"* OR *"competency identification"* OR *"competency assessment"* OR *"expertise identification"* OR *"ability identification"* OR *"talent assessment"* OR *"skill evaluation"* OR *"skills evaluation"* OR *"competence identification"* OR *"competence assessment"* OR *"aptitude assessment"* OR *"proficiency assessment"* OR *"capability assessment"* OR *"skill appraisal"* OR *"competency appraisal"* OR *"technical skills"* OR *"coding skills"* OR *"software development skills"* OR *"programming skills"* OR *"software engineering skills"* OR *"developer expertise"* OR *"developer competencies"*)

Na Tabela 3.3, apresenta-se a *string* de busca final a ser utilizada nas bibliotecas digitais. Para a base *Scopus*, a busca é restrita ao título, resumo e palavras-chave, com

o objetivo de obter resultados mais alinhados aos objetivos do estudo e evitar menções indiretas no texto completo. Para a base *IEEE*, a busca é realizada em todo o conteúdo, uma vez que o número de registros retornados nos testes foi consideravelmente menor, permitindo uma abordagem mais abrangente.

Tabela 3.3: *String* de busca

(*"developers"* OR *"programmers"* OR *"software engineers"* OR *"IT professionals"* OR *"development teams"* OR *"coders"*) AND (*"code repository analysis"* OR *"source code analysis"* OR *"code review"* OR *"source code evaluation"* OR *"mining software"* OR *"source code mining"* OR *"software repository mining"* OR *"source code inspection"*) AND (*"skills identification"* OR *"skill assessment"* OR *"competency identification"* OR *"competency assessment"* OR *"expertise identification"* OR *"ability identification"* OR *"talent assessment"* OR *"skill evaluation"* OR *"skills evaluation"* OR *"competence identification"* OR *"competence assessment"* OR *"aptitude assessment"* OR *"proficiency assessment"* OR *"capability assessment"* OR *"skill appraisal"* OR *"competency appraisal"* OR *"technical skills"* OR *"coding skills"* OR *"software development skills"* OR *"programming skills"* OR *"software engineering skills"* OR *"developer expertise"* OR *"developer competencies"*)

A seleção inicial das publicações, realizada por meio da *string* de busca, não assegura que todas as publicações selecionadas sejam relevantes para o contexto da pesquisa. Por isso, o primeiro filtro aplicado após a identificação das publicações preliminares consiste na leitura e análise dos títulos e resumos, utilizando os seguintes critérios de exclusão (CE):

- **CE1:** Documentos como resumos de conferências, editoriais ou artigos de opinião que não são baseados em pesquisa científica.
- **CE2:** Trabalhos que não estejam escritos em português ou inglês.
- **CE3:** Publicações que não tenham sido publicadas dentro do intervalo de 2015 a 2024.
- **CE4:** Artigos que não estão acessíveis em sua totalidade sem custos, caso o acesso ao texto completo seja necessário para a análise.
- **CE5:** Trabalhos que não abordam especificamente da análise de repositórios de código-fonte para extração de habilidades.

Após esse primeiro filtro, a seleção ainda pode conter materiais não totalmente relevantes, uma vez que essa análise é restrita a títulos e resumos. Para refinar ainda mais o conjunto de publicações, um segundo filtro é aplicado. Ele consiste na leitura completa das publicações selecionadas, com base nos mesmos critérios de exclusão mencionados anteriormente.

Este processo visa garantir que apenas os trabalhos que atendem a todos os critérios de relevância sejam considerados na pesquisa. O **Apêndice A - Lista Completa das Publicações Retornadas pela *String* de Busca** apresenta a lista completa dos trabalhos retornados pela *string* de busca e a aplicação dos filtros 1 e 2.

3.2 Execução da Pesquisa

Seguindo o protocolo de pesquisa definido, o estudo foi executado no dia 25 de agosto de 2024 e atualizado em 26 de janeiro de 2025. Foram retornadas **55** publicações distintas das bibliotecas digitais escolhidas, conforme ilustrado na Figura 3.1.

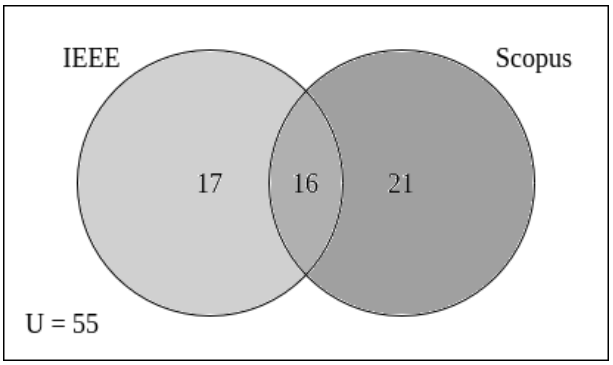

Figura 3.1: Artigos retornados pela *string* de busca

Após a leitura dos resumos, seguindo os critérios de exclusão definidos na Seção 3.1, **18** trabalhos foram selecionados. A Figura 3.2 apresenta a distribuição por biblioteca digital.

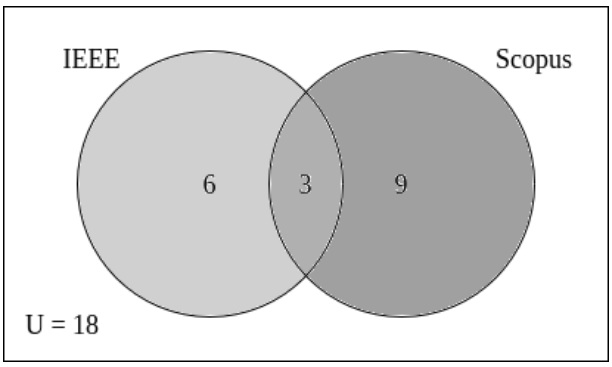

Figura 3.2: Artigos retornados que passaram pelo primeiro filtro

Por fim, foi realizada a leitura completa de todas as **18** publicações e, aplicando os mesmos critérios de exclusão, **8** trabalhos foram classificados como relevantes para esta revisão. A Figura 3.3 apresenta a distribuição final por biblioteca digital.

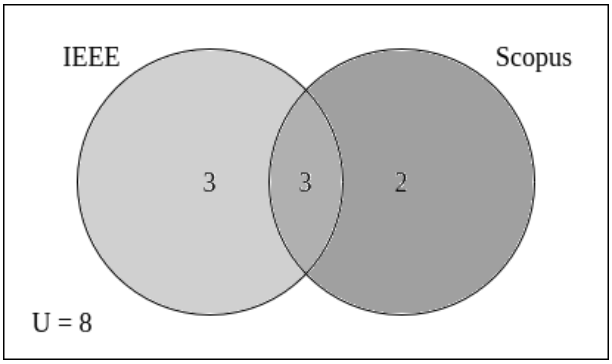

Figura 3.3: Artigos retornados que passaram pelo segundo filtro

3.3 Resultados da Pesquisa

Nesta seção, são apresentadas as respostas para cada uma das questões de pesquisa. O **Apêndice B - Dados Coletados das Publicações Selecionadas** apresenta mais detalhes sobre cada umas das publicações selecionadas.

QP1: Como as habilidades técnicas são identificadas em repositórios de código-fonte?

Os artigos analisados apresentam diversas abordagens, muitas delas concentradas em repositórios do GitHub[14], uma plataforma amplamente utilizada para hospedagem e colaboração em projetos Git.

[14]⟨https://github.com⟩

As mensagens de *commit* são uma fonte importante de análise. Por exemplo, **Alkhazi et al. (2020)** utilizam essas mensagens para sugerir o desenvolvedor mais adequado para corrigir *bugs* em um projeto. **Greene e Fischer (2016)** também analisam mensagens de *commit* e as complementam com a análise de arquivos *README*, buscando palavras-chave que identifiquem conhecimento técnico. Além disso, utilizam a extensão dos arquivos como um indicador direto de proficiência em linguagens de programação. **Song et al. (2022)** expandem o escopo de análise para além do GitHub e das mensagens de *commit*, utilizando *tags* de perguntas e respostas no Stack Overflow[15] para traçar um panorama mais amplo das habilidades dos desenvolvedores.

O conteúdo do código-fonte também é analisado. **Atzberger et al. (2022)** se concentram nos comentários e nos nomes de identificadores dos arquivos para identificar habilidades técnicas. **Oliveira (2022)** analisa as bibliotecas importadas nos arquivos e avalia a proficiência dos desenvolvedores com base na quantidade de linhas de código adicionadas. Seguindo uma abordagem semelhante, **Montandon, Silva e Valente (2019)** investigam três bibliotecas JavaScript específicas, avaliando o número de *commits* em projetos que utilizam essas bibliotecas, a frequência de alterações, o volume de linhas de código adicionadas e removidas e a quantidade de projetos nos quais o desenvolvedor contribuiu.

Em uma linha diferente, **Kini e Tosun (2018)** avaliam o conhecimento do desenvolvedor em um repositório específico, sugerindo que aqueles com maior número de *commits* em um módulo tendem a ter maior propriedade sobre ele, o que pode resultar em código de melhor qualidade.

Por fim, **Jaruchotrattanasakul et al. (2016)** associam as habilidades técnicas dos desenvolvedores às linguagens de programação com as quais eles trabalharam, fornecendo um indicador direto das tecnologias dominadas pelos programadores.

QP2: Como as habilidades comportamentais são identificadas em repositórios de código-fonte?

Apenas dois dos trabalhos analisados exploram a identificação de habilidades

[15] ⟨https://stackoverflow.com/⟩

comportamentais em repositórios de código-fonte. O estudo de **Song et al. (2022)** concentra-se na avaliação da habilidade de colaboração dos desenvolvedores por meio da análise de uma rede construída a partir de suas interações. Nesse modelo, os desenvolvedores são representados como nós, enquanto suas interações — como trocas em perguntas e respostas, revisões de *commits* e projetos compartilhados no GitHub — são representadas pelas arestas entre eles. Para quantificar a importância de cada desenvolvedor na rede, foi utilizado o algoritmo *PageRank*, que atribui maior prioridade aos desenvolvedores que interagem com outros centralizados na rede. Essa abordagem fornece uma métrica baseada nas conexões entre os desenvolvedores e sua posição dentro da rede colaborativa.

Por outro lado, **Kini e Tosun (2018)** avaliam a colaboração analisando a frequência com que diferentes desenvolvedores trabalham juntos no mesmo módulo ou projeto. A colaboração entre os membros da equipe pode impactar diretamente a qualidade do *software*, uma vez que equipes bem integradas e experientes tendem a produzir código mais estável e com menos defeitos. Em contrapartida, equipes com alta rotatividade podem enfrentar desafios maiores, levando a um aumento no número de erros e na necessidade de retrabalho.

Ambos os estudos analisam a habilidade de colaboração por meio de métricas e padrões de interação extraídos dos repositórios de código. Essas abordagens, essencialmente quantitativas e orientadas por dados, possibilitam uma avaliação objetiva da colaboração entre desenvolvedores, baseada em registros concretos do histórico de desenvolvimento.

QP3: Quais são as técnicas e ferramentas utilizadas?

Song et al. (2022) usaram o GitHub e o Stack Overflow para conectar desenvolvedores com base em suas interações, como revisões de código e respostas a perguntas. Eles aplicaram o algoritmo *PageRank* sensível a tópicos para identificar os desenvolvedores mais influentes e usaram o método *k-Nearest Neighbors* (kNN) para comparar habilidades com base em palavras analisadas pelo TF-IDF.

Atzberger et al. (2022) utilizou o modelo *Labeled Latent Dirichlet Allocation* (LLDA) para organizar palavras avançadas e expressões regulares para identificar lingua-

gens e bibliotecas usadas pelos desenvolvedores.

Greene e Fischer (2016) usou a API do GitHub e a biblioteca Eclipse JGit para analisar *commits* e suas mensagens. Eles aplicaram uma lista de palavras-chave para identificar habilidades e criaram uma visualização usando a ferramenta ConceptCloud.

Em **Oliveira (2022)**, habilidades foram avaliadas com métricas como número de arquivos alterados e bibliotecas usadas. Ele classifica os desenvolvedores com base em suas habilidades usando uma heurística para associar código a bibliotecas específicas.

Kini e Tosun (2018) usou *scripts* em PL/SQL para calcular métricas de experiência dos desenvolvedores e aplicou algoritmos de aprendizado de máquina com a ferramenta Weka, como *Random Forest*, *Naive Bayes* e *k-Nearest Neighbors* (kNN), para prever falhas no *software*.

O estudo de **Montandon, Silva e Valente (2019)** usou técnicas de aprendizado de máquina, como *Random Forest*, SVM e *clustering*, para identificar especialistas em JavaScript no GitHub, validando os resultados com perfis do LinkedIn.

Em **Alkhazi et al. (2020)**, foram aplicadas técnicas como *learning-to-rank*, análise de texto com similaridade de cosseno e ferramentas como NLTK e vetorização TF-IDF para avaliar relatórios e mensagens de *commits*.

Por fim, **Jaruchotrattanasakul et al. (2016)** utiliza a API REST do GitHub para coletar dados de perfis, organizados em bancos de dados MySQL. A visualização foi criada com o *framework* Ruby on Rails e gráficos interativos implementados com D3.js.

Em resumo, os estudos utilizam algoritmos de aprendizado de máquina (como *Random Forest*, SVM e kNN), análise de texto (usando TF-IDF, similaridade de cosseno, *stemming* e lematização), ferramentas de mineração de dados (como API do GitHub, PL/SQL, e Weka) e visualização interativa (com D3.js, Ruby on Rails e ConceptCloud).

3.4 Discussão

Os resultados desta revisão de literatura indicam que a análise automatizada de repositórios de código pode ser uma ferramenta valiosa no apoio à avaliação das habilidades dos desenvolvedores, o que reforça a relevância deste estudo ao investigar e propor formas eficazes de extração dessas informações.

Embora parte dos estudos priorizem a análise de mensagens de *commit* como a principal fonte de dados para identificar habilidades técnicas, essas mensagens podem ser inconsistentes, incompletas ou até ausentes, dependendo das práticas adotadas pelos desenvolvedores. Em contrapartida, estudos que extraem dados diretamente do código-fonte, como bibliotecas importadas e linhas adicionadas, apresentam uma abordagem promissora. No entanto, ainda carecem de uma análise mais detalhada, como a verificação de que uma biblioteca importada foi de fato utilizada ou se as linhas adicionadas introduzem alguma nova funcionalidade. Além disso, alguns estudos utilizam algoritmos complexos de aprendizado de máquina, o que pode dificultar a interpretação dos resultados.

Apenas dois estudos abordaram habilidades comportamentais, e ambos se concentraram na habilidade de colaboração, sem considerar o comprometimento dos desenvolvedores com o projeto.

Dessa forma, este trabalho se justifica ao propor uma avaliação mais aprofundada das habilidades técnicas, verificando as inovações de métodos nas linhas adicionadas como evidência do conhecimento dos desenvolvedores sobre uma classe. Também inclui a análise de duas habilidades comportamentais — colaboração e comprometimento, sendo este último um aspecto não abordado nos estudos desta revisão. Além disso, a abordagem proposta oferece um modelo mais simples e de fácil entendimento para avaliar as competências técnicas e comportamentais dos desenvolvedores.

3.5 Ameaças à Validade

Ao conduzir uma revisão da literatura, é fundamental considerar as ameaças à validade, pois podem influenciar a qualidade e a confiabilidade dos resultados. Neste estudo, foram identificadas algumas dessas ameaças, cuja análise é essencial para compreender as limitações e o alcance da generalização dos achados. A seguir, as ameaças são classificadas em ameaças à validade da conclusão, à validade interna ou à validade externa, juntamente com as medidas adotadas para minimizar seus efeitos.

Ameaças à Validade de Conclusão

Os trabalhos selecionados não apresentam uma distinção clara entre habilidades técnicas e habilidades comportamentais, o que pode levar a variações na interpretação dos resultados conforme a perspectiva de cada pesquisador. Para mitigar essa limitação, a classificação das habilidades foi discutida entre os autores, buscando um consenso que reduzisse a subjetividade e garantisse maior rigor na análise.

Ameaças à Validade Interna

Embora os critérios de inclusão e exclusão tenham sido cuidadosamente definidos, é possível que alguns artigos relevantes tenham sido excluídos inadvertidamente. Por exemplo, trabalhos que analisam repositórios de código-fonte, mas com enfoque em outras áreas, como segurança ou desempenho, podem ter sido descartados. Para mitigar esse risco, a revisão foi realizada mais de uma vez, incluindo uma verificação manual dos artigos excluídos para identificar possíveis falsos negativos.

Ameaças à Validade Externa

A limitação linguística da pesquisa, que priorizou artigos em português e inglês, pode ter excluído estudos relevantes em outros idiomas, nos quais a análise de habilidades no código-fonte é abordada de maneira distinta. No entanto, é importante ressaltar que o inglês domina a comunicação científica global. Em 2020, 95% dos artigos científicos foram publicados nesse idioma[16].

Além disso, a escolha de apenas duas bases de dados pode ter limitado a pesquisa, excluindo artigos relevantes de outras fontes. A ausência de fontes adicionais pode ter reduzido a diversidade dos artigos analisados, impactando a completude dos resultados. No entanto, essa limitação foi atenuada pela seleção da base *Scopus*, que indexa estudos de diversas outras bases de busca, ampliando o escopo da pesquisa, e pela inclusão da *IEEE Xplore*, reconhecida por sua relevância nas áreas de tecnologia e engenharia (QUINTELA et al., 2024).

Por fim, a última execução do protocolo de pesquisa ocorreu em 26 de janeiro

[16]https://brasil.elpais.com/ciencia/2021-07-28/em-95-dos-artigos-cientificos-ingles-cria-ditadura-da-lingua-apenas-1-esta-em-portugues-e-espanhol.html

de 2025, refletindo o estado da literatura até essa data. Recomenda-se a replicação do estudo para incorporar possíveis novos artigos publicados ou indexados posteriormente, o que pode impactar a análise e as conclusões obtidas.

3.6 Considerações Finais

A partir da revisão da literatura realizada, foi possível identificar diversas abordagens e técnicas que têm sido exploradas para a análise de habilidades de desenvolvedores em repositórios de código-fonte. Os estudos analisados investigam tanto as habilidades técnicas quanto as comportamentais, utilizando métricas quantitativas e ferramentas de análise de dados para avaliar a *expertise* dos desenvolvedores.

Os estudos abordam desde métricas que analisam as mensagens dos *commits* até algoritmos de aprendizado de máquina. Em termos de ferramentas, o GitHub tem sido amplamente utilizado como fonte de dados, fornecendo uma base rica para a extração de informações sobre as habilidades dos desenvolvedores.

Os resultados da revisão sugerem que a análise automatizada de repositórios de código-fonte pode ser uma abordagem promissora para apoiar a identificação de habilidades. As descobertas deste capítulo contribuem para uma compreensão mais clara dos caminhos possíveis para a identificação e a avaliação de habilidades dos desenvolvedores, apontando para áreas que podem demandar mais investigação e desenvolvimento metodológico.

4 Identificação de Habilidades no Código-Fonte

Neste capítulo, são apresentadas a abordagem proposta para a identificação das habilidades técnicas e comportamentais dos desenvolvedores a partir de repositórios de código-fonte, bem como a implementação da ferramenta automatizada para essa análise. A Seção 4.1 descreve a abordagem empregada, apresentando as métricas definidas para a identificação das habilidades técnicas e comportamentais. A Seção 4.2 descreve a implementação da ferramenta que aplica a metodologia proposta. Por fim, a Seção 4.3 apresenta as considerações finais deste capítulo.

4.1 Abordagem Proposta

Este estudo visa identificar Habilidades Técnicas e Comportamentais dos desenvolvedores em repositórios de código-fonte aberto. O escopo concentra-se na análise dessas habilidades em repositórios que apresentam as seguintes características: terem o código-fonte versionados pelo Git, serem escritos na linguagem Java e utilizarem o Apache Maven para construção.

A escolha do Git se deve à sua capacidade de armazenar informações detalhadas sobre o histórico de mudanças no código. A linguagem Java foi escolhida por ser uma das mais populares entre os desenvolvedores, conforme evidenciado por pesquisas como a do *Stack Overflow Developer Survey*[17]. Além disso, o uso do Maven viabiliza a compilação dos projetos Java, juntamente com suas dependências, automatizadamente.

Foram definidas métricas quantitativas para medir as habilidades dos desenvolvedores. As Habilidades Técnicas visam identificar a familiaridade dos desenvolvedores com classes de bibliotecas externas ao projeto durante o seu desenvolvimento. Elas são avaliadas pelas invocações de métodos nas linhas adicionadas por cada desenvolvedor no

[17] ⟨https://survey.stackoverflow.co/2023/#section-most-popular-technologies-other-tools⟩

histórico de desenvolvimento do projeto. As linhas adicionadas servem como uma medida tangível do trabalho realizado, enquanto as invocações de métodos indicam a interação com classes específicas, fornecendo *insights* valiosos sobre as áreas de conhecimento dos desenvolvedores. A Subseção 4.1.1 mostra a equação matemática utilizada para essa análise.

Para as Habilidades Comportamentais, vislumbra-se identificar o comprometimento do desenvolvedor através das linhas adicionadas nos arquivos, desconsiderando linhas em branco. O número de linhas adicionadas reflete diretamente a sua atividade no projeto. Além disso, a habilidade de colaboração é medida pelas linhas adicionadas em arquivos alterados por outros desenvolvedores. Trabalhar em arquivos compartilhados pode demonstrar a capacidade de um desenvolvedor de colaborar e integrar suas soluções com o trabalho de outros. A Seção 4.1.2 apresenta o cálculo de cada uma dessas habilidades.

A Figura 4.1 ilustra o esquema da abordagem proposta. A partir de um repositório de entrada, obtém-se a lista completa de *commits* de cada desenvolvedor, permitindo a identificação dos arquivos adicionados ou modificados por eles.

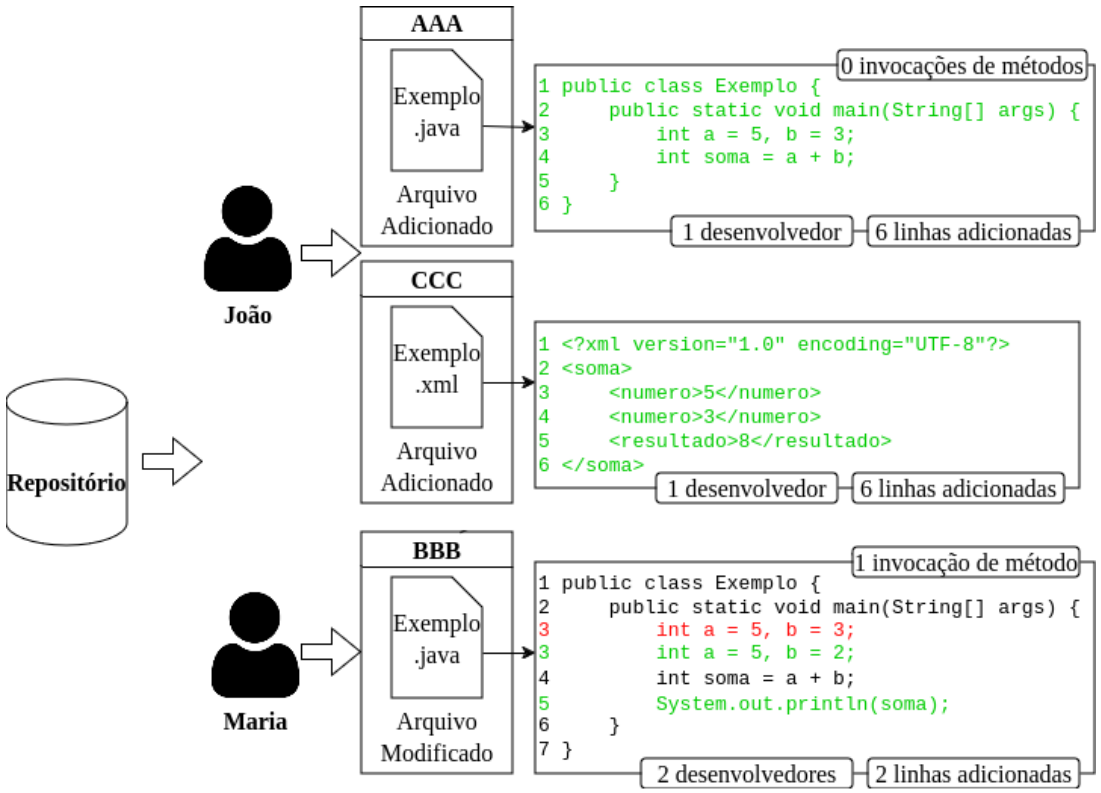

Figura 4.1: Esquema da abordagem proposta

Cada *commit* realizado por cada desenvolvedor é visitado e para cada arquivo alterado são extraídas as linhas adicionadas. As linhas removidas não são consideradas pela análise, pois não estarão na versão resultante gerada pelo desenvolvedor que está sendo analisado.

Por exemplo, o desenvolvedor João é responsável pelas versões `AAA` e `CCC` do repositório. Na versão `AAA`, ele adiciona o arquivo `Exemplo.java`, introduzindo 6 novas linhas. Como não há invocações de métodos nessas linhas, não é possível extrair nenhuma habilidade técnica deste arquivo. Além disso, apenas João modificou o arquivo `Exemplo.java` até o momento, o que impede a inferência de sua habilidade de colaboração. Ao adicionar 6 linhas, ele acumula 6 pontos de comprometimento.

Na versão `CCC`, João adiciona o arquivo `Exemplo.xml`. Embora esse arquivo não seja avaliado para a identificação de habilidades técnicas, ele ainda é considerado na análise das habilidades comportamentais. Assim como no caso anterior, João é o único a modificar o arquivo até o momento, o que não permite a extração de informações sobre sua habilidade de colaboração. Ao adicionar 6 linhas, ele acumula mais 6 pontos na pontuação de comprometimento.

Na versão `BBB`, Maria modifica o arquivo existente `Exemplo.java`. Nessa versão, o arquivo foi alterado por dois desenvolvedores: primeiro por João e depois por Maria, o que sugere a capacidade de colaboração dela. Maria adicionou 2 novas linhas. Essas linhas são analisadas para buscar invocações de métodos e as classes a que pertencem. É identificada na linha 5 a invocação do método `println`, que pertence à classe `System`, indicando o conhecimento de Maria em classes básicas do Java.

O exemplo de habilidade técnica apresentado nessa seção é simples para facilitar o entendimento no leitor. Porém, a abordagem pode ser utilizada para encontrar habilidades técnicas em bibliotecas mais avançadas e amplamente utilizadas no ecossistema de desenvolvimento de *software*, abrangendo áreas como desenvolvimento *web*, inteligência artificial, ciência de dados e segurança da informação.

4.1.1 Identificação de Habilidades Técnicas

As habilidades técnicas dos desenvolvedores são avaliadas por meio de seu conhecimento sobre classes de bibliotecas externas que eles utilizam durante o desenvolvimento do projeto. Para determinar o uso de uma classe, são analisadas as linhas de código que o desenvolvedor adicionou, buscando por invocações de métodos nessas linhas.

Essas classes podem ser analisadas individualmente ou agrupadas em categorias, permitindo que um interessado (recrutador ou gerente de projetos) identifique o conhecimento técnico dos desenvolvedores em diferentes áreas de conhecimento. Por exemplo, as classes `java.sql.Connection`, `java.sql.Statement` e `java.sql.ResultSet` podem ser agrupadas em uma categoria "Banco de Dados"para identificar desenvolvedores com *expertise* nesta área do conhecimento.

Para medir a habilidade do desenvolvedor em cada categoria é utilizada a Média Móvel Exponencial (MME). Essa técnica valoriza as interações mais recentes do desenvolvedor com as classes, uma vez que as tecnologias estão em constante evolução. Ainda assim, ela considera o conhecimento anterior por contribuir para o domínio técnico atual.

A MME é calculada conforme a Equação 4.1. Nesta equação, o termo MME_t representa a Média Móvel Exponencial no tempo t, P_t é o valor atual do dado observado, MME_{t-1} é a MME do período anterior, e N é o número de períodos utilizados para o cálculo da MME.

$$\text{MME(N)}_t = \frac{2}{N+1} \times (P_t - \text{MME(N)}_{t-1}) + \text{MME(N)}_{t-1} \tag{4.1}$$

No início do processo, quando ainda não há uma MME anterior disponível, o primeiro valor da MME é calculado como uma média simples dos primeiros N períodos. Esse valor inicial serve como base para os cálculos subsequentes da MME. A partir do período $N+1$, a MME passa a ser atualizada exponencialmente, ponderando mais fortemente os valores mais recentes, enquanto ainda considera o histórico anterior.

Para exemplificar, considere a Tabela 4.1. Nela, um desenvolvedor realizou a invocação de 5 métodos de uma categoria `C` nos sete primeiros dias (P_1 a P_7). Considerando um período de $N = 7$, a MME inicial ($\text{MME}(7)_1$) é calculada como a média simples

desses valores, resultando em 5,00. A partir do oitavo dia, a MME passa a ser atualizada exponencialmente, utilizando a fórmula da Equação 4.1.

Dia	$\mathbf{P}_t$	$\mathbf{MME}_t$
1	5	-
2	5	-
3	5	-
4	5	-
5	5	-
6	5	-
7	5	5,00
8	0	3,75
9	4	3,81

Tabela 4.1: Valores calculados para MME com N = 7

No oitavo dia ($P_8 = 0$), o desenvolvedor não realizou nenhuma invocação de métodos da categoria `C`, o que faz a MME cair para 3,75. Essa queda sugere que o desenvolvedor perdeu temporariamente o foco ou a necessidade de utilizar essas classes específicas, resultando em uma diminuição na estimativa de conhecimento na categoria. Entretanto, é importante ressaltar que isso não significa que o desenvolvedor perdeu a habilidade, mas apenas que seu uso recente dessas classes diminuiu. A MME reflete a frequência de uso, e não a retenção de conhecimento.

No nono dia ($P_9 = 4$), com a realização de quatro invocações, a MME aumenta ligeiramente para 3,81. Esse aumento sugere que o desenvolvedor retomou o foco na área, demonstrando um reengajamento com as classes da categoria `C`. Esse comportamento dinâmico da MME ilustra como o conhecimento técnico pode flutuar ao longo do tempo, dependendo do envolvimento do desenvolvedor com determinadas tecnologias.

É importante destacar que, se a MME atingir valores próximos de zero, isso não significa que o desenvolvedor perdeu completamente o conhecimento na categoria. Apenas indica que ele não tem utilizado essas classes recentemente.

Em resumo, a MME ilustra a dinâmica do conhecimento do desenvolvedor, destacando momentos de maior aprendizado e fases de menor envolvimento, demonstrando a evolução contínua das habilidades ao longo do tempo.

4.1.2 Identificação de Habilidades Comportamentais

Nesta abordagem são mensuradas duas habilidades comportamentais dos desenvolvedores: comprometimento e colaboração. O comprometimento com o projeto, que reflete o grau de dedicação do desenvolvedor ao progresso do projeto, é quantificado pelo número de linhas de código adicionadas em cada versão, excluindo linhas em branco.

O cálculo do comprometimento é apresentado na Equação 4.2. Nela, o comprometimento do desenvolvedor no tempo t (comprometimento_t) é calculado pela soma do número de linhas adicionadas L_{add}, desconsiderando as linhas em branco, para cada arquivo a em cada uma das versões v. O número total de versões é representado por n, e o número de arquivos modificados em uma versão específica por m.

$$\text{comprometimento}_t = \sum_{v=1}^{n} \sum_{a=1}^{m} L_{add}^{(v,a)} \tag{4.2}$$

Para ilustrar esse cálculo, no exemplo mencionado anteriormente na Seção 4.1, João adicionou 6 linhas ao arquivo `Exemplo.java` na versão `AAA` e mais 6 linhas ao arquivo `Exemplo.xml` na versão `CCC`. Considere a versão `AAA` como sendo de 3 de fevereiro de 2024, enquanto a versão `CCC` como sendo de 20 de abril de 2024.

Se t for o ano de 2024, o comprometimento de João totalizaria 12 pontos. Por outro lado, se considerarmos apenas o mês de fevereiro ou o mês de abril, seu comprometimento seria de 6 pontos em cada mês. Nos demais meses, o comprometimento seria zero, evidenciando a ausência de contribuições, e sugerindo uma falta de envolvimento com o projeto durante esses períodos.

A segunda habilidade comportamental mensurada é a capacidade de colaboração, que reflete a interação do desenvolvedor com outros membros da equipe. Essa habilidade é medida pela quantidade de linhas adicionadas em arquivos alterados por outros desenvolvedores.

A Equação 4.3 indica como a capacidade de colaboração do desenvolvedor é calculada. Nessa equação, *colaboração* representa a capacidade de colaboração do desenvolvedor, v é o índice que corresponde a uma versão, n é a quantidade total de versões, a corresponde a um arquivo com contribuições de mais de um desenvolvedor, e m é o

número total de arquivos que possui contribuição de mais de um desenvolvedor. $L_{add}^{(v,a)}$ é a quantidade de linhas adicionadas pelo desenvolvedor em uma versão v e um arquivo a, desconsiderando linhas em branco, e $L_{totais}^{(v,a)}$ é a quantidade total de linhas em uma versão v e um arquivo a, também desconsiderando as linhas em branco.

$$\text{colaboração} = \sum_{v=1}^{n} \sum_{a=1}^{m} \frac{L_{add}^{(v,a)}}{L_{totais}^{(v,a)}} \tag{4.3}$$

Essa equação calcula a capacidade de colaboração do desenvolvedor, levando em consideração a proporção de linhas que ele adicionou em relação ao total de linhas, para todos os arquivos colaborativos, ao longo de todas as versões do projeto. É importante destacar que a quantidade de linhas adicionadas deve ser inferior à quantidade total de linhas do arquivo, uma vez que reescrever completamente um arquivo de um desenvolvedor não caracteriza colaboração.

No exemplo da Seção 4.1, Maria adicionou 2 linhas ao arquivo `Exemplo.java` na versão `BBB`. O arquivo, originalmente escrito por João, evidencia a interação de Maria com ele. Sua capacidade de colaboração foi aumentada em 0,29 pontos, já que ela adicionou 2 linhas a um arquivo com 7 linhas no total.

4.2 Implementação e Persistência dos Dados

A implementação da ferramenta de análise foi realizada em **Java 8**, escolhido por sua robustez e ampla adoção na indústria. O gerenciamento de dependências e a construção do projeto foram facilitados pelo uso do **Maven**, versão 3.8.7, garantindo uma estrutura modular e eficiente.

A clonagem dos repositórios a serem examinados, extração dos *commits*, associação com os desenvolvedores, identificação dos arquivos modificados e linhas adicionadas é feita por meio do **Git**.

Para cada *commit*, tenta-se realizar a compilação. O analisador foi configurado para trabalhar com quatro versões do JDK: 21 (a versão LTS mais recente), 17, 11 e 8, que são as principais versões apresentadas no site oficial do JDK[18]. Em caso de falha na

[18] ⟨https://www.oracle.com/br/java/technologies/downloads/⟩

compilação, a análise do *commit* é interrompida.

A identificação dos métodos invocados nas linhas adicionadas e suas respectivas classes é realizada por meio da análise do arquivo de *bytecode* (`.class`), utilizando a biblioteca **ASM**[19], versão 9.5. Essa biblioteca foi selecionada devido à sua eficiência e flexibilidade na manipulação do *bytecode* Java, permitindo a extração das invocações de métodos e a associação com suas classes correspondentes.

Os dados extraídos durante o processo de análise são armazenados em um banco de dados **PostgreSQL**[20], versão 42.7.3. A modelagem do banco foi estruturada de forma relacional, permitindo consultas otimizadas e eficientes para análises posteriores.

O fluxo de funcionamento da ferramenta é ilustrado no diagrama de atividades da Figura 4.2. O processo inicia com a clonagem do repositório e a identificação das versões e desenvolvedores. Em seguida, para cada versão, é realizada a compilação do código-fonte. Caso a compilação seja bem-sucedida, o processo avança para a identificação dos arquivos modificados e das linhas adicionadas. Para cada arquivo modificado, são identificados os desenvolvedores presentes e, se o arquivo forem escritos em Java, são extraídas as invocações de métodos e suas respectivas classes. Caso o arquivo não seja escrito em Java, o processo é interrompido para aquele arquivo específico. Após a análise de todos os arquivos modificados, os dados são persistidos no banco de dados. Se a compilação de uma versão falhar, o processo é interrompido para aquela versão, e a análise prossegue para a próxima.

[19] ⟨https://asm.ow2.io/⟩
[20] ⟨https://www.postgresql.org/⟩

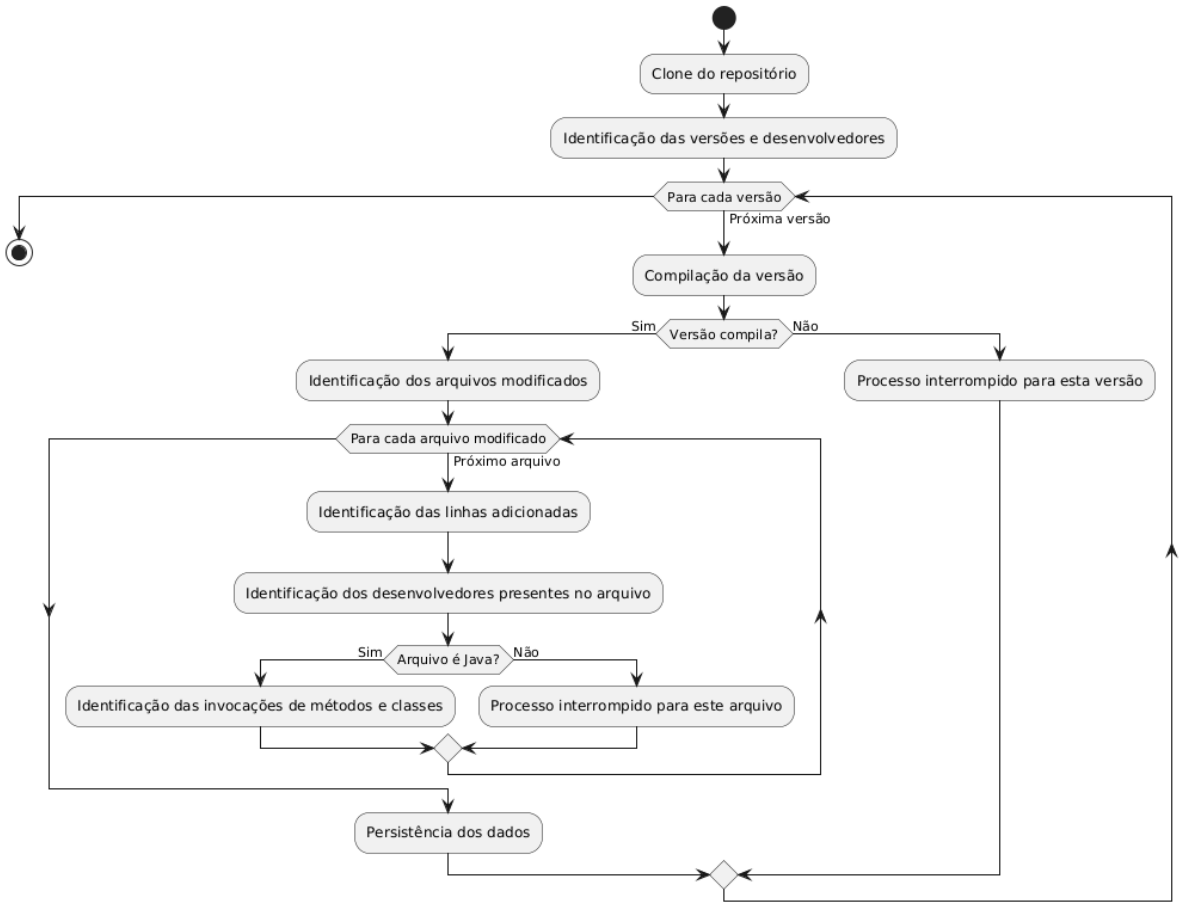

Figura 4.2: Diagrama de atividades do fluxo de funcionamento da ferramenta

Essa organização detalhada do processo de implementação e funcionamento da ferramenta permite uma compreensão clara de cada etapa, desde a coleta de dados até a persistência dos resultados.

4.3 Considerações Finais

Neste capítulo, foi apresentada uma abordagem para a identificação de habilidades técnicas e comportamentais de desenvolvedores a partir da análise de repositórios de código-fonte versionados com Git, escritos na linguagem Java e construídos com Apache Maven. Foram definidas métricas quantitativas para avaliar tanto o conhecimento técnico quanto o comprometimento e a colaboração dos desenvolvedores.

A abordagem proposta oferece uma metodologia estruturada para identificar o conhecimento dos desenvolvedores em classes de bibliotecas externas, utilizando a Média

Móvel Exponencial para capturar a evolução do conhecimento ao longo do tempo. Além disso, a mensuração do comprometimento e da colaboração permite obter uma visão abrangente do envolvimento dos desenvolvedores no projeto, fornecendo *insights* valiosos para recrutadores e gerentes de projeto.

A implementação da ferramenta de análise foi realizada com tecnologias amplamente adotadas, como Java 8, Apache Maven, ASM para análise de *bytecode* e PostgreSQL para armazenamento estruturado dos dados.

Em conclusão, a abordagem apresentada uma alternativa para identificação automática de habilidades técnicas e comportamentais em repositórios de código-fonte, contribuindo para uma melhor compreensão do perfil dos desenvolvedores e para a tomada de decisões em ambientes de desenvolvimento de *software*.

5 Avaliação Experimental

Neste capítulo são apresentados os resultados da avaliação experimental conduzida para validar a abordagem proposta, bem como a análise de seus impactos e limitações. A Seção 5.1 apresenta as questões de pesquisa que orientam a avaliação experimental, destacando os aspectos centrais a serem analisados. A Seção 5.2 descreve a seleção dos repositórios utilizados para a análise e o processo de coleta dos dados. A Seção 5.3 apresenta as respostas obtidas para cada questão de pesquisa. A Seção 5.4 discute os achados da avaliação experimental. Na Seção 5.5 são apresentadas as ameaças à validade, ressaltando as limitações que devem ser consideradas ao interpretar os resultados. Por fim, a Seção 5.6 apresenta as considerações finais deste capítulo.

5.1 Questões de Pesquisa

Esta avaliação experimental visa identificar as habilidades dos desenvolvedores a partir do histórico de desenvolvimento de projetos de *software*. Para guiar a análise, foram estabelecidas as seguintes questões de pesquisa:

QP1: Quais são as habilidades técnicas predominantes nos repositórios analisados, com base nas classes externas mais utilizadas, e quais desenvolvedores se destacam nessas áreas?

Esta questão visa identificar as categorias de classes mais utilizadas nos repositórios analisados, a fim de determinar as habilidades técnicas predominantes entre os desenvolvedores. Com base nas dez classes mais frequentes, serão definidas categorias que representam áreas de conhecimento específicas, como banco de dados, interface gráfica e processamento de dados. Além disso, a pesquisa buscará identificar quais desenvolvedores demonstram maior competência nessas categorias, destacando-se como elementos-chave para a manutenção e evolução dos projetos.

QP2: Como os desenvolvedores colaboram nos repositórios analisados e quais

se destacam nesse aspecto?

Esta questão busca compreender a dinâmica de colaboração entre os desenvolvedores nos repositórios analisados, focando na identificação daqueles que mais contribuem para arquivos previamente modificados por outros. O objetivo é reconhecer os desenvolvedores com maior envolvimento colaborativo, o que pode indicar um papel central na interação e no desenvolvimento coletivo dos projetos.

QP3: Como o comprometimento dos desenvolvedores evolui ao longo do tempo e quais se destacam pela dedicação nas contribuições?

Esta questão busca avaliar o comprometimento dos desenvolvedores ao longo do tempo, com base na quantidade de linhas de código adicionadas. O objetivo é identificar os desenvolvedores com maior engajamento nos repositórios analisados, destacando aqueles que contribuem de maneira consistente.

5.2 *Design* da Avaliação Experimental

5.2.1 Seleção dos Repositórios

Para a execução da avaliação experimental da abordagem proposta, foram selecionados três repositórios do time de desenvolvimento do Google[21] disponíveis no GitHub. Os projetos escolhidos — Guava[22], Gson[23] e Guice[24] — foram selecionados com base nos seguintes critérios: serem escritos predominantemente em Java, utilizarem o Apache Maven, apresentarem alta popularidade e possuírem um volume significativo de *commits*.

Os dois primeiros critérios estão diretamente relacionados ao foco da abordagem proposta, garantindo a compatibilidade com as ferramentas utilizadas na análise. Já o critério de popularidade foi considerado para selecionar projetos amplamente reconhecidos e utilizados pela comunidade. Nesse sentido, os repositórios escolhidos possuem um alto número de estrelas no GitHub, com o Guava alcançando 50,5 mil, o Gson 23,5 mil e o Guice 12,6 mil. Além disso, o critério de quantidade de *commits* contribui para a disponibilidade de uma base de dados robusta para a investigação. O Guava, com 6.687

[21] ⟨https://www.google.com/⟩
[22] ⟨https://github.com/google/guava⟩
[23] ⟨https://github.com/google/gson⟩
[24] ⟨https://github.com/google/guice⟩

commits, o Gson, com 2.062, e o Guice, com 2.123, possuem um histórico significativo de desenvolvimento até a data de corte da análise, em 20 de novembro de 2024.

- O projeto Guava é uma biblioteca de utilitários Java, abrangendo coleções, *cache*, suporte a tipos primitivos, concorrência, anotações comuns, processamento de *strings* e operações com arquivos[25].
- O projeto Gson é uma biblioteca que facilita a conversão de objetos Java para *strings* JSON e vice-versa[26].
- O projeto Guice é um *framework* leve para injeção de dependência em Java, projetado para tornar o código mais modular e testável[27].

A seleção de repositórios de um mesmo time de desenvolvimento visa identificar se os desenvolvedores estão envolvidos em múltiplos projetos, permitindo uma análise mais aprofundada sobre suas habilidades.

5.2.2 Coleta dos Dados

A coleta dos dados foi realizada de forma automatizada. O analisador foi configurado para receber a URL dos repositórios a serem analisados, cloná-los para o servidor local e identificar os *commits* presentes no histórico de cada um. Para cada *commit*, tentou-se realizar a compilação, e aqueles que não foram compilados foram descartados. A Tabela 5.1 apresenta a quantidade de *commits* identificados, a quantidade de *commits* compilados e a respectiva porcentagem de sucesso para cada repositório.

Repositório	**Total de *commits***	***Commits* compilados**	**Porcentagem**
Guava	6.687	4.245	63,5%
Gson	2.062	936	45,4%
Guice	2.123	1.488	70,1%

Tabela 5.1: Quantidade de *commits* compilados por repositório analisado

Para cada arquivo modificado em um *commit* compilado, foram identificadas as linhas adicionadas (excluindo as em branco) e o desenvolvedor responsável pelas alterações.

[25] ⟨https://github.com/google/guava/wiki⟩
[26] ⟨https://github.com/google/gson/blob/main/UserGuide.md#overview⟩
[27] ⟨https://github.com/google/guice/wiki/Motivation#motivation⟩

Em seguida, as invocações de métodos e suas respectivas classes foram extraídas do arquivo *bytecode* (`.class`) correspondente a cada arquivo Java (`.java`), utilizando a associação pelo número da linha.

Por exemplo, a Listagem 5.1 apresenta o arquivo `DirectStackWalkerFinder.java`, incluído no *commit* `cc5511b94ffbd695d42f120727d45c072ebe204f` do repositório Guice. Nesse *commit*, o desenvolvedor Sam Berlin[28] adicionou esse arquivo ao repositório, resultando em 17 novas linhas de código, das quais 14 não estavam em branco. Ao mapear as linhas do respectivo arquivo `DirectStackWalkerFinder.class`, foi possível identificar que, nessas novas linhas, o desenvolvedor realizou um total de nove invocações de métodos. Na linha 8, ocorreu a invocação do método `getInstance` da classe `java.lang.StackWalker`. Já na linha 13, foram identificadas seis invocações: `walk` da classe `java.lang.StackWalker`, `skip` e `filter` da classe `java.util.stream.Stream`, `test` da classe `java.util.function.Predicate`, `getClassName` da classe `java.lang.StackWalker.StackFrame` e `findFirst` da classe `java.util.stream.Stream`. Na linha 14, houve a invocação do método `map` da classe `java.util.Optional`, enquanto na linha 15 foi identificado o método `orElseThrow`, também da classe `java.util.Optional`.

```
1   package com.google.inject.internal.util;
2
3   import java.util.function.Predicate;
4
5   /** A CallerFinder directly compiled against
    StackWalker. Requires compiling against jdk11+. */
6   final class DirectStackWalkerFinder implements CallerFinder {
7     private static final StackWalker WALKER =
8         StackWalker.getInstance(StackWalker.Option.
          RETAIN_CLASS_REFERENCE);
9
10    @Override
11    public StackTraceElement findCaller(Predicate<String>
      shouldBeSkipped) {
12        return WALKER
13            .walk(s -> s.skip(2).filter(f -> !shouldBeSkipped.
                  test(f.getClassName())).findFirst())
14            .map(StackWalker.StackFrame::toStackTraceElement)
15            .orElseThrow(AssertionError::new);
16    }
17  }
```

Listagem 5.1: Arquivo `DirectStackWalkerFinder.java`

[28] ⟨https://github.com/sameb⟩

Os dados coletados foram armazenados em um banco de dados, permitindo a realização dos cálculos definidos na Seção 4.1.

5.3 Resultados

QP1: Quais são as habilidades técnicas predominantes nos repositórios analisados, com base nas classes externas mais utilizadas, e quais desenvolvedores se destacam nessas áreas?

As 10 classes externas mais utilizadas nos repositórios analisados estão listadas na Tabela 5.2. A partir da análise dessas classes, foram identificadas três categorias de habilidades técnicas predominantes:

1. **Java *Base***: Esta categoria engloba habilidades relacionadas à manipulação de estruturas fundamentais da linguagem Java. As classes que sinalizam essa categoria são `java.lang.StringBuilder`, `java.lang.String`, `java.lang.Integer`, `java.lang.Object` e `java.lang.Class`, que são essenciais para operações básicas como manipulação de *strings*, criação de objetos e reflexão.

2. **Estruturas de Dados**: Esta categoria está associada à proficiência em coleções e estruturas de dados fundamentais. As classes `java.util.List`, `java.util.Map`, `java.util.Set` e `java.util.Iterator` refletem essa competência, demonstrando a capacidade dos desenvolvedores em trabalhar com estruturas de armazenamento e recuperação de dados.

3. **Teste de *Software***: Esta categoria destaca a habilidade de implementar testes automatizados. A classe `org.junit.Assert` representa essa competência, mostrando que os desenvolvedores estão aplicando práticas de teste para garantir a qualidade do código e a confiabilidade do *software*.

Classe	Quantidade de Invocações de Método
java.lang.StringBuilder	10.031
java.util.Iterator	2.585
java.util.List	2.015
java.util.Map	1.922
java.lang.Class	1.829
java.lang.Integer	1.756
java.util.Set	1.742
org.junit.Assert	1.273
java.lang.String	1.242
java.lang.Object	1.041

Tabela 5.2: Top 10 classes mais utilizadas nos repositórios analisados

A ampla utilização dessas classes sugere a relevância do domínio técnico nas categorias identificadas para o desenvolvimento dos repositórios analisados. Diante disso, é crucial identificar os desenvolvedores que demonstram maior proficiência nessas áreas, destacando-se como peças-chave no avanço dos projetos.

Para cada categoria, foram selecionados os 10 desenvolvedores que mais invocaram métodos das classes que a compõem. As Tabelas 5.3, 5.4 e 5.5 apresentam esses resultados para as categorias Java *Base*, Estruturas de Dados e Teste de *Software*, respectivamente.

Nome	E-mail	Invocações
-	zhenghua@google.com	1.962
Chris Povirk	cpovirk@google.com	1.280
Luke Sandberg	lukes@google.com	926
Marcono1234	Marcono1234@users.noreply.github.com	858
Sam Berlin	sameb@google.com	856
Maicol Antali	79454487+MaicolAntali@users.noreply.github.com	582
-	limpbizkit@d779f126-a31b-0410-b53b-1d3aecad763e	483
-	clm@google.com	449
-	dhanji@d779f126-a31b-0410-b53b-1d3aecad763e	361
Kurt Alfred Kluever	kak@google.com	302

Tabela 5.3: Top 10 desenvolvedores com mais invocações de método da categoria Java *Base*

Nome	E-mail	Invocações
-	zhenghua@google.com	2.716
Chris Povirk	cpovirk@google.com	1.089
James Sexton	jasexton@google.com	890
Louis Wasserman	lowasser@google.com	273
-	limpbizkit@d779f126-a31b-0410-b53b-1d3aecad763e	270
Marcono1234	Marcono1234@users.noreply.github.com	259
Colin Decker	cgdecker@google.com	244
Sam Berlin	sameb@google.com	230
Kurt Alfred Kluever	kak@google.com	208
Maicol Antali	79454487+MaicolAntali@users.noreply.github.com	136

Tabela 5.4: Top 10 desenvolvedores com mais invocações de método da categoria Estrutura de dados

Nome	E-mail	Invocações
Marcono1234	Marcono1234@users.noreply.github.com	836
Maicol Antali	79454487+MaicolAntali@users.noreply.github.com	175
Éamonn McManus	emcmanus@google.com	58
Wonil	cwi5525@naver.com	36
Inderjeet Singh	inder123@gmail.com	20
Chris Povirk	cpovirk@google.com	12
Stuart McCulloch	mcculls@gmail.com	11
Mihai Nita	nmihai_2000@yahoo.com	11
Sam Berlin	sameb@google.com	7
Jiechuan Chen	654815312@qq.com	6

Tabela 5.5: Top 10 desenvolvedores com mais invocações de método da categoria Teste de *Software*

Utilizando o cálculo da MME apresentado na Seção 4.1.1, com um período de 365 dias (1 ano), foram calculados os pontos de habilidade técnica de cada desenvolvedor para cada categoria. O valor de P_t da equação é a quantidade de métodos invocados de cada categoria por dia. As Tabelas 5.6, 5.7 e 5.8 apresentam os 10 desenvolvedores com maior pontuação no ano de 2024, para as categorias Java *Base*, Estruturas de Dados e Teste de *Software*, respectivamente.

Nome	**E-mail**	**MME 2024**
Chris Povirk	cpovirk@google.com	2,28647
Marcono1234	Marcono1234@users.noreply.github.com	0,32179
Maicol Antali	79454487+MaicolAntali@users.noreply.github.com	0,26929
Andrey Litvitski	120543954+panic08@users.noreply.github.com	0,16852
Sam Berlin	sameb@google.com	0,02009
Chaoren Lin	aoe@google.com	0,01364
Éamonn McManus	emcmanus@google.com	0,01268
Carpe-Wang	78642589+Carpe-Wang@users.noreply.github.com	0,01062
Lyubomyr Shaydariv	lyubomyr-shaydariv@users.noreply.github.com	0,00862
Stefan Haustein	haustein@google.com	0,00828

Tabela 5.6: Top 10 desenvolvedores com maior conhecimento na categoria Java *Base*

Nome	**E-mail**	**MME 2024**
Chris Povirk	cpovirk@google.com	1,85930
Marcono1234	Marcono1234@users.noreply.github.com	0,11770
Chaoren Lin	aoe@google.com	0,04027
Maicol Antali	79454487+MaicolAntali@users.noreply.github.com	0,02154
Andrew Szeto	github@jabagawee.com	0,01343
Martin Kretzschmar	mkretzschmar@google.com	0,00983
Stefan Haustein	haustein@google.com	0,00951
Jean Pierre Lerbscher	jean-pierre.lerbscher@jperf.com	0,00490
Wonil	cwi5525@naver.com	0,00411
Andrey Litvitski	120543954+panic08@users.noreply.github.com	0,00366

Tabela 5.7: Top 10 desenvolvedores com maior conhecimento na categoria Estrutura de Dados

Nome	**E-mail**	**MME 2024**
Marcono1234	Marcono1234@users.noreply.github.com	0,60110
Maicol Antali	79454487+MaicolAntali@users.noreply.github.com	0,06574
Éamonn McManus	emcmanus@google.com	0,01279
Wonil	cwi5525@naver.com	0,01139
Chris Povirk	cpovirk@google.com	0,00470
Mihai Nita	nmihai_2000@yahoo.com	0,00139
Sam Berlin	sameb@google.com	0,00116
Shivam sehgal	sehgals622@gmail.com	0,00108
SP	34406014+sachinp97@users.noreply.github.com	0,00091
Clemens Lieb	vogel612@gmx.de	0,00073

Tabela 5.8: Top 10 desenvolvedores com maior conhecimento na categoria Teste de *Software*

É possível notar uma diferença entre os *rankings* de desenvolvedores que mais invocaram métodos de classes de uma categoria e aqueles que demonstraram maior conhecimento na categoria em 2024.

Os resultados indicam que o tempo das contribuições influencia diretamente o nível de conhecimento técnico. Isso explica por que desenvolvedores, mesmo que tenham utilizado amplamente métodos das classes de uma determinada categoria, não figuram entre os mais conhecedores no ano de 2024.

Por exemplo, o desenvolvedor identificado apenas pelo e-mail `zhenghua@google.com` aparece como o que mais utilizou métodos da categoria Java *Base*. No entanto, ele não figura entre os dez principais desenvolvedores conhecedores da categoria em 2024. Isso ocorre porque, embora tenha realizado contribuições significativas no passado, esse desenvolvedor não interage mais com os métodos dessas classes.

A Figura 5.1 compara a evolução do conhecimento técnico ao longo dos anos desse desenvolvedor com a do desenvolvedor Chris Povirk (`cpovirk@google.com`), que ocupa o primeiro lugar como especialista na categoria Java *Base* em 2024. Nela, é possível observar que `zhenghua@google.com` utilizou métodos das classes pertencentes à categoria em 2017, mas seu conhecimento foi decaindo à medida que deixou de utilizá-los. Por outro lado, Chris Povirk utiliza esses métodos desde 2015 e, em 2024, atingiu seu pico de conhecimento técnico.

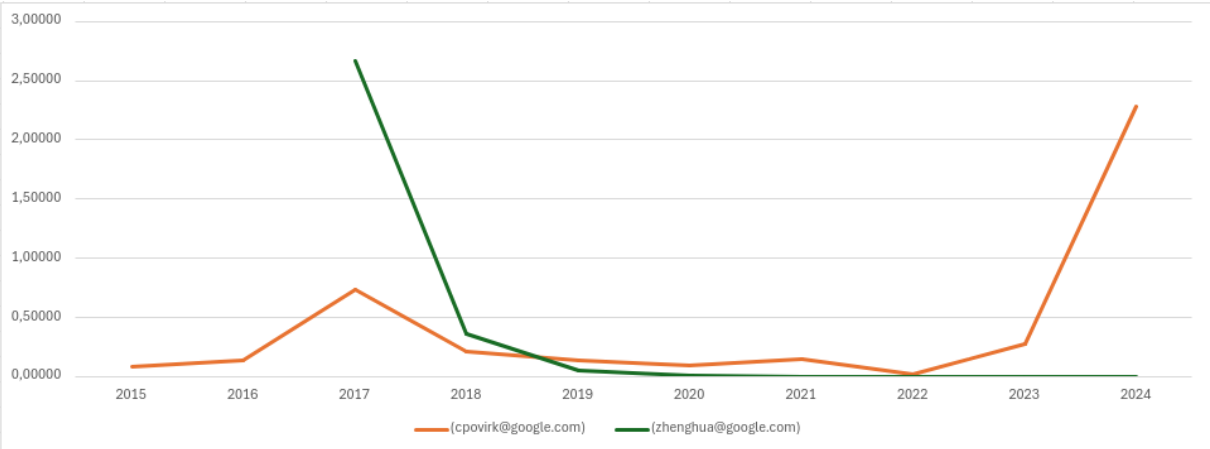

Figura 5.1: Evolução do conhecimento técnico dos desenvolvedores `zhenghua@google.com` e `cpovirk@google.com` ao longo dos anos

Esses achados destacam a eficácia da MME como métrica para identificar desenvolvedores que, apesar de um menor volume de contribuições, têm um impacto recente e significativo no projeto em relação a uma habilidade específica.

QP2: Como os desenvolvedores colaboram nos repositórios analisados e quem se destaca nesse aspecto?

Como esperado em repositórios de código-fonte *open-source*, a maioria dos desenvolvedores apresentou alguma pontuação na habilidade de colaboração, considerando o critério definido na Seção 4.1.2. Dos 724 desenvolvedores presentes nos três repositórios analisados, 655 contribuíram em pelo menos um arquivo previamente modificado por outro desenvolvedor, representando aproximadamente 90,47% do total.

A Tabela 5.9 apresenta os 10 desenvolvedores com maior pontuação em colaboração nos repositórios analisados. Chris Povirk se destaca como o mais colaborativo, obtendo a maior pontuação nessa métrica. Ele contribuiu com 181.320 linhas em arquivos previamente modificados por outros desenvolvedores, estabelecendo colaborações com 485 desenvolvedores ao longo do histórico dos repositórios. Suas contribuições estão concentradas principalmente no repositório Guava (99,9%), enquanto sua participação nos repositórios Guice e Gson é pouco expressiva.

O segundo colocado, Sam Berlin, contribuiu com 47.215 linhas em arquivos, quase quatro vezes menos que Povirk, e colaborou com 184 desenvolvedores. Suas contribuições estão majoritariamente no repositório Guice (99,9%), com participação pouco expressiva no repositório Guava.

Nome	**E-mail**	**Pontuação**
Chris Povirk	cpovirk@google.com	617,96
Sam Berlin	sameb@google.com	150,34
-	limpbizkit@d779f126-a31b-0410-b53b-1d3aecad763e	84,95
Maicol Antali	79454487+MaicolAntali@users.noreply.github.com	73,52
Luke Sandberg	lukes@google.com	68,62
Louis Wasserman	lowasser@google.com	68,02
Marcono1234	Marcono1234@users.noreply.github.com	57,17
James Sexton	jasexton@google.com	53,95
Kurt Alfred Kluever	kak@google.com	50,94
Éamonn McManus	emcmanus@google.com	43,60

Tabela 5.9: Top 10 desenvolvedores mais colaborativos

Povirk não apenas contribuiu com um volume muito maior de linhas de código (quase quatro vezes mais que Berlin), mas também colaborou com um número significativamente maior de desenvolvedores (485 contra 184). Suas contribuições, concentradas no repositório Guava, podem indicar uma especialização nesse projeto, possivelmente desempenhando um papel central na sua manutenção e evolução. Embora Berlin tenha contribuído menos em termos absolutos de linhas de código, sua colaboração com 184 desenvolvedores ainda é relevante. Seguindo a mesma lógica, ele parece ter uma especialização no repositório Guice, desempenhando um papel importante nesse projeto.

QP3: Como o comprometimento dos desenvolvedores evolui ao longo do tempo e quais se destacam pela dedicação nas contribuições?

Entre os 724 desenvolvedores presentes nos três repositórios analisados, 494 realizaram contribuições apenas uma vez, indicando um engajamento pontual. Esse resultado revela um baixo comprometimento por parte de uma parcela significativa dos desenvolvedores. Em contrapartida, os 230 desenvolvedores restantes foram responsáveis por 99,55% das linhas adicionadas, evidenciando um maior comprometimento com os projetos. A Tabela 5.10 apresenta os 10 desenvolvedores mais comprometidos em 2024, destacando também suas contribuições nos anos anteriores (2023 e 2022).

Nome	**E-mail**	**2024**	**2023**	**2022**
Chris Povirk	cpovirk@google.com	31.392	18.198	15.361
Marcono1234	Marcono1234@users.noreply.github.com	4.212	4.622	7.594
Martin Kretzschmar	mkretzschmar@google.com	3.086	749	8
Louis Wasserman	lowasser@google.com	1.118	4	-
Kurt Alfred Kluever	kak@google.com	848	41	933
Stefan Haustein	haustein@google.com	701	2.546	-
Istvan Neuwirth	istvan_neuwirth@epam.com	524	74	-
Chaoren Lin	aoe@google.com	510	314	-
Michael	mboulos95@gmail.com	336	-	-
Jean Pierre Lerbscher	jean-pierre.lerbscher@jperf.com	315	-	-

Tabela 5.10: Top 10 desenvolvedores mais comprometidos em 2024

Chris Povirk lidera o *ranking*, com 31.392 linhas de código adicionadas em 2024, seguido por Marcono1234 e Martin Kretzschmar. A análise revela que, enquanto alguns desenvolvedores mantêm um alto nível de atividade ao longo dos anos, outros apresentam flutuações significativas em suas contribuições.

Focando em Povirk, que apresentou o maior comprometimento em 2024, suas contribuições ao longo dos anos são mostradas na Figura 5.2. Nota-se que o pico de comprometimento ocorreu em 2017. A Figura 5.3 mostra o comprometimento mensal em 2017, indicando que, de janeiro a novembro, as contribuições foram menores, enquanto em dezembro houve um grande pico. A Figura 5.4 mostra que esse pico ocorreu no dia 4 de dezembro de 2017.

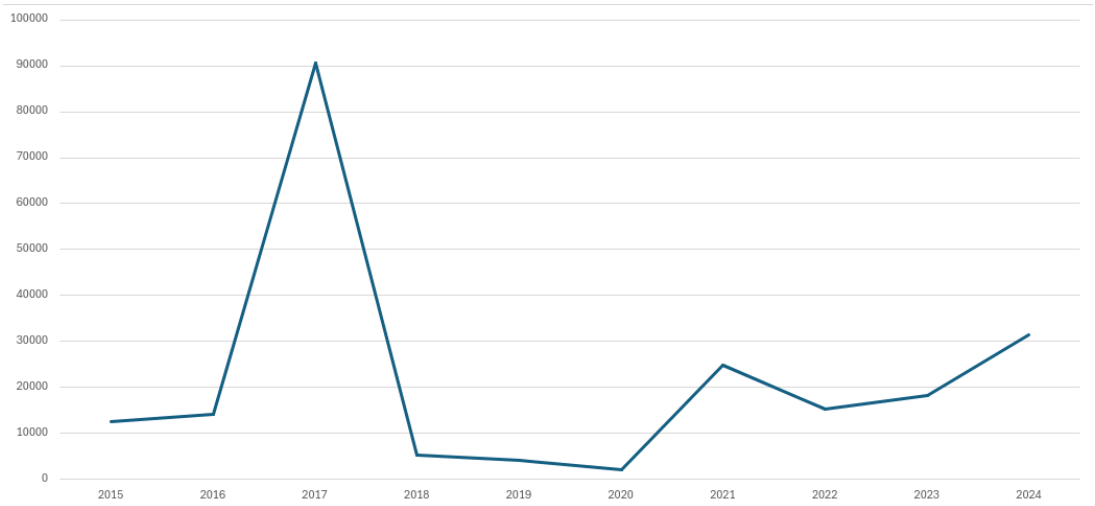

Figura 5.2: Comprometimento anual do desenvolvedor Chris Povirk

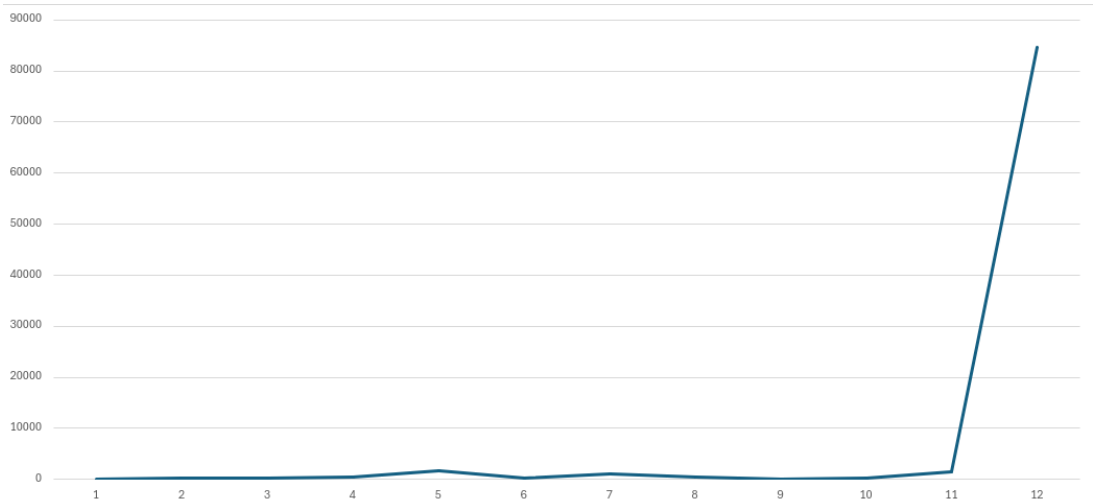

Figura 5.3: Comprometimento mensal do desenvolvedor Chris Povirk no ano de 2017

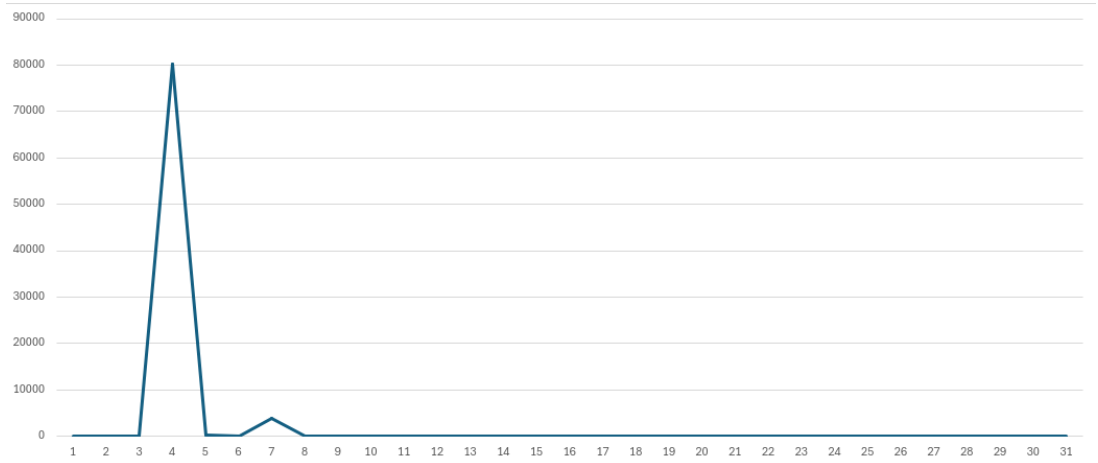

Figura 5.4: Comprometimento diário do desenvolvedor Chris Povirk no mês de dezembro de 2017

Uma análise dos *commits* realizados em 4 de dezembro de 2017, por meio das mensagens associadas, revelou que o alto comprometimento desse dia está relacionado a atividades de refatoração de código. Essas refatorações incluíram melhorias de compatibilidade (ajuste de dependências), correções de segurança, limpeza de código e reformatações para garantir conformidade com os padrões de estilo de codificação. Esse esforço concentrado em um único dia sugere uma dedicação intensa para resolver problemas acumulados ou implementar mudanças significativas no projeto.

5.4 Discussão

Os resultados obtidos nesta pesquisa fornecem uma visão abrangente sobre as habilidades técnicas, a colaboração e o comprometimento dos desenvolvedores nos repositórios analisados. A análise das classes mais utilizadas revelou três categorias predominantes de habilidades técnicas: Java *Base*, Estruturas de Dados e Teste de *Software*, refletindo a importância dessas competências no desenvolvimento dos projetos.

A MME mostrou-se capaz de identificar desenvolvedores com conhecimento atualizado em cada categoria. Por exemplo, Chris Povirk destacou-se como o principal conhecedor em Java *Base* e Estruturas de Dados em 2024, enquanto Marcono1234 liderou na categoria Teste de *Software*. Esses resultados evidenciam que o tempo das contribuições é um fator crucial para o nível de conhecimento técnico, como observado no caso do desenvolvedor `zhenghua@google.com`, que, apesar de ter sido um dos maiores contribuidores

no passado, não figura entre os mais conhecedores em 2024 devido à falta de interação recente com as classes da categoria.

Para o cálculo da MME, P_t foi definido como a quantidade de invocações de métodos de cada categoria por dia. Assim, a MME é ajustada diariamente, refletindo rapidamente o impacto das contribuições mais recentes. Como consequência, se um desenvolvedor parar de contribuir por alguns dias, sua MME pode sofrer uma queda mais acentuada.

Em equipes nas quais os ciclos de trabalho são bem definidos, seria interessante adaptar o período considerado para P_t. Por exemplo, em equipes que operam com ciclos de 30 dias, pode ser mais apropriado considerar P_t como a quantidade de invocações de métodos de cada categoria por mês. Dessa forma, faltas de contribuição de curto prazo (dias isolados) teriam um impacto menor na média.

A colaboração mostrou-se um aspecto central, com 90,47% dos desenvolvedores contribuindo em arquivos previamente modificados por outros. Chris Povirk destacou-se como o mais colaborativo, com uma pontuação significativamente maior que os demais.

A análise do comprometimento revelou que 68% dos desenvolvedores fizeram uma única contribuição nos projetos analisados. Os 32% restantes foram responsáveis por 99,55% das linhas de código adicionadas, demonstrando um papel crucial na evolução dos projetos. Chris Povirk novamente se destacou, com um alto volume de contribuições em 2024 e um histórico consistente ao longo dos anos.

Uma pesquisa adicional na rede social LinkedIn[29] revelou que Chris Povirk é atualmente um Engenheiro de *Software* Sênior no Google, com uma carreira sólida de 17 anos na área de tecnologia. Essa trajetória profissional, aliada à sua atuação em projetos de alto impacto, reforça seu elevado nível de proficiência e relevância no setor, corroborando os achados desta pesquisa e destacando-o como um desenvolvedor-chave para a manutenção e evolução dos projetos analisados.

[29] ⟨https://linkedin.com⟩

5.5 Ameaças à Validade

Esta avaliação experimental foi conduzida com base na análise de três repositórios do time de desenvolvimento do Google. Embora esses projetos sejam amplamente reconhecidos, a quantidade limitada de repositórios analisados pode comprometer a generalização dos resultados. Uma amostra pequena pode não ser suficiente para permitir conclusões robustas sobre o conhecimento técnico ou comportamental dos desenvolvedores.

Em particular, o conhecimento técnico dos desenvolvedores pode não ser completamente refletido nas classes utilizadas nesses projetos, uma vez que elas atendem a requisitos técnicos específicos de cada repositório. Além disso, habilidades comportamentais, como comprometimento e colaboração, embora avaliadas por métricas extraídas do código-fonte, podem não ser plenamente capturadas. Isso ocorre porque esses aspectos também dependem de fatores externos, como processos internos de trabalho e dinâmicas de equipe, que não são diretamente evidentes no histórico de contribuições.

Outro ponto a destacar é que a prova de conceito apresenta limitações técnicas. Como ela depende da compilação bem-sucedida de cada versão do projeto, diversas versões foram excluídas da análise devido a falhas de compilação em todas as versões do JDK testadas. Essa limitação resultou na perda de dados. No repositório Guava, 36% das versões não puderam ser compiladas, enquanto 55% e 30% das versões apresentaram o mesmo problema nos repositórios Gson e Guice, respectivamente.

A não compilação de algumas versões pode ser atribuída à idade dos projetos analisados e às limitações das ferramentas utilizadas. Um exemplo disso é o projeto Guava, no qual foram encontradas versões do código que utilizavam o JDK 1.5, uma versão muito antiga do Java que não recebe atualizações públicas desde 2009[30]. Além disso, o Apache Maven 3.8.6, utilizado para gerenciamento de dependências, não oferece suporte para versões do Java anteriores ao 7. A tentativa de usar o Apache Maven 2.0.11, compatível com o JDK 1.5, também não foi bem-sucedida, pois essa versão está desatualizada e não é mais mantida pela comunidade. O Maven 2 utiliza protocolos obsoletos, como HTTP, para baixar *plugins* e recursos. No entanto, os servidores *web* atuais exigem conexões seguras via HTTPS, o que impossibilitou a compilação de código-fonte dessas versões.

[30] ⟨https://www.oracle.com/java/technologies/javase/j2se-1-5.html⟩

Outro desafio significativo está relacionado ao processo de análise baseado no *bytecode*, que apresenta limitações devido às otimizações realizadas pelo compilador Java. Essas otimizações podem comprometer a correspondência direta entre as linhas do código-fonte (.java) e o *bytecode* gerado (.class). Entre as alterações mais comuns estão a reordenação de instruções e a eliminação de trechos de código não utilizados (*dead code*). Como consequência, as associações entre as linhas do código-fonte e as instruções do *bytecode* podem se tornar imprecisas ou mesmo inexistentes.

Correspondências incorretas podem resultar na associação de métodos inexistentes às linhas modificadas e na perda de métodos que deveriam ser identificados. Para mitigar o primeiro problema, adotou-se uma abordagem em que um método só é registrado se seu nome for encontrado na linha correspondente do arquivo *.java* por meio de casamento de padrão. No entanto, não foi encontrada uma estratégia eficaz para evitar a perda de métodos, o que permanece como uma limitação da análise.

Além disso, o tempo de análise representa uma limitação considerável. O processo de clonar repositórios, compilar múltiplas versões de código e analisar o *bytecode* de cada arquivo Java exige um esforço computacional significativo, tornando a análise demorada, especialmente para projetos grandes ou com históricos extensos de contribuições.

Essas limitações ressaltam a importância de considerar tanto os desafios técnicos quanto as nuances contextuais ao interpretar os resultados deste estudo. A Tabela 5.5 resume as principais ameaças à validade interna, externa, de construção e de conclusão identificadas nesta pesquisa.

Tipo de Validade	**Ameaças Identificadas**
Validade Interna	- Correspondência imprecisa entre código-fonte e *bytecode* - Perda de dados devido à falha na compilação
Validade Externa	- Amostra limitada de repositórios - Fatores comportamentais não capturados
Validade de Construção	- Tempo de análise elevado - Limitações técnicas de ferramentas (JDK e Apache Maven)
Validade de Conclusão	- Limitação das métricas para avaliar conhecimento técnico

Tabela 5.11: Resumo das Ameaças à Validade

Trabalhos futuros podem abordar essas questões ampliando a amostra de repositórios e aprimorando as ferramentas de coleta e análise para mitigar os impactos dessas

restrições.

5.6 Considerações Finais

Este capítulo apresentou os resultados da avaliação experimental conduzida para validar a abordagem proposta, com foco na identificação de habilidades técnicas, colaboração e comprometimento dos desenvolvedores em repositórios de código-fonte. A análise dos dados coletados permitiu responder às questões de pesquisa propostas, fornecendo *insights* valiosos sobre as competências dos desenvolvedores nos projetos analisados.

Os resultados evidenciaram a predominância de três categorias de habilidades técnicas: Java *Base*, Estruturas de Dados e Teste de *Software*, refletindo a importância dessas competências para o desenvolvimento dos repositórios. A métrica MME demonstrou sua capacidade de identificar desenvolvedores com conhecimento atualizado, destacando a influência do tempo das contribuições no nível de proficiência técnica. Além disso, a análise da colaboração revelou que a maioria dos desenvolvedores (90,47%) interage com arquivos previamente modificados por outros. Por fim, a análise do comprometimento dos desenvolvedores revelou que 68% deles apresentaram uma única contribuição, enquanto os 32% restantes foram responsáveis por 99,55% das linhas de código adicionadas.

Apesar dos resultados promissores, algumas limitações devem ser consideradas. A análise foi restrita a três repositórios do Google, o que pode comprometer a generalização dos achados. Além disso, a dependência da compilação bem-sucedida das versões do projeto resultou na exclusão de parte dos dados, principalmente em versões mais antigas que utilizavam JDKs desatualizados. Outro desafio foi a imprecisão na associação entre o código-fonte e o *bytecode*, causada por otimizações do compilador Java. Essas limitações ressaltam a necessidade de aprimoramentos nas ferramentas e métodos de coleta e análise de dados.

6 Conclusão e Trabalhos Futuros

Esta monografia apresentou uma abordagem para a identificação de habilidades técnicas e comportamentais de forma automatizada em repositórios de código-fonte. Para avaliar a abordagem, foi implementada uma prova de conceito que extrai dados de projetos escritos em Java, versionados pelo Git e que utilizam o Maven. A avaliação foi realizada em três repositórios do time de desenvolvimento do Google — Guava, Gson e Guice. As habilidades técnicas foram coletadas a partir de invocações de métodos de classes pertencentes a bibliotecas externas. Duas habilidades comportamentais foram avaliadas: comprometimento e colaboração. O comprometimento foi medido pela quantidade de linhas de código adicionadas ao repositório pelo desenvolvedor. Já a colaboração foi representada pela quantidade de linhas adicionadas em arquivos alterados por outros desenvolvedores.

Os resultados evidenciaram a predominância de três categorias de habilidades técnicas: Java *Base*, Estruturas de Dados e Teste de *Software*, refletindo a importância dessas competências para o desenvolvimento dos repositórios analisados. Além disso, a aplicação da Média Móvel Exponencial (MME) permitiu identificar desenvolvedores com contribuições recentes e relevantes, proporcionando uma visão dos desenvolvedores essenciais para evolução dos projetos.

Em termos de habilidades comportamentais, verificou-se que 90,47% dos desenvolvedores apresentam algum nível de colaboração, refletindo a natureza do desenvolvimento de *software open-source*. Entretanto, 68% dos desenvolvedores contribuíram apenas uma vez, indicando um baixo nível de comprometimento.

Algumas limitações da abordagem foram identificadas e são sugeridos os trabalhos futuros apresentados a seguir.

A análise de um conjunto maior de projetos em que os desenvolvedores tenham colaborado. Embora as métricas definidas tenham apresentado resultados promissores, o conhecimento técnico dos desenvolvedores pode não ser totalmente refletido nas classes utilizadas nos projetos analisados, pois essas classes atendem a requisitos

técnicos específicos de cada repositório.

Análise qualitativa das habilidades comportamentais. As habilidades comportamentais, como comprometimento e colaboração, podem não ser completamente capturadas por meio do código-fonte. Isso ocorre porque esses aspectos também dependem de fatores externos, como processos internos de trabalho e dinâmicas de equipe.

Utilização de ferramentas de análise de código-fonte que não dependam do *bytecode*. A dependência do sucesso da compilação dos repositórios para a coleta dos dados e as dificuldades associadas à análise do *bytecode*, que pode sofrer alterações devido às otimizações realizadas pelo compilador Java, resultam na perda de parte dos dados. Dessa forma, sugere-se a utilização de ferramentas que analise diretamente o código-fonte (.java), como, por exemplo, o ANTLR[31].

Refinamento da métrica de avaliação de habilidades técnicas. Analisar a complexidade dos métodos e classes utilizados pelos desenvolvedores, para determinar as habilidades técnicas de forma criteriosa.

[31]https://www.antlr.org/

Bibliografia

AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. *Compiladores: Princípios, Técnicas e Ferramentas*. 2. ed. São Paulo: Pearson, 2008. E-book. Disponível em: ⟨https://plataforma.bvirtual.com.br⟩.

ALEX, K. *Soft Skills: Know Yourself & Know the World*. 1. ed. New Delhi: S Chand Publishing, 2009. 264 p. ISBN 9788121931922.

ALKHAZI, B.; DISTASI, A.; ALJEDAANI, W.; ALRUBAYE, H.; YE, X.; MKAOUER, M. W. Learning to rank developers for bug report assignment. *Applied Soft Computing*, v. 97, p. 106667, 2020.

ATZBERGER, D.; SCORDIALO, N.; CECH, T.; SCHEIBEL, W.; TRAPP, M.; DÖLLNER, J. Codecv: Mining expertise of github users from coding activities. In: *2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 2022. p. 143–147. Disponível em: ⟨https://doi.org/10.1109/SCAM55253.2022.00021⟩.

CHACON, S.; STRAUB, B. *Pro git*. [S.l.]: Springer Nature, 2014.

CHIAVENATO, I. *Planejamento, recrutamento e seleção de pessoal: como agregar talentos à empresa*. [S.l.]: Manole, 2009.

CHIAVENATO, P. *recrutamento e seleção de pessoal: como agregar talentos à empresa*. [S.l.]: São Paulo: Atlas, 1999.

FRANDSEN, T. F.; NIELSEN, M. F. B.; LINDHARDT, C. L.; ERIKSEN, M. B. Using the full pico model as a search tool for systematic reviews resulted in lower recall for some pico elements. *Journal of Clinical Epidemiology*, v. 127, p. 69–75, 2020. ISSN 0895-4356. Disponível em: ⟨https://www.sciencedirect.com/science/article/pii/S0895435620305692⟩.

GAGNÉ, M. *The Oxford handbook of work engagement, motivation, and self-determination theory*. [S.l.]: Oxford University Press, USA, 2014.

GREENE, G. J.; FISCHER, B. Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In: *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. [S.l.: s.n.], 2016. p. 804–809.

JARUCHOTRATTANASAKUL, T.; YANG, X.; MAKIHARA, E.; FUJIWARA, K.; IIDA, H. Open source resume (osr): A visualization tool for presenting oss biographies of developers. In: IEEE. *2016 7th International Workshop on Empirical Software Engineering in Practice (IWESEP)*. [S.l.], 2016. p. 57–62.

KINI, S. O.; TOSUN, A. Periodic developer metrics in software defect prediction. In: IEEE. *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. [S.l.], 2018. p. 72–81.

MAJID, S.; LIMING, Z.; TONG, S.; RAIHANA, S. Importance of soft skills for education and career success. *International Journal for Cross-Disciplinary Subjects in Education*, v. 2, n. 2, p. 1037–1042, 2012.

MARTINS, J. C. C. *Soft skills: conheça as ferramentas para você adquirir, consolidar e compartilhar conhecimentos*. 1ª. ed. [S.l.]: Brasport, 2017. ISBN 9788574528489.

MENEZES, G. G. L. On the nature of software merge conflicts. *Universidade Federal Fluminense-UFF*, 2016.

MONTANDON, J. E.; SILVA, L. L.; VALENTE, M. T. Identifying experts in software libraries and frameworks among github users. In: *Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, 2019. p. 276–286. Disponível em: ⟨https://doi.org/10.1109/MSR.2019.00054⟩.

OLIVEIRA, J. How to identify programming skills from source code? In: *Anais do 25. Congresso Ibero-Americano em Engenharia de Software (CIBSE)*. [S.l.]: Sociedade Brasileira de Computação, 2022. p. 384–391.

PATACSIL, F. F.; TABLATIN, C. L. S. Exploring the importance of soft and hard skills as perceived by it internship students and industry: A gap analysis. *Journal of Technology and Science education*, OmniaScience, v. 7, n. 3, p. 347–368, 2017.

PILATO, C. M.; COLLINS-SUSSMAN, B.; FITZPATRICK, B. W. *Version control with subversion: next generation open source version control*. [S.l.]: "O'Reilly Media, Inc.", 2008.

PRAKASH, M. Software build automation tools a comparative study between maven, gradle, bazel and ant. *Int. J. Softw. Eng. & Appl. DOI https//doi. org/10.5121/ijsea*, 2022.

QUINTELA, B. M.; SILVA, S. C. F. d.; ESTEVES, L. O.; DIAS, L. d. S.; DANIEL, L. C.; GUIMARÃES, M. L. R.; OLIVEIRA, A. Ferramentas e estratégias para aumentar a inclusão de gênero e raça na computação: um mapeamento sistemático. In: *Anais do 35º Simpósio Brasileiro de Informática na Educação (SBIE)*. Porto Alegre: Sociedade Brasileira de Computação, 2024. p. 2456–2468. Disponível em: ⟨https://doi.org/10.5753/sbie.2024.242550⟩.

REIS, A.; MEDEIROS, A.; COSTA, L.; WALTER, C. E.; AU-YONG-OLIVEIRA, M. Um estudo de metodologia mista sobre soft e hard skills. *ICIEMC Proceedings*, n. 3, p. 210–218, 2022.

ROBBINS, S.; JUDGE, T.; JUDGE, T. *Organizational Behavior*. Pearson, 2019. (Always learning). ISBN 9781292259239. Disponível em: ⟨https://books.google.com.br/books?id=_cwqtgEACAAJ⟩.

SGOBBI, T.; ZANQUIM, S. Soft skills: habilidades e competências profissionais requisitadas pelo mercado empreendedor. *Revista Científica Multidisciplinar Núcleo do Conhecimento. Ano*, v. 5, p. 70–92, 2020.

SONG, X.; YAN, J.; HUANG, Y.; SUN, H.; ZHANG, H. A collaboration-aware approach to profiling developer expertise with cross-community data. In: *2022 IEEE 22nd International Conference on Software Quality, Reliability and Security (QRS)*. Beijing, China: IEEE, 2022. p. 1–12.

Apêndices

Apêndice A - Lista Completa das Publicações Retornadas pela *String* de Busca

A lista completa das publicações identificadas pela *string* de busca utilizada na revisão da literatura, conforme critérios definidos no Capítulo 3, está apresentada a seguir. A Tabela 1 inclui informações como título, ano de publicação, referência completa e a biblioteca onde a publicação foi encontrada. Além disso, a tabela contém as colunas "1º Filtro", que se refere à leitura do título e do resumo, indicando o critério de exclusão aplicado ou "Ok" quando a publicação foi avaliada como pertinente. E "2º Filtro", que se trata da leitura completa da publicação, aplicando os mesmos critérios de exclusão.

Por exemplo, o artigo intitulado "A Collaboration-Aware Approach to Profiling Developer Expertise with Cross-Community Data", após a filtragem, está marcado com "Ok" nas colunas "1º Filtro" e "2º Filtro", indicando que foi aceito em ambas as etapas de avaliação. Por outro lado, o artigo "Assigning Bug Reports Using a Vocabulary-Based Expertise Model of Developers" foi excluído, conforme o Critério de Exclusão CE3 apresentado na Seção 3.1.

Tabela 1: Lista completa das publicações retornadas pela expressão de busca

Título	**Ano**	**Referência Completa**	**1º Filtro**	**2º Filtro**	**Biblioteca**
30th International Conference on Software Engineering, ICSE 2008 - 2008 International Working Conference on Mining Software Repositories, MSR'08	2008	30th International Conference on Software Engineering. Proceedings - International Conference on Software Engineering. Leipzig, Germany, 2008. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-57049114250&partnerID=40&md5=1d0b2f50a4dcf9a3bbe000ebce21ddb0⟩. Acesso em: 25 ago. 2024.	CE3	-	Scopus
A Collaboration-Aware Approach to Profiling Developer Expertise with Cross-Community Data	2022	SONG, Xiaotao; YAN, Jiafei; HUANG, Yuexin; SUN, Hailong; ZHANG, Hongyu. A collaboration-aware approach to profiling developer expertise with cross-community data. In: IEEE 22nd International Conference on Software Quality, Reliability, and Security (QRS), 2022. p. 1-12. DOI: 10.1109/QRS57517.2022.00043.	OK	OK	IEEE
A feature location approach supported by time-aware weighting of terms associated with developer expertise profiles	2016	ZAMANI, S.; LEE, S. P.; SHOKRIPOUR, R.; ANVIK, J. A feature location approach supported by time-aware weighting of terms associated with developer expertise profiles. Knowledge and Information Systems, [S.l.], v. 49, n. 2, p. 629-659, 2016. DOI: 10.1007/s10115-015-0909-5. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-84954327443&doi=10.1007%2fs10115-015-0909-5&partnerID=40&md5=b3bcb941a395d7e0af7d31a1b55c0e36⟩. Acesso em: 25 ago. 2024.	CE5	-	Scopus

Título	**Ano**	**Referência Completa**	**1º Filtro**	**2º Filtro**	**Biblioteca**
A Global View on the Hard Skills and Testing Tools in Software Testing	2019	FLOREA, Raluca; STRAY, Viktoria. A global view on the hard skills and testing tools in software testing. In: 2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE). São Paulo: IEEE, 2019. p. 143-151. DOI: 10.1109/ICGSE.2019.00035.	Ok	CE5	IEEE
A Theory of Scientific Programming Efficacy	2024	PERTSEVA, Elizaveta et al. A theory of scientific programming efficacy. In: Proceedings of the IEEE/ACM 46th International Conference on Software Engineering. 2024. p. 1-12.	CE5	-	IEEE
Analyzing the Learning Effectiveness of Generative AI for Software Development for Undergraduates in Sri Lanka	2024	SAMARAKOON, Pramodya et al. Analyzing the learning effectiveness of generative AI for software development for undergraduates in Sri Lanka. In: 2024 International Research Conference on Smart Computing and Systems Engineering (SCSE). IEEE, 2024. p. 1-7.	CE5	-	IEEE
Applying Two-rounds Collaborative Peer Review to Assess Students' Programming Assignments	2022	WANG, Ya-Chi; CHEN, Hsi-Min; HUANG, Chuan-Lin. Applying two-rounds collaborative peer review to assess students' programming assignments. In: 2022 IEEE International Conference on e-Business Engineering (ICEBE). IEEE, 2022. p. 246-251.	CE5	-	Scopus e IEEE
Assigning bug reports using a vocabulary-based expertise model of developers	2009	MATTER, Dominique; KUHN, Adrian; NIERSTRASZ, Oscar. Assigning bug reports using a vocabulary-based expertise model of developers. In: 2009 6th IEEE international working conference on mining software repositories. IEEE, 2009. p. 131-140.	CE3	-	IEEE
Bio-Docklets: Virtualization containers for single-step execution of NGS pipelines	2017	KIM, Baekdoo; ALI, Thahmina; LIJERON, Carlos; AFGAN, Enis; KRAMPIS, Konstantinos. Bio-Docklets: virtualization containers for single-step execution of NGS pipelines. Gigascience, v. 6, n. 8, 2017. gix048.	CE5	-	Scopus

Título	Ano	Referência Completa	1º Filtro	2º Filtro	Biblioteca
Can Source Code Analysis Indicate Programming Skills? A Survey with Developers	2022	OLIVEIRA, Johnatan; SOUZA, Maurício; FLAUZINO, Matheus; DURELLI, Rafael; FIGUEIREDO, Eduardo. Can source code analysis indicate programming skills? A survey with developers. In: International Conference on the Quality of Information and Communications Technology. Cham: Springer International Publishing, 2022. p. 156-171.	OK	CE4	Scopus
CIL-BSP: Bug Report Severity Prediction based on Class Imbalanced Learning	2022	U, Yu; HU, Xinping; CHEN, Xiang; QU, Yubin; MENG, Qianshuang. CIL-BSP: Bug report severity prediction based on class imbalanced learning. In: 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C). IEEE, 2022. p. 298-306.	CE5	-	Scopus
Code Review Quality: How Developers See It	2016	KONONENKO, Oleksii; BAYSAL, Olga; GODFREY, Michael W. Code review quality: how developers see it. In: Proceedings of the 38th International Conference on Software Engineering. 2016.	CE5	-	Scopus e IEEE
Code-Review-as-an-Educational-Service: A tool for Java code review in programming education	2025	BEATTIE, M.; WATSON, M.; GREER, D.; TOH, B. Y.; LI, Z. Code-Review-as-an-Educational-Service: A tool for Java code review in programming education. SoftwareX, v. 29, p. 102048, 2025.	CE5	-	Scopus
CodeCV: Mining Expertise of GitHub Users from Coding Activities	2022	ATZBERGER, Daniel; SCORDIALO, Nico; CECH, Tim; SCHEIBEL, Willy; TRAPP, Matthias; DÖLLNER, Jürgen. CodeCV: Mining Expertise of GitHub Users from Coding Activities. In: 2022 IEEE 22nd International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2022. p. 1-5. DOI: 10.1109/SCAM55253.2022.00021.	OK	OK	Scopus e IEEE

Título	**Ano**	**Referência Completa**	**1º Filtro**	**2º Filtro**	**Biblioteca**
CoLab: A collaborative laboratory for facilitating code reviews through a peer-to-peer network	2010	KINI, Shwetha. CoLab: a collaborative laboratory for facilitating code reviews through a peer-to-peer network. E-Learning and Digital Media, v. 7, n. 4, p. 416-429, 2010.	CE3	-	Scopus
Computing curricula for the 21st century	2008	KORNECKI, A. J. Computing curricula for the 21st century. IEEE Distributed Systems Online, v. 9, n. 2, 2008. DOI: 10.1109/MDSO.2008.5. Disponível em: ⟨https://www.scopus.com/inward/record.uri?eid=2-s2.0-40549105417&doi=10.1109%2fMDSO.2008.5&partnerID=40&md5=64aed774eb815b929a37b0ede3f38eda.⟩	CE3	-	Scopus
Conceptual framework for programming skills development based on microlearning and automated source code evaluation in virtual learning environment	2021	SKALKA, J.; DRLIK, M.; BENKO, L.; KAPUSTA, J.; RODRÍGUEZ DEL PINO, J. C.; SMYRNOVA-TRYBULSKA, E.; STOLINSKA, A.; SVEC, P.; TURCINEK, P. Conceptual framework for programming skills development based on microlearning and automated source code evaluation in virtual learning environment. Sustainability, v. 13, n. 6, p. 3293, 2021. Disponível em: https://doi.org/10.3390/su13063293. Acesso em: 27 jan. 2025.	OK	CE5	Scopus
CVExplorer: Identifying candidate developers by mining and exploring their open source contributions	2016	GREENE, G. J.; FISCHER, B. Cvexplorer: Identifying candidate developers by mining and exploring their open source contributions. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. [S.l.: s.n.], 2016. p. 804–809.	OK	OK	Scopus e IEEE

Título	Ano	Referência Completa	1º Filtro	2º Filtro	Biblioteca
DASP: A Framework for Driving the Adoption of Software Security Practices	2023	LARIOS-VARGAS, Enrique et al. Dasp: A framework for driving the adoption of software security practices. IEEE Transactions on Software Engineering, v. 49, n. 4, p. 2892-2919, 2023.	CE5	-	IEEE
Developing an Institutional Peer Code Review Tool using Software Engineering Principles	2022	BROWN, Tamaike; TENBERGEN, Bastian. Developing an institutional peer code review tool using software engineering principles. In: 2022 IEEE Frontiers in Education Conference (FIE). IEEE, 2022. p. 1-4.	CE5	-	Scopus e IEEE
Development of a Check Sheet for Code-review towards Improvement of Skill Level of Novice Programmers	2018	OEDA, Shinichi; KOSAKU, Hajime. Development of a check sheet for code-review towards improvement of skill level of novice programmers. Procedia Computer Science, v. 126, p. 841-849, 2018.	CE5	-	Scopus
EARec: Leveraging Expertise and Authority for Pull-Request Reviewer Recommendation in GitHub	2016	YING, Haochao; CHEN, Liang; LIANG, Tingting; WU, Jian. EAREC: leveraging expertise and authority for pull-request reviewer recommendation in GitHub. In: Proceedings of the 3rd International Workshop on Crowdsourcing in Software Engineering. 2016. p. 29-35.	OK	CE5	IEEE
Educational Code-Review Tool: A First Glimpse	2023	KUBINCOVÁ, Zuzana; KL'UKA, Ján; HOMOLA, Martin; MARUŠÁK, Adrián. Educational code-review tool: A first glimpse. In: International Conference on Methodologies and Intelligent Systems for Technology Enhanced Learning. Cham: Springer International Publishing, 2022. p. 113-122.	OK	CE4	Scopus

Título	Ano	Referência Completa	1º Filtro	2º Filtro	Biblioteca
Effective assignment and assistance to software developers and reviewers	2016	ZANJANI, Motahareh Bahrami. Effective assignment and assistance to software developers and reviewers. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2016. p. 1091-1093.	CE5	-	Scopus
General Principles of Programming: Computer and Statistical	2006	ISUKAPALLI, S. S.; ROY, A. General principles of programming (computer and statistical). In: Pharmacometrics: the Science of Quantitative Pharmacology. Wiley, 2007. capítulo 2.	CE1	-	Scopus
Hearing the voice of experts: Unveiling Stack Exchange communities' knowledge of test smells	2023	MARTINS, Luana et al. Hearing the voice of experts: unveiling Stack Exchange communities' knowledge of test smells. In: 2023 IEEE/ACM 16th International Conference on Cooperative and Human Aspects of Software Engineering (CHASE). IEEE, 2023. p. 80-91.	CE5	-	IEEE
History slicing	2011	SERVANT, Francisco; JONES, James A. History slicing. In: 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). IEEE, 2011. p. 452-455.	CE3	-	Scopus e IEEE
How long do junior developers take to remove technical debt items?	2020	LENARDUZZI, Valentina; MANDIĆ, Vladimir; KATIN, Andrej; TAIBI, Davide. How long do junior developers take to remove technical debt items? In: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). 2020. p. 1-6.	CE5	-	Scopus
How to Identify Programming Skills from Source Code?	2022	OLIVEIRA, Johnatan. How to Identify Programming Skills from Source Code?. In: CONGRESSO IBERO-AMERICANO EM ENGENHARIA DE SOFTWARE (CIBSE), 25. , 2022, Córdoba. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2022 . p. 384-391. DOI: https://doi.org/10.5753/cibse.2022.20988.	OK	OK	Scopus

Título	Ano	Referência Completa	1º Filtro	2º Filtro	Biblioteca
Identifying Experts in Software Libraries and Frameworks Among GitHub Users	2019	MONTANDON, João Eduardo; SILVA, Luciana Lourdes; VALENTE, Marco Tulio. Identifying Experts in Software Libraries and Frameworks among GitHub Users. In: Proceedings of the IEEE/ACM International Conference on Mining Software Repositories (MSR), 2019. p. 1-12. DOI: 10.1109/MSR.2019.00054.	OK	OK	IEEE
Implementing Effective Code Reviews: How to Build and Maintain Clean Code	2020	CARULLO, Giuliana. Implementing effective code reviews: how to build and maintain clean code. Apress, 2020.	CE1	-	Scopus
Inferring developer expertise through defect analysis	2012	NGUYEN, Tung Thanh; NGUYEN, Tien N.; DUESTERWALD, Evelyn; KLINGER, Tim; SANTHANAM, Peter. Inferring developer expertise through defect analysis. In: 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012. p. 1297-1300.	CE3	-	Scopus e IEEE
Learning programming with peer support, games, challenges and scratch	2015	BITTENCOURT, Roberto A.; SANTOS, David Moises B. dos; RODRIGUES, Carlos A.; BATISTA, Washington P.; CHALEGRE, Henderson S. Learning programming with peer support, games, challenges and scratch. In: 2015 IEEE Frontiers in Education Conference (FIE). IEEE, 2015. p. 1-9.	CE5	-	IEEE
Learning to rank developers for bug report assignment	2020	ALKHAZI, Bader; DISTASI, Andrew; ALJEDAANI, Wajdi; ALRUBAYE, Hussein; YE, Xin; MKAOUER, Mohamed Wiem. Learning to rank developers for bug report assignment. Applied Soft Computing, v. 106, p. 106667, 2020. DOI: 10.1016/j.asoc.2020.106667.	OK	OK	Scopus

Título	Ano	Referência Completa	1º Filtro	2º Filtro	Biblioteca
Mining expertise of developers from software repositories	2020	HAMMAD, Maen; HIJAZI, Haneen; HAMMAD, Mustafa; OTOOM, Ahmed Fawzi. Mining expertise of developers from software repositories. International Journal of Computer Applications in Technology, v. 62, n. 3, 2020. p. 227-239.	OK	CE4	Scopus
Mining software repositories to identify library experts	2018	SANTOS, Adriano; SOUZA, Maurício; OLIVEIRA, Johnatan; FIGUEIREDO, Eduardo. Mining software repositories to identify library experts. In: Proceedings of the VII Brazilian Symposium on Software Components, Architectures, and Reuse. 2018. p. 83-91.	OK	CE4	Scopus
Navigating Expertise in Configurable Software Systems through the Maze of Variability	2024	MILANO, Karolina; CAFEO, Bruno. Navigating expertise in configurable software systems through the maze of variability. arXiv preprint arXiv:2401.10699, 2024.	OK	-	Scopus e IEEE
On the randomness and seasonality of affective metrics for software development	2017	DESTEFANIS, Giuseppe; ORTU, Marco; COUNSELL, Steve; SWIFT, Stephen; TONELLI, Roberto; MARCHESI, Michele. On the randomness and seasonality of affective metrics for software development. In: Proceedings of the Symposium on Applied Computing. 2017. p. 1266-1271.	OK	CE4	Scopus
Open Borders? Immigration in Open Source Projects	2007	BIRD, C.; GOURLEY, A.; DEVANBU, P.; SWAMINATHAN, A.; HSU, G. Open borders? Immigration in open source projects. In: FOURTH INTERNATIONAL WORKSHOP ON MINING SOFTWARE REPOSITORIES (MSR'07: ICSE WORKSHOPS 2007). IEEE, 2007. p. 6-6.	CE3	-	IEEE

Título	Ano	Referência Completa	1º Filtro	2º Filtro	Biblioteca
Open Source Resume (OSR): A Visualization Tool for Presenting OSS Biographies of Developers	2016	JARUCHOTRATTANASAKUL, Thunyathon; YANG, Xin; MAKIHARA, Erina; FUJIWARA, Kenji; IIDA, Hajimu. Open Source Resume (OSR): A Visualization Tool for Presenting OSS Biographies of Developers. In: 2016 7th International Workshop on Empirical Software Engineering in Practice. [S.l.: s.n.], 2016. p. 57-62. DOI: 10.1109/IWESEP.2016.17.	OK	OK	Scopus e IEEE
Perceptions of Diversity on Git Hub: A User Survey	2015	VASILESCU, Bogdan; FILKOV, Vladimir; SEREBRENIK, Alexander. Perceptions of diversity on GitHub: a user survey. In: 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE, 2015. p. 50-56.	OK	CE5	IEEE
Periodic Developer Metrics in Software Defect Prediction	2018	KINI, Seldag Ozcan; TOSUN, Ayse. Periodic developer metrics in software defect prediction. In: 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2018. p. 72-81.	OK	OK	IEEE
phpSAFE: A Security Analysis Tool for OOP Web Application Plugins	2015	NUNES, Paulo Jorge Costa; FONSECA, José; VIEIRA, Marco. phpSAFE: A security analysis tool for OOP web application plugins. In: 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. IEEE, 2015. p. 299-306.	CE5	-	Scopus e IEEE
Recognizing eye tracking traits for source code review	2017	CHANDRIKA, K. R.; AMUDHA, Joseph; SUDARSAN, Sithu D. Recognizing eye tracking traits for source code review. In: 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA). IEEE, 2017. p. 1-8.	CE5	-	Scopus e IEEE

Título	**Ano**	**Referência Completa**	**1º Filtro**	**2º Filtro**	**Biblioteca**
Revisit of Automatic Debugging via Human Focus-Tracking Analysis	2016	XIE, Xiaoyuan; LIU, Zicong; SONG, Shuo; CHEN, Zhenyu; XUAN, Jifeng; XU, Baowen. Revisit of automatic debugging via human focus-tracking analysis. In: Proceedings of the 38th International Conference on Software Engineering. 2016. p. 808-819.	CE5	-	IEEE
Software engineering wastes-A perspective of modern code review	2020	ATIMA, Nargis; NAZIR, Sumaira; CHUPRAT, Suriayati. Software engineering wastes – a perspective of modern code review. In: Proceedings of the 3rd International Conference on Software Engineering and Information Management. 2020. p. 93-99.	CE5	-	Scopus
Software Evolution and Quality Data from Controlled, Multiple, Industrial Case Studies	2017	YAMASHITA, Aiko; ABTAHIZADEH, S. Amirhossein; KHOMH, Foutse; GUÉHÉNEUC, Yann-Gaël. Software evolution and quality data from controlled, multiple, industrial case studies. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017. p. 507-510.	CE5	-	IEEE
Teaching Code Review Management Using Branch Based Workflows	2016	KRUSCHE, Stephan; BERISHA, Mjellma; BRUEGGE, Bernd. Teaching code review management using branch based workflows. In: Proceedings of the 38th International Conference on Software Engineering Companion. 2016. p. 384-393.	CE5	-	Scopus e IEEE
The Open-Closed Principle of Modern Machine Learning Frameworks	2018	BRAIEK, Houssem Ben; KHOMH, Foutse; ADAMS, Bram. The open-closed principle of modern machine learning frameworks. In: Proceedings of the 15th International Conference on Mining Software Repositories. 2018. p. 353-363.	CE5	-	IEEE

Título	Ano	Referência Completa	1º Filtro	2º Filtro	Biblioteca
Understanding Code Change with Micro-Changes	2024	CHEN, L.; LANZA, M.; HAYASHI, S. Understanding code change with micro-changes. In: 2024 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME). IEEE, 2024. p. 363-374.	CE5	-	Scopus e IEEE
Using Open Source Tools to Prevent Write-Only Code	2009	LOVELAND, S. Using open source tools to prevent write-only code. In: 2009 SIXTH INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: NEW GENERATIONS. IEEE, 2009. p. 671-677.	CE3	-	Scopus e IEEE
Visualization of Knowledge Distribution across Development Teams using 2.5D Semantic Software Maps	2022	ATZBERGER, Daniel; CECH, Tim; JOBST, Adrian; SCHEIBEL, Willy; LIMBERGER, Daniel; TRAPP, Matthias; DÖLLNER, Jürgen. Visualization of Knowledge Distribution across Development Teams using 2.5 D Semantic Software Maps. In: VISIGRAPP (3: IVAPP). 2022. p. 210-217.	OK	CE4	Scopus
What Makes a Great Maintainer of Open Source Projects?	2021	DIAS, E.; MEIRELLES, P.; CASTOR, F.; STEINMACHER, I.; WIESE, I.; PINTO, G. What makes a great maintainer of open source projects?. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021. p. 982-994.	CE5	-	IEEE
Who Does What? Work Division and Allocation Strategies of Computer Science Student Teams	2021	VAN DER MEULEN, Anna; AIVALOGLOU, Efthimia. Who does what? Work division and allocation strategies of computer science student teams. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET). IEEE, 2021. p. 273-282.	CE5	-	Scopus e IEEE

Título	**Ano**	**Referência Completa**	**1º Filtro**	**2º Filtro**	**Biblioteca**
Who Will Leave the Company?: A Large-Scale Industry Study of Developer Turnover by Mining Monthly Work Report	2017	BAO, Lingfeng; XING, Zhenchang; XIA, Xin; LO, David; LI, Shanping. Who will leave the company?: a large-scale industry study of developer turnover by mining monthly work report. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017. p. 170-181.	CE5	-	Scopus e IEEE

Apêndice B - Dados Coletados das Publicações Selecionadas

A seguir, são apresentados os dados coletados das publicações selecionadas durante a revisão da literatura definida no Capítulo 3. Esses dados incluem o título da publicação, os autores envolvidos, a data de publicação e um resumo.

Dados da Publicação	
Título	A Collaboration-Aware Approach to Profiling Developer Expertise with Cross-Community Data
Autores	Xiaotao Song; Jiafei Yan; Yuexin Huang; Hailong Sun; Hongyu Zhang
Data da Publicação	2022
Resumo	
O artigo de Song et al. (2022) propõe uma abordagem para avaliar as habilidades dos desenvolvedores com base em suas contribuições nas plataformas Stack Overflow e GitHub. As *tags* associadas às perguntas e respostas no Stack Overflow, como Python, Java, entre outras, são utilizadas para identificar termos que fazem referência às habilidades dos desenvolvedores. No GitHub, são contabilizados os *commits* que o desenvolvedor criou relacionados a esses termos. A *expertise* total é obtida combinando os *scores* das duas plataformas. Além da quantidade de contribuições, sua importância também é considerada. No Stack Overflow, a relevância de uma pergunta ou resposta é definida pela quantidade de votos que ela recebeu. No GitHub, contribuições em repositórios com maior número de "*watchers*" e contribuições mais recentes tem maior relevância. Adicionalmente, foi construída uma rede de desenvolvedores na qual os nós representam os próprios desenvolvedores, e as arestas indicam suas interações (como trocas em perguntas e respostas, revisões de *commits* e projetos compartilhados no GitHub). O algoritmo *PageRank* foi aplicado com base no princípio de que um desenvolvedor é considerado importante se ele interage com outros desenvolvedores também considerados relevantes. Os autores afirmam que os experimentos realizados demonstraram a eficácia da abordagem proposta, superando as limitações de métodos anteriores que consideravam apenas uma única comunidade ou ignoravam as interações entre desenvolvedores. No entanto, destacam que os desenvolvedores analisados representam apenas uma pequena fração do total de desenvolvedores nas comunidades estudadas, o que pode limitar a generalização dos resultados.	

Dados da Publicação	
Título	CodeCV: Mining Expertise of GitHub Users from Coding Activities
Autores	Daniel Atzberger; Nico Scordialo; Tim Cech; Willy Scheibel; Matthias Trapp; Jürgen Döllner
Data da Publicação	2022

Resumo

Atzberger et al. (2022) apresentam o CodeCV, uma ferramenta que analisa os *commits* dos repositórios de usuários no GitHub para entender suas habilidades em linguagens de programação, bibliotecas de *software* e conceitos avançados, como Aprendizado de Máquina e Criptomoeda. O CodeCV utiliza um modelo chamado Labeled Latent Dirichlet Allocation (LLDA) para analisar essas atividades. O modelo é treinado com projetos do GitHub para identificar palavras que mostram o conhecimento dos desenvolvedores em conceitos específicos. O processo inclui coletar informações dos projetos, extrair palavras dos comentários e nomes de identificadores no código, e calcular a pontuação de *expertise* com base na frequência dessas palavras.

A abordagem descrita no artigo utiliza um conjunto de desenvolvedores para calcular a relevância dos termos em relação a um conceito específico. A relevância de um termo ww para um conceito cc é determinada com base na frequência de uso desse termo entre os desenvolvedores do grupo considerado. Isso significa que a análise considera como esses desenvolvedores usam os termos em seus históricos de *commits*, considerando que palavras menos frequentes podem ser mais relevantes para o conceito em questão.

Os resultados iniciais mostram que o CodeCV é bom para identificar o conhecimento dos desenvolvedores em conceitos avançados. No entanto, a ferramenta ainda não passou por muitos testes. Ela depende de um grupo de desenvolvedores para definir a importância dos termos, e mudanças nesse grupo podem alterar as pontuações. O artigo ressalta a necessidade de mais testes para confirmar se a ferramenta funciona bem em diferentes contextos.

Dados da Publicação	
Título	CVExplorer: Identifying Candidate Developers by Mining and Exploring Their Open Source Contributions
Autores	A Gillian J. Greene; Bernd Fischer
Data da Publicação	2016

Resumo

Greene e Fischer (2016) apresentam o CVExplorer, uma ferramenta desenvolvida para recomendar os melhores desenvolvedores em uma localização específica com base em habilidades técnicas desejadas. A ferramenta analisa os repositórios dos desenvolvedores para identificar suas habilidades, apresentadas em uma nuvem de *tags* interativa. As habilidades são derivadas dos tipos de arquivos que o desenvolvedor altera em cada *commit*. Por exemplo, se um desenvolvedor altera arquivos com a extensão ".java", isso indica sua habilidade em Java. Além disso, o CVExplorer extrai habilidades a partir dos arquivos *README* e analisa mensagens de *commit*. Para identificar as habilidades, o CVExplorer obtém uma lista dos 1000 primeiros desenvolvedores listados pela API do GitHub para uma determinada localização. Em seguida, a ferramenta analisa todos os repositórios desses desenvolvedores. Além da identificação das extensões dos arquivos, a ferramenta busca por termos catalogados pelos autores nos arquivos *README* e nas mensagens de *commit*.

A ferramenta foi avaliada em dois processos de seleção para vagas de emprego reais, recebendo *feedbacks* positivos. No entanto, a precisão na identificação das habilidades depende da corretude das mensagens de *commit*. Além disso, nem todos os projetos possuem arquivos *README* detalhados ou atualizados, o que pode limitar a extração de habilidades relevantes.

Dados da Publicação	
Título	How to Identify Programming Skills from Source Code?
Autores	Johnatan Oliveira
Data da Publicação	2022

Resumo

Oliveira (2022) propõe uma abordagem para identificar habilidades de desenvolvedores através da análise de repositórios no GitHub, com base em cinco dimensões: Amplitude de Conhecimento (diversidade de bibliotecas utilizadas), Intensidade de Conhecimento (profundidade do conhecimento em uma biblioteca), Proficiência em Código (produtividade baseada em linhas de código adicionadas), Colaboração (capacidade de trabalhar em equipe) e Experiência (uso de bibliotecas em diferentes projetos).

Para extrair essas habilidades, a abordagem utiliza quatro métricas principais. A primeira métrica é o número de *commits*, que avalia a frequência de uso de bibliotecas específicas. A segunda métrica é o número de *imports*, que mede a frequência de importações de uma biblioteca, refletindo o nível de interação do desenvolvedor com ela. A terceira métrica é o número de linhas de código (LOC) associadas a uma biblioteca, calculando a razão entre linhas de código modificadas e o total de *imports* no arquivo, multiplicada pelo número de *imports* relacionados à biblioteca. A quarta métrica contabiliza-se a quantidade de arquivos de código-fonte diferentes que o desenvolvedor modificou em parceria com outros desenvolvedores.

O estudo revelou uma precisão média de 88,49% na identificação das habilidades, com base em uma pesquisa realizada com 1137 desenvolvedores. Isso indica que a maioria dos candidatos identificados como especialistas possuía, de fato, um alto conhecimento sobre as bibliotecas avaliadas.

No entanto, é importante destacar que as métricas fornecem uma visão quantitativa da atividade dos desenvolvedores e não avaliam a qualidade do código ou a complexidade das tarefas. Portanto, um desenvolvedor com muitos *commits* pode não ser necessariamente mais habilidoso do que um com menos *commits*.

Dados da Publicação	
Título	Identifying Experts in Software Libraries and Frameworks Among GitHub Users
Autores	João Eduardo Montandon; Luciana Lourdes Silva; Marco Tulio Valente
Data da Publicação	2019

Resumo

O artigo de Montandon, Silva e Valente (2019) explora como identificar a experiência de desenvolvedores em três bibliotecas JavaScript: Facebook/React, MongoDB/Node-MongoDB e SocketIO/Socket.IO. A partir dos dados coletados da API do GitHub de repositórios que possuem dependências nessas bibliotecas, foram extraídas características indicativas de *expertise*. Essas características incluem o número de *commits* em projetos que utilizam as bibliotecas, a frequência de alterações, a quantidade de projetos nos quais o desenvolvedor contribuiu, e o volume de linhas de código adicionadas e removidas em arquivos que dependem das bibliotecas estudadas.

A experiência dos desenvolvedores foi avaliada utilizando algoritmos de aprendizado de máquina, especificamente *Random Forest* e *Support Vector Machine*. O modelo foi treinado com dados obtidos a partir de uma pesquisa realizada com 575 desenvolvedores, na qual eles autorrelataram sua experiência com as bibliotecas em questão, classificando-se em uma escala de 1 a 5. Essas autoavaliações foram utilizadas para rotular os dados, categorizando os desenvolvedores como especialistas, intermediários ou não especialistas.

O estudo destaca que os classificadores de aprendizado de máquina não apresentaram um bom desempenho, principalmente porque muitos especialistas têm pouca atividade no GitHub. Além disso, o modelo utilizado para treinamento pode conter erros de rotulagem, uma vez que é baseado nas autoavaliações dos desenvolvedores que participaram da pesquisa.

Dados da Publicação	
Título	Learning to rank developers for bug report assignment
Autores	Bader Alkhazi; Andrew DiStasi; Wajdi Aljedaani; Hussein Alrubaye; Xin Ye; Mohamed Wiem Mkaouer
Data da Publicação	2020

Resumo

A abordagem proposta por Alkhazi et al. (2020) é um modelo que automatiza a atribuição de relatórios de *bugs* aos desenvolvedores mais qualificados para corrigi-los. O modelo usa informações das mensagens de *commit* para entender a experiência dos desenvolvedores com certos tipos de *bugs*.

Ele aplica técnicas de aprendizado de máquina para dar uma pontuação a cada desenvolvedor com base no histórico de *bugs* que eles resolveram anteriormente. O modelo foi testado com cerca de 22.000 relatórios de *bugs* de quatro grandes projetos de código aberto em Java.

Uma limitação da abordagem proposta é que ela depende da qualidade das mensagens de *commit* e dos relatórios de *bugs*. Mensagens vagas ou mal escritas podem dificultar a avaliação precisa da experiência do desenvolvedor, afetando a eficácia do modelo. Além disso, pode não funcionar bem em projetos onde os desenvolvedores frequentemente mudam suas áreas de atuação.

Dados da Publicação	
Título	Open Source Resume (OSR): A Visualization Tool for Presenting OSS Biographies of Developers
Autores	Thunyathon Jaruchotrattanasakul; Xin Yang; Erina Makihara; Kenji Fujiwara; Hajimu Iida
Data da Publicação	2016

Resumo

Jaruchotrattanasakul et al. (2016) descrevem a ferramenta Open Source Resume (OSR), que visa apresentar as biografias de desenvolvedores que contribuíram para projetos de código aberto.
A OSR coleta informações como: dados pessoais dos desenvolvedores (nome, ID do GitHub e e-mail); quantidade de código adicionado em diferentes linguagens ao longo do tempo; uma lista de repositórios públicos com os quais o desenvolvedor esteve envolvido; uma linha do tempo das contribuições; e as atividades recentes.
Essas informações são obtidas por meio da API do GitHub e armazenadas em um banco de dados, facilitando consultas e visualizações. Contudo, a API do GitHub apresenta limitações, como um tempo de resposta de aproximadamente 10 segundos por requisição, e a OSR não tem acesso a repositórios privados, o que pode restringir a validação das biografias.

Dados da Publicação	
Título	Periodic Developer Metrics in Software Defect Prediction
Autores	Seldag Ozcan Kini; Ayse Tosun
Data da Publicação	2018

Resumo

Kini e Tosun (2018) investiga como a experiência dos desenvolvedores influencia a previsão de defeitos em projetos de *software*. Os autores usam métricas para medir a quantidade de código alterado ao longo do tempo, já que mudanças frequentes podem causar instabilidade e aumentar a chance de erros. O estudo também considera fatores como o número de *commits* e a propriedade do código, mostrando que desenvolvedores que fazem mais alterações em um módulo tendem a conhecê-lo melhor, o que pode melhorar a qualidade do código.
Além disso, o artigo analisa como os desenvolvedores colaboram, observando com que frequência diferentes pessoas trabalham juntas no mesmo módulo ou projeto. A colaboração entre os membros da equipe pode impactar diretamente a qualidade do *software*, pois equipes bem integradas e experientes costumam produzir código mais estável e com menos defeitos. Por outro lado, equipes com alta rotatividade ou pouca especialização podem enfrentar desafios maiores, o que pode resultar em um aumento no número de erros e na necessidade de retrabalho.
Para testar as métricas, os autores utilizam algoritmos de aprendizado de máquina, como *Random Forest*, J48 e *Naive Bayes*, e comparam seus resultados. A pesquisa foi realizada em dois projetos de código aberto em Java e ressalta que as métricas e métodos usados podem ser específicos para esse tipo de projeto, o que pode dificultar sua aplicação em outros contextos.