

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**BRENO FARIAS DA SILVA**

**ABORDAGEM PARA SELEÇÃO DE EXEMPLOS DE CÓDIGO DE SISTEMAS  
DISTRIBUÍDOS PARA A CRIAÇÃO DE EXEMPLOS TRABALHADOS DE  
ENGENHARIA DE SOFTWARE**

**CAMPO MOURÃO, PR, BRASIL**

**2024**

**BRENO FARIAZ DA SILVA**

**ABORDAGEM PARA SELEÇÃO DE EXEMPLOS DE CÓDIGO DE SISTEMAS  
DISTRIBUÍDOS PARA A CRIAÇÃO DE EXEMPLOS TRABALHADOS DE  
ENGENHARIA DE SOFTWARE**

**An approach for selecting code examples of distributed systems for  
creating worked examples of software engineering**

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Marco Aurélio Graciotto Silva  
Coorientador: Prof. Dr. Rodrigo Campiolo

**CAMPO MOURÃO, PR, BRASIL  
2024**



Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.  
[4.0 Internacional](#)

## **BRENO FARIAS DA SILVA**

# **ABORDAGEM PARA SELEÇÃO DE EXEMPLOS DE CÓDIGO DE SISTEMAS DISTRIBUÍDOS PARA A CRIAÇÃO DE EXEMPLOS TRABALHADOS DE ENGENHARIA DE SOFTWARE**

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Ciência da Computação do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 02/setembro/2024

---

Marco Aurélio Graciotto Silva  
Doutorado em Ciência de Computação e Matemática Computacional  
Universidade Tecnológica Federal do Paraná - Campo Mourão

---

Rodrigo Campiolo  
Doutorado em Ciência da Computação  
Universidade Tecnológica Federal do Paraná - Campo Mourão

---

Igor Scaliante Wiese  
Doutorado em Ciência da Computação  
Universidade Tecnológica Federal do Paraná - Campo Mourão

---

Lucio Geronimo Valentin  
Doutorado em Ciência da Computação  
Universidade Tecnológica Federal do Paraná - Campo Mourão

**CAMPO MOURÃO, PR, BRASIL  
2024**

## **AGRADECIMENTOS**

Certamente não haveria forma diferente de começar os meus agradecimentos sem expressar minha profunda gratidão ao meu orientador, Prof. Dr. Marco Aurélio Graciotto Silva, e ao coorientador, Prof. Dr. Rodrigo Campiolo, pelas imensuráveis orientações e feedback constante ao longo deste projeto de pesquisa. A dedicação e apoio de ambos foram fundamentais para o sucesso deste trabalho e por estarem presentes em minha trajetória durante a graduação.

Além disso, quero agradecer ao Prof. Dr. Luiz Arthur, que não esteve diretamente envolvido no projeto, mas que desempenhou um papel crucial como professor, mas também como amigo. Sua amizade e apoio constante tornaram minha jornada de aprendizado uma experiência enriquecedora. Suas conversas e mentorias moldaram minha perspectiva e contribuíram extraordinariamente para o meu crescimento como estudante e como pessoa.

Ao Prof. Dr. Igor Wiese, agradeço seu apoio e sua disposição para compartilhar conhecimento e incentivar minha participação em atividades extracurriculares, abrindo portas que foram fundamentais para o meu desenvolvimento como aluno nessa instituição.

Gostaria também de agradecer a minha namorada, Amanda Carvalho, visto que, todos os dias, ela esteve ao meu lado, oferecendo apoio inabalável, especialmente nos momentos em que duvidava das minhas capacidades, deixando-se sempre disponível para dividir as minhas dores e anseios com ela.

Gostaria de expressar minha profunda gratidão à minha mãe, Márcia Farias, pois, mesmo mantendo pouco contato, sempre esteve preocupada em saber se estou bem e feliz. Agradeço por ser uma fonte inesgotável de amor, por sempre me incentivar a alcançar os meus objetivos que, independente do quanto grande possam chegar a ser, sempre me apoiou e nunca duvidou de mim.

Gostaria de enviar um abraço especial aos meus amigos incríveis de Portugal, Bernardo Louro, Diogo Lopes e Simão Farias. Mesmo estando a mais de 7 mil km de distância, vocês fizeram parte de cada passo dessa jornada maluca. Vocês são mais do que amigos; são a prova de que as verdadeiras conexões resistem a qualquer distância. Obrigado por serem parte da minha vida de uma maneira tão única e significativa!

Por último, mas definitivamente não menos importante, dedico um agradecimento especial ao meu pai, Manoel Campos, minha maior inspiração de valor incomensurável. Seu apoio constante e palavras de incentivo não apenas foram um fator motivador crucial, mas também por sempre demonstrar orgulho e apontar que estou no caminho certo. Agradeço sinceramente por ter um pai cuja presença é insubstituível, tornando minha jornada ainda mais significativa.

A todos os que, de alguma forma, contribuíram para a realização desta pesquisa, meu sincero agradecimento.

## RESUMO

**Contexto:** Esta monografia investiga a evolução de projetos de Sistema Distribuído (SD)s de código aberto, incluindo *Apache Kafka* e *ZooKeeper*. **Objetivo:** O objetivo é selecionar trechos de código candidatos para a criação de exemplos trabalhados, que demonstrem conceitos importantes de SD e Engenharia de Software (ES) aplicáveis ao ensino de tais disciplinas. **Método:** Foi desenvolvida a ferramenta denominada *Worked-Example-Miner* (WEM), que implementa uma heurística baseada em métricas estáticas de qualidade de código, como acoplamento e complexidade, além da detecção de padrões de refatoração que indiquem melhorias no *design* do *software*. A heurística foi aplicada a vários repositórios de código aberto para identificar candidatos promissores. A integração com o modelo de Inteligência Artificial (IA) *Google Gemini* permitiu a identificação de candidatos que abordam tópicos específicos de SD, como consenso distribuído e tolerância a falhas. **Resultados:** A aplicação da heurística resultou na identificação de algumas dezenas de candidatos relevantes para a criação de exemplos trabalhados. A redução da quantidade de candidatos foi de até 99,760%, com uma média de 95,740% e um desvio padrão pequeno de 3,877% em relação ao número total de classes em cada repositório, demonstrando a eficácia da heurística. **Conclusões:** A heurística desenvolvida apresentou resultados promissores ao reduzir significativamente o número de candidatos para análise, mantendo a relevância educacional dos exemplos selecionados. A avaliação dos candidatos por especialistas em SD e ES confirmou a utilidade dos trechos de código para o ensino.

**Palavras-chave:** exemplo trabalhado; seleção de código; engenharia de software; sistemas distribuídos; métrica.

## ABSTRACT

This study investigates the evolution of open-source Distributed Systems (DS) projects, including *Apache Kafka* and *ZooKeeper*, which aims at selecting code snippets as candidates for creating worked examples that demonstrate important concepts of Software Engineering (SE) and DS, applicable to the teaching of these subjects. To achieve this, the tool called *Worked-Example-Miner* (WEM) was developed, implementing a heuristic that applies static code quality metrics, such as coupling and complexity, as well as detecting refactoring patterns that indicate improvements in software design. The application of the heuristic to various open-source repositories resulted in the identification of promising candidates for the creation of worked examples. The integration with the Artificial Intelligence (AI) model *Google Gemini* allowed the identification of candidates addressing specific DS topics, such as distributed consensus and fault tolerance. The results demonstrate that the developed heuristic successfully selects several dozen candidates within the analyzed projects. Considering that the viability of such a heuristic depends on selecting a reduced and relevant number of candidates for expert analysis, this figure represents a reduction in the number of candidates by up to 99.760%, with an average of 95.740% and a small standard deviation of 3.877% relative to the total number of classes in a repository, with thousands of files and tens of thousands of changes, demonstrating that the developed heuristic yields promising results. The evaluation of the candidates by experts in DS and SE confirmed the relevance of the selected code segments for educational purposes.

**Keywords:** worked examples; code selection; software engineering; distributed systems; metric.

## LISTA DE FIGURAS

<b>Figura 1 – Relação entre objetivo, questões e métricas, conforme a abordagem <i>Goal, Question, Metric</i> (GQM).</b>	<b>28</b>
<b>Figura 2 – Fluxograma das etapas do método.</b>	<b>39</b>
<b>Figura 3 – Diagrama da heurística.</b>	<b>40</b>
<b>Figura 4 – Arquivo <i>json</i> gerado pelo <i>repositories_picker.py</i></b>	<b>45</b>
<b>Figura 5 – Exemplo de repositórios selecionados pelo <i>repositories_picker.py</i></b>	<b>45</b>
<b>Figura 6 – Refatorações da classe <i>org.apache.zookeeper.server.quorum.Learner</i></b>	<b>48</b>
<b>Figura 7 – Lista de candidatos com quedas substanciais e padrões de refatorações identificados ZooKeeper (ZK)</b>	<b>48</b>
<b>Figura 8 – <i>Coupling Between Object</i> (CBO) da classe <i>org.apache.zookeeper.server.persistence.FileTxnLog</i>.</b>	<b>51</b>
<b>Figura 9 – <i>Response for a Class</i> (RFC) da classe <i>org.apache.zookeeper.server.persistence.FileTxnLog</i>.</b>	<b>52</b>
<b>Figura 10 – <i>Weight Method Class</i> (WMC) da classe <i>org.apache.zookeeper.server.persistence.FileTxnLog</i>.</b>	<b>53</b>
<b>Figura 11 – Regressão linear da métrica CBO da classe <i>org.apache.zookeeper.server.quorum.Follower</i> do ZK</b>	<b>54</b>
<b>Figura 12 – Regressão linear da métrica RFC da classe <i>org.apache.zookeeper.server.quorum.Follower</i> do ZK</b>	<b>55</b>
<b>Figura 13 – Regressão linear da métrica WMC da classe <i>org.apache.zookeeper.server.quorum.Follower</i> do ZK</b>	<b>56</b>
<b>Figura 14 – Evolução das métricas da classe <i>org.apache.zookeeper.server.quorum.Follower</i> do ZK</b>	<b>56</b>
<b>Figura 15 – Estatísticas das métricas do ZK.</b>	<b>56</b>
<b>Figura 16 – Regressão linear da métrica CBO da classe <i>org.apache.zookeeper.server.quorum.Learner</i> do ZK</b>	<b>57</b>
<b>Figura 17 – Regressão linear da métrica RFC da classe <i>org.apache.zookeeper.server.quorum.Learner</i> do ZK</b>	<b>58</b>
<b>Figura 18 – Regressão linear da métrica WMC da classe <i>org.apache.zookeeper.server.quorum.Learner</i> do ZK</b>	<b>59</b>

**Figura 19 – Regressão linear da métrica CBO da classe *org.corfudb.infrastructure.-orchestrator.Orchestrator* do *CorfuDB*** . . . . . 64

**Figura 20 – Regressão linear da métrica CBO da classe *org.corfudb.infrastructure.-ManagementAgent* do *CorfuDB*** . . . . . 65

## LISTA DE TABELAS

<b>Tabela 1 – Características dos Repositórios: Quantidade de Classes, Linhas de Código (<i>Lines of Code (LOC)</i>), Número de <i>Commits</i> e Tempo de Execução . . . . .</b>	<b>49</b>
<b>Tabela 2 – Quantidade de classes, candidatos e a redução de escopo dos repositórios após execução do terceiro algoritmo <i>PyDriller/metrics_changes.py</i>. . . . .</b>	<b>61</b>
<b>Tabela 3 – Similaridade média e intervalos de confiança para diferentes números de execuções. . . . .</b>	<b>62</b>
<b>Tabela 4 – Quantidade de classes, candidatos e a redução de escopo dos repositórios após execução do quarto algoritmo <i>gemini.py</i>. . . . .</b>	<b>62</b>
<b>Tabela 5 – Redução de escopo da quantidade de candidatos da 3<sup>a</sup> e 4<sup>a</sup> etapas. . . . .</b>	<b>62</b>
<b>Tabela 6 – Estatísticas descritivas da redução de escopo para os repositórios analisados. . . . .</b>	<b>63</b>
<b>Tabela 7 – Média das respostas dos especialistas de ES para os candidatos . . . . .</b>	<b>67</b>

## LISTA DE ABREVIATURAS E SIGLAS

### Siglas

ABE	Aprendizagem Baseada em Exemplos
ACID	Atomicidade, Consistência, Isolamento e Durabilidade
API	<i>Application Programming Interface</i>
AST	<i>Abstract Syntax Tree</i>
BSP	<i>Bulk Synchronous Parallel</i>
CAP	<i>Consistency, Availability and Partition Tolerance</i>
CBO	<i>Coupling Between Object</i>
CBOM	<i>Coupling Between Object Modified</i>
CCC	<i>Class Central Coupling</i>
CCL	<i>Class Communication Load</i>
CK	<i>Chidamber-Kemerer</i>
CVE	<i>Common Vulnerabilities and Exposures</i>
DIT	<i>Depth of Inheritance Tree</i>
ES	Engenharia de Software
GQM	<i>Goal, Question, Metric</i>
HiPC	<i>High-Performance Computing</i>
IA	Inteligência Artificial
IoT	<i>Internet of Things</i>
IP	<i>Internet Protocol</i>
IPC	<i>Inter-Process Comunication</i>
IPR	<i>Interprocess Reuse</i>
ISP	<i>Interface Segregation Principle</i>

JVM	<i>Java Virtual Machine</i>
LCOM	<i>Lack of Cohesion in Methods</i>
LOC	<i>Lines of Code</i>
MQTT	<i>Message Queuing Telemetry Transport</i>
NOC	<i>Number of Children</i>
OCP	<i>Open-Closed Principle</i>
PDC	<i>Parallel and Distributed Computing</i>
PLC	<i>Process-Level Cohesion</i>
PR	<i>Pull Request</i>
RCC	<i>Run-time Class Coupling</i>
RFC	<i>Response for a Class</i>
RMC	<i>Run-time Message Coupling</i>
RPC	<i>Remote Procedure Call</i>
SRP	<i>Single Responsibility Principle</i>
TCP	<i>Transmission Control Protocol</i>
TDD	<i>Test-Driven Development</i>
TF-IDF	<i>Term Frequency - Inverse Document Frequency</i>
UI	<i>User Interface</i>
WEM	<i>Worked-Example-Miner</i>
WMC	<i>Weight Method Class</i>
ZAB	<i>Zookeeper Atomic Broadcast Protocol</i>
ZK	ZooKeeper

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>12</b>
1.1	Estrutura do trabalho . . . . .	15
<b>2</b>	<b>REFERENCIAL TEÓRICO . . . . .</b>	<b>16</b>
2.1	Sistemas Distribuídos (SDs), pesquisa e práticas pedagógicas . . . . .	16
2.2	Exemplos trabalhados . . . . .	20
2.3	Exemplos trabalhados na Engenharia de <i>Software</i> . . . . .	22
2.4	Métricas para auxílio à seleção de códigos para exemplos trabalhos . .	24
2.5	Considerações finais . . . . .	26
<b>3</b>	<b>METODOLOGIA . . . . .</b>	<b>27</b>
3.1	Questões de pesquisa . . . . .	27
3.2	Abordagem proposta . . . . .	27
3.2.1	Métricas e critérios de qualidade e relevância utilizados . . . . .	28
3.2.2	Ferramentas . . . . .	32
3.2.3	Critérios de seleção de repositórios . . . . .	33
3.3	<b>Métodos . . . . .</b>	<b>34</b>
3.3.1	Processo de seleção . . . . .	35
3.3.2	Refinamento da Heurística . . . . .	36
3.3.3	Avaliação de candidatos . . . . .	37
<b>4</b>	<b>RESULTADOS . . . . .</b>	<b>41</b>
<b>4.1</b>	<b>Repositórios selecionados . . . . .</b>	<b>41</b>
4.1.1	<i>Apache Kafka</i> . . . . .	41
4.1.2	<i>ZooKeeper (ZK)</i> . . . . .	42
4.1.3	<i>CorfuDB</i> . . . . .	43
4.1.4	<i>ScaleCube</i> . . . . .	43
4.1.5	<i>Moleculer for Java</i> . . . . .	44
4.1.6	Conclusões da seleção de repositórios . . . . .	44
<b>4.2</b>	<b>Ferramenta <i>Worked-Example-Miner</i> (WEM) . . . . .</b>	<b>44</b>
<b>4.3</b>	<b>Análises Preliminares dos Projetos Selecionados . . . . .</b>	<b>49</b>
<b>4.4</b>	<b>Análises com a Heurística Proposta . . . . .</b>	<b>60</b>
<b>4.5</b>	<b>Validação de candidatos . . . . .</b>	<b>65</b>

<b>4.6</b>	<b>Limitações . . . . .</b>	<b>69</b>
<b>4.7</b>	<b>Ameaças à validade . . . . .</b>	<b>70</b>
4.7.1	Ameaças à Validade Interna . . . . .	70
4.7.2	Ameaças à Validade Externa . . . . .	70
4.7.3	Ameaças à Validade de Construção . . . . .	71
4.7.4	Ameaças à Validade de Conclusão . . . . .	71
4.7.5	Mitigação das Ameaças . . . . .	71
<b>4.8</b>	<b>Considerações finais . . . . .</b>	<b>72</b>
<b>5</b>	<b>CONCLUSÕES . . . . .</b>	<b>73</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>76</b>
	<b>APÊNDICES</b>	<b>80</b>
	<b>APÊNDICE A – QUESTIONÁRIO DE VALIDAÇÃO DO CANDIDATO 1</b>	
	<b>PARA EXEMPLO DIDÁTICO EM ENGENHARIA DE SOFTWARE . . . . .</b>	
		<b>81</b>
	<b>APÊNDICE B – QUESTIONÁRIO DE VALIDAÇÃO DO CANDIDATO 2</b>	
	<b>PARA EXEMPLO DIDÁTICO EM SISTEMAS DISTRIBUÍDOS . . . . .</b>	
		<b>85</b>

## 1 INTRODUÇÃO

No cenário atual da computação, os Sistemas Distribuídos (SD) desempenham um papel fundamental, alimentando a infraestrutura de serviços e aplicações que impulsionam nosso mundo digital. De acordo com Steen e Tanenbaum (2016, p. 968), um SD é definido como “um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente.” Diante disso, emergem desafios inerentes devido à necessidade de apresentar o sistema ao usuário como uma entidade homogênea, embora, intrinsecamente, o sistema seja constituído por diversas partes heterogêneas.

A complexidade inerente aos SDs os torna um campo de pesquisa desafiador, mas, ao mesmo tempo, altamente relevante na computação moderna. Compreender e lidar com essa heterogeneidade e complexidade é essencial para o desenvolvimento, manutenção e escalabilidade de SDs. Concomitantemente, requer-se o adequado desenvolvimento de competências em Engenharia de Software (ES) para garantir que esses sistemas sejam desenvolvidos com a qualidade necessária. De igual modo, a crescente dependência de empresas e instituições por esses sistemas torna o estudo de SDs indispensável para a formação de profissionais capacitados e para o avanço da qualidade do ensino em ES.

Nos SDs, as preocupações com a segurança dos dados e da comunicação, o desempenho eficiente e a capacidade de manter a operação contínua, mesmo diante de falhas, são instigações extremamente pertinentes. A necessidade de proteger informações sensíveis, assegurar rápido tempo de resposta, garantir que a troca de mensagens seja eficiente e manter a disponibilidade de serviços, torna o estudo da evolução do código em SDs ainda mais vital. Essas complexidades são estudadas em cursos universitários e estão intrinsecamente ligadas ao código que impulsiona esses sistemas, tornando a investigação sobre a evolução do código em SDs uma fonte valiosa sobre como esses sistemas evoluem para atender às demandas constantes.

Curriculos de Ciência da Computação, como o ACM/IEEE/AAAI CS2023, abordam computação distribuída e paralela como um tópico principal (RAJ; KUMAR, 2022, p. 3). Um estudo identificou que tópicos como Processos e *Threads*, Replicação, Chamadas de Sistema, Controle de Concorrência, Tolerância a Falhas, Sincronização, Comunicação, entre outros constituem os elementos-chave em cursos de SDs (ABAD; ORTIZ-HOLGUIN; BOZA, 2021).

Contudo, a ausência de pesquisas recentes sobre a elaboração de exemplos trabalhados utilizando SDs como objeto de estudo destaca a importância de investigar a evolução do código de tais sistemas para identificar melhorias e resolução de problemas que podem ser utilizados como exemplos em sala de aula. Não obstante esta exigência nos currículos, não se observam muitos esforços para desenvolver abordagens, técnicas e atividades de ensino que ajudem em sua realização de forma conjugada com outras competências, como as de ES. Isso ressalta a relevância dessa investigação, especialmente no que se refere ao aprimoramento da educação em ES. Essa importância é evidenciada pela relevância crítica desses sistemas, ex-

plicada pelo avanço de tecnologias de *software* e *hardware*, pelo aumento do acesso à internet e número de usuários.

A eficácia do ensino em Ciência da Computação, notadamente no âmbito da ES, seria consideravelmente aprimorada pelo desenvolvimento e aplicação de exemplos trabalhados (do inglês, *worked examples*). Um exemplo trabalhado é uma ferramenta pedagógica poderosa, definida como um trabalho cognitivo e experimental que oferece uma solução ideal e praticável para um problema específico, permitindo que aprendizes examinem e aprendam com a solução proposta (ATKINSON *et al.*, 2000). Diferentemente de um exemplo prático, que foca na implementação direta de uma solução, o exemplo trabalhado detalha o *passo a passo* da resolução, enfatizando o processo lógico e as decisões tomadas em cada etapa, o que facilita a compreensão profunda e a internalização dos conceitos abordados.

Entretanto, é relevante observar que exemplos trabalhados não são tão proeminente- mente empregados no contexto da ES. A literatura destaca a importância dos exemplos tra- balhados na disseminação de conceitos e padrões, fornecendo uma solução representativa do estado da arte para um tópico específico. A falta de pesquisa dedicada aos exemplos tra- balhados nesta área é evidente, e muitos professores enfrentam obstáculos ao incorporar exemplos reais em suas práticas pedagógicas (TONHÃO; COLANZI; STEINMACHER, 2023).

No âmbito da ES, os exemplos trabalhados frequentemente assumem a forma de apre- sentação passo-a-passo da execução de códigos específicos. No entanto, pesquisas indicam a escassez de estudos específicos sobre exemplos trabalhados na Ciência da Computação, uma vez que tarefas de programação exigem alto esforço cognitivo (SKUDDER; LUXTON-REILLY, 2014). Além disso, estratégias como “*Faded Worked Examples*”, que envolvem a apresentação gradual de exemplos resolvidos, têm mostrado impacto significativo na aprendizagem, permi- tindo ao aluno abstrair conceitos e desenvolver a capacidade de recuperar informações (SKUD- DER; LUXTON-REILLY, 2014).

O objetivo geral desta pesquisa é desenvolver uma heurística que viabilize a seleção de códigos representativos de melhorias de *software*, utilizando métricas específicas de qualidade de código em projetos de SDs. A pesquisa visa aprofundar a compreensão da evolução do código em resposta a mudanças nas métricas escolhidas, contribuindo para a construção de exemplos trabalhados que sejam aplicáveis ao contexto da ES.

Para alcançar esse objetivo, foram estabelecidos os seguintes objetivos específicos:

- Selecionar métricas e padrões de refatoração que reflitam melhorias na qualidade do *software*.
- Selecionar repositórios de código aberto relevantes para a área de SDs, garantindo que sejam projetos ativos, amplamente utilizados e que apresentem características técnicas complexas e aplicáveis ao mercado.
- Analisar a evolução do código-fonte nesses repositórios utilizando as métricas e pa- drões de refatoração selecionados.

- Realizar uma avaliação qualitativa dos candidatos em colaboração com especialistas, a fim de verificar sua relevância pedagógica e assegurar que estão aptos para a criação de exemplos trabalhados, contribuindo assim para o aprimoramento do ensino em SD e ES.

Os avanços da digitalização de serviços e produtos tornou a sociedade dependente de SDs, acessados por usuários a partir de múltiplos dispositivos como, computadores pessoais, smartphones e TVs. A dependência abrange desde redes sociais até sistemas de transporte, saúde, serviços governamentais e muito mais. Neste sentido, para conquistar uma boa formação de estudantes e profissionais na área da Computação, faz-se necessário o entendimento de ES e SDs, tornando-se uma necessidade iminente para atender à demanda crescente por profissionais qualificados nesse campo.

Dado o contexto anterior, a investigação sobre a evolução do código em SDs assume destaque. O estudo visa proporcionar uma compreensão mais profunda de como o código-fonte em SDs se adapta ao longo do tempo para atender a demandas como o aumento do número de usuários em escala não esperada, descobertas de vulnerabilidades e novas formas de ataque aos sistemas, além do uso de novos protocolos de comunicação para tentar resolver desafios enfrentados por estes sistemas.

Os objetivos traçados neste trabalho se alinham com a necessidade de abordar essas complexidades. O desenvolvimento de uma heurística para identificar trechos de código-fonte que sejam candidatos para a criação de exemplos trabalhados representativos da evolução em SDs contribuirá para uma melhor formação de estudantes e profissionais tanto na ES quanto em SDs, levando-se em consideração as inúmeras características de SDs já mencionadas anteriormente. Esses candidatos servirão como valiosos recursos de ensino e estudo, facilitando a compreensão das complexidades e desafios dos SDs em um ambiente distribuído e interconectado.

As principais contribuições deste trabalho são:

- I) Desenvolvimento de uma heurística para a seleção de trechos de códigos sobre SD candidatos para a criação de exemplos trabalhados, focada em identificar códigos que demonstrem melhorias significativas na qualidade do *design de software* do ponto de vista da ES;
- II) Implementação da ferramenta *Worked-Example-Miner* (WEM) (SILVA, 2023c), projetada especificamente para automatizar a coleta de dados e metadados relevantes para a seleção de exemplos de código-fonte. Esta ferramenta, amplamente documentada, utiliza a heurística desenvolvida para filtrar e identificar os candidatos mais promissores, visando a criação de exemplos trabalhados para ES.
- III) Conjunto de trechos de código candidatos para serem transformados em exemplos trabalhados em ES. Esta seleção é o resultado direto da aplicação da heurística de-

senvolvida, demonstrando sua aplicabilidade e eficácia na identificação de candidatos que podem potencializar o ensino e aprendizado de práticas de desenvolvimento de *software*;

- IV) Inclusão da métrica *MethodInvocationCounter* no projeto CK para Java (SILVA, 2024b), que implementa o conjunto de métricas CK (CHIDAMBER; KEMERER, 1994). Esta métrica permite uma análise detalhada dos métodos invocados por cada método em uma classe, registrando, de forma precisa, a frequência com que cada método é chamado. Essa contribuição proporciona uma visão mais abrangente e sintetizada das interações entre os métodos, enriquecendo a análise de código com informações valiosas sobre o comportamento e a complexidade das classes.

Portanto, a justificativa para este estudo se baseia na aplicação das práticas e técnicas de ES à evolução do código em SDs, contribuindo para a formação de profissionais mais capacitados e avançando a qualidade do ensino em ES. Embora seja possível abordar o ensino de SDs sem o uso da heurística desenvolvida neste estudo, isso demandaria um esforço considerável por parte do professor para identificar, selecionar e adaptar exemplos didáticos relevantes. Este estudo busca preencher uma lacuna na literatura sobre a criação de exemplos trabalhados em SDs, facilitando o trabalho docente e fornecendo percepções valiosas para o desenvolvimento e manutenção de SDs, além de aprimorar a prática profissional neste campo da computação. Além disso, os exemplos seriam atuais e relacionados a *software* em uso e não exemplos didáticos desconectados do mercado de trabalho.

### **1.1 Estrutura do trabalho**

Esta monografia está dividida em cinco capítulos. O Capítulo 2 abrange a fundamentação teórica e revisão de trabalhos relacionados, proporcionando a base conceitual para a compreensão do trabalho. O Capítulo 3 detalha a abordagem proposta, incluindo questões da pesquisa, ferramentas e métodos considerados na nesta pesquisa. O Capítulo 4 apresenta os resultados, a partir da análise das métricas obtidas dos projetos selecionados. Finalmente, o Capítulo 5 oferece uma síntese dos principais resultados, mencionando as limitações do estudo e direções para trabalhos futuros.

## 2 REFERENCIAL TEÓRICO

Este capítulo oferece uma base conceitual para compreender os principais elementos de SDs e seus desafios. São explorados conceitos acerca de exemplos trabalhados, além de expor alguns estudos sobre sua utilização. Tais estudos englobam não apenas a educação em geral, mas também a área de ES. Por fim, é fornecido um contexto abrangente que sustentará a análise e discussão dos resultados obtidos ao longo deste trabalho.

### 2.1 Sistemas Distribuídos (SDs), pesquisa e práticas pedagógicas

Um SD é aquele no qual os componentes localizados em computadores interligados em rede se comunicam e coordenam suas ações apenas passando mensagens. A motivação para o desenvolvimento de SDs é impulsionada pela necessidade de melhorar a eficiência na utilização de recursos computacionais, possibilitando que a carga de trabalho seja dividida entre vários servidores, oferecendo suporte a aplicações multitarefa e fornecendo maior confiabilidade por meio da redundância de dados e serviços. A facilidade de acesso remoto a recursos também é uma motivação significativa, permitindo que usuários acessem informações e serviços independentemente da localização física dos recursos (STEEN; TANENBAUM, 2016).

É notável a relevância de SDs para a sociedade, a elevada produção científica neste tema e o interesse no ensino de tal disciplina em cursos de graduação e pós-graduação. Neste sentido, eventos como o EduPar (*Education on Parallel and Distributed Computing*) e EduHiPC (*Education for High-Performance Computing*) reúnem educadores, pesquisadores e profissionais da indústria para discutir e compartilhar ideias sobre o ensino de *Parallel and Distributed Computing* (PDC) e *High-Performance Computing* (HiPC).

O EduPar é um evento que tem se dedicado, desde sua primeira edição em 2011, a explorar tópicos como o *design* de currículos, metodologias pedagógicas, ferramentas e recursos educacionais, e a integração de PDC em cursos de ciência da computação e engenharia (NSF/TCPP, 2023b).

Ao longo das edições, o EduPar tem testemunhado um aumento no interesse por tópicos emergentes como HiPC, aprendizado de máquina e computação em nuvem, além de um foco crescente em questões de diversidade e inclusão na educação em computação. Novas áreas de interesse também surgiram, como a integração de PDC e a exploração de ambientes de aprendizado informal. O EduPar continua a ser um fórum importante para a comunidade educacional em PDC, promovendo a troca de conhecimento e a colaboração para melhorar a qualidade do ensino e preparar os alunos para os desafios do futuro (NSF/TCPP, 2023b).

O EduHiPC é um *workshop* anual dedicado à educação em HiPC e PDC. Desde sua primeira edição em 2013, o evento tem se expandido, abrangendo um público cada vez mais amplo de acadêmicos, profissionais da indústria e governos, principalmente da Índia e da Ásia. Os tópicos de interesse do EduHiPC refletem as tendências emergentes no campo da HiPC e

PDC, com foco em questões pedagógicas, métodos inovadores de ensino, recursos educacionais e desafios específicos enfrentados por países em desenvolvimento (NSF/TCPP, 2023a).

Ao longo das edições, houve um aumento constante no interesse por tópicos relacionados a SD, incluindo a integração de conceitos de PDC em cursos básicos de ciência da computação e engenharia da computação, o desenvolvimento de ferramentas pedagógicas e ambientes de programação para SD, e a exploração de modelos de programação e computação distribuída adequados para ensino. O EduHiPC tem sido um fórum importante para a troca de ideias e experiências sobre a educação em HiPC e PDC, e tem desempenhado um papel de promoção da inclusão de tópicos relacionados a SD em currículos de ciência da computação e engenharia da computação em todo o mundo (NSF/TCPP, 2023a).

O currículo do NSF/IEEE-TCPP para PDC, versão 2.0, aborda diversos tópicos relevantes para a disciplina de SD, explorando os desafios e oportunidades inerentes à computação em ambientes distribuídos (NSF/IEEE-TCPP, 2024). Dentre eles, destacam-se:

- i) arquiteturas de memória compartilhada e distribuída, incluindo conceitos como NUMA, *cache coherence* e protocolos de consistência;
- ii) mecanismos de comunicação, como *message passing*, *handshaking* e protocolos de roteamento;
- iii) modelos de programação distribuída, como *Bulk Synchronous Parallel* (BSP) e *MapReduce*;
- iv) algoritmos distribuídos, incluindo técnicas de sincronização, exclusão mútua, detecção de término e algoritmos para problemas como ordenação e busca;
- v) SD emergentes, como Internet das Coisas (*Internet of Things* (IoT)), computação de borda, sistemas ciberfísicos e blockchain;
- vi) e questões de segurança em SD, incluindo ataques distribuídos e a tensão entre privacidade e segurança.

O currículo do CS2023 para PDC aborda tópicos relevantes para a disciplina de SD, principalmente nas áreas de Comunicação, Coordenação e Avaliação em PDCs (ACM, 2023). Sobre comunicação em PDC, o currículo discute diferentes tipos de comunicação em SD, incluindo canais, memória compartilhada e *datastores*, além de suas propriedades e extensões, como topologias, consistência, desempenho e segurança. A coordenação em PDC trata de mecanismos de coordenação entre processos em SD, como dependências, construções de controle, atomicidade e suas propriedades e extensões, incluindo progressão, atomicidade e interação com outras formas de controle de programa. Por fim, a avaliação em PDC apresenta métodos e ferramentas para avaliar SD, incluindo especificação formal, análise estática (a qual é utilizada neste trabalho), avaliação empírica e técnicas específicas de domínio de aplicação.

Esses tópicos também relacionam-se diretamente com o estudo de SD, explorando os desafios e as soluções para a comunicação, coordenação e avaliação de sistemas complexos e distribuídos (ACM, 2023).

Esses eventos e iniciativas para desenvolvimento de currículos de referência destacam a crescente relevância da educação em SD. A realização contínua e o crescimento desses *workshops* ilustram uma tendência de se concentrar em metodologias educacionais e na integração de tópicos de SD para preparar os alunos para os desafios futuros no campo da computação. Embora existam estudos sobre a integração de conteúdos e habilidades sobre SD em disciplinas introdutórias de algoritmos e estruturas de dados (CS1 e CS2) (MCQUAIGUE *et al.*, 2023; GHAFOOR *et al.*, 2023), não se observa esta integração com conteúdos de ES.

Um estudo abrangente foi conduzido em 2019 sobre a grade curricular de cursos de Computação (ABAD; ORTIZ-HOLGUIN; BOZA, 2021). As grades das 51 disciplinas mais famosas sobre SDs em programas de Ciência da Computação foram analisadas, conforme as melhores universidades pelo *Times Higher Education*, visando identificar os principais tópicos abordados. A análise desses dados revelou que conceitos fundamentais, como Processos e *Threads*, Replicação, Chamadas de Sistema, Controle de Concorrência, Tolerância a Falhas, Sincronização, Comunicação, entre outros, constituem os elementos-chave frequentemente lecionados em disciplinas de SDs. Além disso, a pesquisa abordou a questão das referências utilizadas na elaboração do conteúdo dos cursos, identificando que obras como “*Distributed Systems*” na edição de 2017, de Van Steen e Tanenbaum, e “*Distributed Systems: Concepts and Design*” na edição de 2011, de Coulouris, são fontes comuns amplamente utilizadas para explicar os princípios dos SDs (ABAD; ORTIZ-HOLGUIN; BOZA, 2021).

Assim, observa-se que a educação em SD faz parte do núcleo de cursos de graduação e pós-graduação em muitas universidades ao redor do mundo. Estes cursos são projetados para fornecer uma base sólida em teoria, bem como prática, preparando os alunos para enfrentar os desafios contemporâneos de desenvolvimento e manutenção de *software* em ambientes distribuídos (ABAD; ORTIZ-HOLGUIN; BOZA, 2021). A estrutura desses cursos frequentemente incorpora:

- Aulas teóricas que cobrem os fundamentos e os conceitos avançados.
- Laboratórios e projetos práticos que permitem aos alunos aplicarem o conhecimento teórico em cenários reais, usando ferramentas e tecnologias atuais.
- Estudos de caso e análises de SDs existentes para entender melhor as soluções práticas e os desafios do mundo real.

Além de aulas e projetos práticos, os estudantes podem encontrar uma variedade de recursos externos para aprofundar seu conhecimento. Por exemplo, o repositório *Awesome Distributed Systems* (ANALYST, 2015) fornece uma lista curada de materiais sobre SD. Este repositório foi inspirado por outras listas “awesome” e se concentra principalmente em leituras sobre

arquitetura de SDs, ao invés de código puro. Inclui uma seção de “*Bootcamp*” com recursos essenciais para iniciantes, uma ampla seleção de livros, desde introduções básicas até obras acadêmicas avançadas, e uma coleção de artigos importantes que todo profissional da área deve conhecer. Além disso, cobre tópicos avançados como consenso distribuído, tolerância a falhas, sistemas de armazenamento e bases de dados distribuídas, oferecendo aos estudantes e profissionais uma rica fonte de informação e inspiração para ambos estudos e aplicações práticas.

A inclusão de SD nos currículos de Ciência da Computação também apresenta desafios, especialmente no que diz respeito à necessidade de manter o conteúdo atualizado com as rápidas mudanças tecnológicas, além de garantir que os alunos adquiram experiências práticas relevantes, dada a complexidade dos SD. Ao dominar os conceitos e as tecnologias associadas a SD, os alunos estão bem preparados para carreiras em desenvolvimento de *software* e muitas outras áreas críticas onde SDs desempenham um papel central.

A implementação de SDs enfrenta diversos desafios, sendo um deles a heterogeneidade, que se manifesta em diferentes níveis, como em *hardware*, rede, linguagem de programação e implementações realizadas por diferentes desenvolvedores. Essa diversidade pode exigir o uso de *middlewares* para fornecer uma camada de abstração e facilitar a integração. A ausência de um relógio global em SDs introduz desafios na coordenação de eventos e sincronização entre diferentes partes do sistema. A tolerância a falhas é abordada por meio de técnicas como *checksums* para detecção de erros, mascaramento de falhas usando retransmissão de dados, além de contar com o uso de operações de *rollback* para manter a consistência, mesmo diante de falhas (COULOURIS *et al.*, 2011).

Para atingir escalabilidade, SDs empregam técnicas como replicação de dados, uso de cache distribuída, distribuição de carga entre múltiplos servidores e utilização de *web proxies*, entre outros. No entanto, a escalabilidade também traz consigo desafios, como a necessidade de manter a consistência entre as réplicas de dados. Adicionalmente, o Teorema *Consistency, Availability and Partition Tolerance* (CAP) estabelece que é impossível garantir fortemente, de forma simultânea, características de Consistência, Disponibilidade e Tolerância a partição em um SD. Assim, durante o desenvolvimento de um SD, é preciso fazer um balanço entre diferentes objetivos (BREWER, 2012).

A troca de mensagens *Inter-Process Comunication* (IPC) é uma questão pertinente em SDs, podendo ocorrer de forma síncrona ou assíncrona, além de ser classificada como bloqueante ou não bloqueante, dependendo do comportamento desejado. Do mesmo modo, a coordenação e acordo na seleção de um nó líder, por exemplo, também desempenham papéis cruciais na operação confiável de SDs. Várias técnicas podem ser usadas para isso, incluindo eleição, protocolos de consenso e algoritmos de ordenação de eventos. Em específico, a ordenação de eventos é tarefa particularmente complexa pela ausência de um relógio global, a qual na maioria das vezes é baseada em relações *happened-before* (COULOURIS *et al.*, 2011).

Em resumo, o desenvolvimento de SDs é desafiador, o que evidencia a importância do estudo desses sistemas. A complexidade desses desafios destaca a importância da ES na busca por soluções melhores. Ao analisar a evolução do código em SDs por meio de estudos de casos, a ES pode extrair *insights* valiosos para a aplicação prática de conceitos teóricos e estratégias de resolução de problemas. A utilização de SDs como cenário de estudo oferece uma base sólida para construir exemplos trabalhados, enriquecendo o ensino em ES. A interseção entre os desafios de SDs e os princípios da ES cria uma oportunidade única para desenvolver materiais de ensino que promovam uma compreensão aprofundada, preparando os estudantes para enfrentar as complexidades do desenvolvimento de *software*.

## 2.2 Exemplos trabalhados

Conforme Atkinson *et al.* (2000), um exemplo trabalhado é definido como um trabalho cognitivo e experimental com o intuito de fornecer uma solução ideal, todavia próxima ao praticável, para um problema específico, no qual uma pessoa com escasso ou nenhum conhecimento acerca do tema possa examinar e aprender com a solução proposta. Sendo assim, o desenvolvimento e estudo de exemplos trabalhados enriquecem a qualidade do que é lecionado em sala de aula, uma vez que a solução ideal pode representar muito bem o estado da arte para um determinado tópico, visto que dissemina conceitos e padrões do problema apresentado (ATKINSON *et al.*, 2000).

No mesmo contexto, os principais elementos na elaboração de um exemplo trabalhado de alta qualidade estão intrinsecamente vinculados aos conceitos de “*inter-example feature*” e “*intra-example feature*”. A concepção do termo “*inter-example feature*” destaca a importância de fornecer uma diversidade de exemplos que ilustrem estratégias múltiplas para problemas similares, porém de tipos diferentes, enquanto o termo “*intra-example feature*” trata de características intrínsecas à estrutura interna de cada exemplo (ATKINSON *et al.*, 2000).

O artigo intitulado “*Worked Examples in Computer Science*” (SKUDDER; LUXTON-REILLY, 2014) destaca a escassez de pesquisas sobre exemplos trabalhados na área da Ciência da Computação. Além disso, o trabalho expõe que as tarefas de programação impõem uma alta carga cognitiva, logo isso pode ser um indício para haver poucas publicações estudando exemplos trabalhados nesta área. Porém, o uso de exemplos trabalhados seria benéfico, visto que diminui a carga cognitiva para compreensão do conteúdo (ATKINSON *et al.*, 2000). Adicionalmente, o autor investiga as ramificações de exemplos trabalhados na área da Computação, frequentemente manifestadas na apresentação passo-a-passo da execução de um código específico (do inglês “*code-tracing*”), prática recorrente em disciplinas como análise de algoritmos. Não suficiente, o estudo também explora o emprego do conceito de “*problem solution pair*”, no qual um problema é apresentado, estimulando o aluno a tentar resolvê-lo, seguido pela exposição da solução correspondente. Por fim, de maneira menos convencional, são uti-

lizadas técnicas de geração de código enquanto se leciona um conteúdo, o que contribui para esclarecer o raciocínio empregado na elaboração da solução para o problema.

Muldner, Jennings e Chiarelli (2022) examinaram os exemplos trabalhados nas disciplinas de Ciência da Computação, enfocando especialmente em exemplos de rastreamento e geração de código, conforme discutido previamente. O estudo avança na análise de diferentes estratégias de emprego e criação de exemplos trabalhados. Notavelmente, a prática de rastreamento de código, empregando exemplos de modelagem, demonstrou que alunos expostos a tais exemplos obtiveram melhores resultados acadêmicos e exibiram taxas de desistência inferiores em comparação com seus pares que não tiveram acesso a esses recursos.

Em contraste, o estudo anterior Muldner, Jennings e Chiarelli (2022) expôs que o uso de vídeos instrutivos sugeriu que a presença física do instrutor em sala de aula não influencia de maneira significativa o aprendizado. Ademais, verificou-se que a apresentação estática de exemplos de código favorece o processo educativo de estudantes com menor base de conhecimento prévio sobre o assunto. Concluindo, a implementação de ferramentas de visualização, em detrimento das atividades tradicionais de rastreamento de código, mostrou-se capaz de elevar substancialmente tanto o desempenho quanto o aprendizado dos alunos.

No mesmo contexto, no que se refere aos exemplos vinculados à geração de código, o estudo concluiu que não há evidências suficientes para sustentar que o uso de exemplos de modelagem na geração de código resulte em uma melhoria na aprendizagem quando comparado ao uso de exemplos estáticos. Entretanto, os estudantes expressaram considerar os exemplos de geração de código em tempo real como úteis, enquanto os exemplos estáticos possuem a vantagem de permitir que os alunos comprehendam os elementos do modelo em seu próprio ritmo de assimilação. Uma abordagem sugerida para aprimorar o uso de exemplos de geração de código é a inclusão de submetas nos exemplos, possibilitando uma melhor fragmentação das etapas necessárias para construir a solução do problema. Desta forma, o emprego de submetas indicou uma melhoria significativa no desempenho dos estudantes universitários.

Não suficiente, o artigo Muldner, Jennings e Chiarelli (2022) também explorou o uso de exemplos incompletos, que apresentam lacunas na solução para que os estudantes as preencham: uma abordagem promissora para apoiar alunos em atividades de programação. Exemplos tradicionais com lacunas, onde os alunos completam programas existentes, mostraram-se benéficos. Em comparação com atividades de geração de código, os estudantes que completaram exemplos relataram menor carga cognitiva. Outra modalidade de exemplos incompletos são os quebra-cabeças de *Parsons* (do inglês “*Parsons puzzles*”), nos quais os alunos organizam fragmentos de código desordenados. Embora esses quebra-cabeças tenham reduzido o tempo de conclusão, não há evidências sólidas de que melhorem o aprendizado. O estudo também conclui que a presença de distratores e o tipo de *feedback* afetam os resultados.

A exposição dos alunos a um determinado conteúdo, como já visto, pode ser feita pelo uso de exemplos trabalhados (ATKINSON *et al.*, 2000). Todavia, o uso de “*Faded worked examples*” é uma estratégia de ensino que apresenta exemplos de problemas resolvidos aos alunos

de forma gradual. Inicialmente, fornece-se um exemplo completamente resolvido para garantir a compreensão total da solução. Em seguida, ao longo de problemas subsequentes, partes do exemplo resolvido são progressivamente removidas. Os alunos são desafiados a preencher as lacunas, completando as partes ausentes da solução. Isto implica em abster o discente de lidar com situações passivas, tornando-o proficiente em resgatar informações no hipocampo do cérebro (SKUDDER; LUXTON-REILLY, 2014).

O impacto dos exemplos trabalhados na aprendizagem varia conforme o nível de experiência do aprendiz. Para iniciantes, esses exemplos proveem benefícios ao frisar a atenção nos elementos cruciais do problema, facilitando a criação de esquemas de resolução pertinentes e otimizando recursos cognitivos em comparação com a abordagem direta na resolução de problemas. Em contraste, para indivíduos com certa experiência, os exemplos trabalhados podem tornar-se supérfluos, resultando em uma carga cognitiva extrínseca (SKUDDER; LUXTON-REILLY, 2014).

Assim, de modo geral é possível observar que os estudos existentes evidenciam que o uso de exemplos trabalhados, quando aplicados corretamente, potencializa o engajamento e a retenção do conteúdo exposto (ATKINSON *et al.*, 2000; TONHÃO; COLANZI; STEINMACHER, 2023; TONHÃO *et al.*, 2022; BONETTI *et al.*, 2023). Contudo, a maioria dessas pesquisas se concentra principalmente na engenharia de *software* (ES), o que pode ser problemático ao considerarmos contextos que envolvem sistemas de informação ou sistemas de *software* mais simples, como cliente-servidor.

### **2.3 Exemplos trabalhados na Engenharia de *Software***

No estado atual da arte, poucos trabalhos são encontrados sobre o uso de exemplos trabalhados em ES, muito menos em SDs. Porém, Tonhão, Colanzi e Steinmacher (2023) buscam compreender o uso de exemplos trabalhados na ES. Para tal, foi realizado um *survey* que evidenciou haver pouco uso de exemplos trabalhados reais em sala de aula, devido às dificuldades encontradas na criação dos mesmos. Um questionário também foi feito para os discentes, em que foi apontado que o uso de exemplos trabalhados reais resultou em uma melhoria na percepção e motivação dos alunos.

No mesmo contexto, em um estudo de caso em sala de aula, foram utilizados exemplos trabalhados como material para ensinar a aplicação real de conceitos e para criação de atividades. Os resultados desse estudo de caso mostraram que mais de 80% dos estudantes ficaram mais motivados, sentiram que melhoraram as suas habilidades para o mercado de trabalho e, não suficiente, mais de 90% dos discentes concordaram que os exemplos trabalhados foram relevantes para a qualidade do ensino (TONHÃO; COLANZI; STEINMACHER, 2023).

Um exemplo trabalhado, conforme definido por Tonhão, Colanzi e Steinmacher (2023, p. 1), é um instrumento que comprehende a apresentação de um problema, as etapas envolvidas em sua resolução e o desfecho final. Para ilustrar, considere um exemplo utilizado no

estudo, onde foi feita a explication do conceito da anomalia *large class*, que se refere a uma classe que assume mais responsabilidades do que seria apropriado. Esse problema pode ser identificado por meio da métrica *Lines of Code* (LOC) ou pela análise de coesão da classe. Em seguida, o contexto do problema em um projeto específico é delineado, no qual uma classe chamada *PackWriter* é examinada. Esta classe, responsável por gerar arquivos de pacote para um conjunto específico de objetos do repositório, incorporava uma classe estática, *ObjectToPack*, encarregada de empacotar um objeto.

Para solucionar essa questão, é necessário extrair a classe *ObjectToPack* para uma nova classe, tornando-a independente da classe *PackWriter* e transferindo todas as suas funcionalidades. Como resultado desse processo, a classe *PackWrite* passou a aderir ao princípio de responsabilidade única, no qual uma classe deve ser dedicada a resolver um problema específico, garantindo, assim, sua coesão integral. Dessa forma, o estudo mostrou, de forma eficiente, o que é um exemplo trabalhado, além do fato deles contribuírem grandiosamente para a qualidade do ensino (TONHÃO; COLANZI; STEINMACHER, 2023). No entanto, a criação de exemplos trabalhados não é trivial e apresenta diversos desafios.

Adicionalmente, os exemplos trabalhados utilizados na plataforma de Tonhão *et al.* (2022) foram extraídos de projetos de *Software Livre*, sendo gerados a partir da análise e seleção de casos reais presentes nesses projetos. A coleta dos exemplos envolveu a identificação de situações práticas e desafios enfrentados no desenvolvimento de *software*, com foco em demonstrar a aplicação de conceitos e técnicas de ES. As dificuldades encontradas na criação dos exemplos trabalhados incluíram a necessidade de garantir a relevância e atualidade dos casos selecionados, bem como a adaptação dos exemplos para serem adequados ao contexto educacional, visando proporcionar uma experiência de aprendizado significativa e alinhada com as demandas da disciplina.

A criação ou busca de exemplos trabalhados pode ser, e geralmente é, bem complexa. Uma das conclusões obtidas pela análise do *survey*, realizado pelo estudo anterior, aponta que os professores em geral têm dificuldade em encontrar bons exemplos, de acordo com: a complexidade desejada, adequação à disciplina lecionada, tópico procurado, entre outros aspectos. As principais dificuldades apontadas são a adequação à disciplina e a complexidade, visto que muitos exemplos são: (i) muito grandes e complexos, ou (ii) muito triviais, sem qualquer conexão com a realidade (TONHÃO; COLANZI; STEINMACHER, 2023). Além disso, outra dificuldade relevante apontada está relacionada ao fato de muitos exemplos não serem representativos do mundo real, por estarem desatualizados. Já é conhecido por outro estudo que um fator relevante para que alunos tenham uma boa ideia do que é exigido no mercado de trabalho é o uso de projetos atualizados (PINTO *et al.*, 2017).

Bonetti *et al.* (2023) realizaram uma pesquisa exploratória para aplicar um experimento com professores que nunca haviam utilizado Aprendizagem Baseada em Exemplos (ABE). Professores selecionados implementaram esta abordagem em uma aula sobre modelagem de diagramas de classe. A implementação da ABE foi feita instruindo os professores sobre como a

abordagem proposta funciona. Em seguida, os professores deviam construir o plano de aula, utilizando o portal disponibilizado, para então executarem tal plano. A referida aula foi sucedida por um questionário que visava avaliar as percepções dos aprendizes em relação ao conteúdo apresentado.

A busca ou criação de exemplos trabalhados é uma tarefa consideravelmente complexa. O estudo de Bonetti *et al.* (2023) mostrou que os professores envolvidos colaboraram na seleção de projetos de *Software Livre* e na criação de exemplos relevantes para serem utilizados nas aulas de modelagem de diagramas de classes. As dificuldades e desafios na criação desses exemplos trabalhados incluíram a necessidade de adaptar os exemplos ao nível de experiência dos alunos, garantindo que fossem compreensíveis e relevantes para o aprendizado. Além disso, a criação de exemplos trabalhados errôneos também pode ter sido um desafio, pois requer a identificação e a introdução de erros de forma estratégica para promover a aprendizagem ativa e a correção por parte dos estudantes.

Ainda sobre o mesmo estudo, a análise qualitativa dos resultados teve como objetivo observar os aspectos benéficos e desafiadores desse experimento. Os resultados evidenciaram que a estratégia de ABE não apenas contribuiu significativamente para o grau de compreensão e retenção dos conceitos abordados, mas também facilitou uma maior conexão entre a teoria e a prática. Adicionalmente, o trabalho destacou, como perspectivas futuras, a necessidade de investigar o impacto dessas estratégias em outras áreas da ES. Bonetti *et al.* (2023) investigaram a percepção dos estudantes sobre a ABE no ensino de modelagem de *software*, sendo este um tópico da ES. Por meio de um estudo exploratório, três professores adotaram a ABE em aulas de modelagem de diagrama de classes. Portanto, conclui-se que a ABE é uma abordagem promissora para capacitar os alunos na modelagem de diagramas de classes, contribuindo para práticas pedagógicas mais eficazes no ensino de ES.

## **2.4 Métricas para auxílio à seleção de códigos para exemplos trabalhos**

Para fundamentar a escolha de códigos para a criação de exemplos trabalhados, um recurso frequentemente utilizado é a coleta de medidas ou métricas de software. Dessa forma, é possível selecionar, a partir de projetos relacionados ao tema em estudo, trechos de código adequados para a construção de exemplos didáticos. Nesta seção, abordam-se diferentes métricas e ferramentas que podem ser úteis para essa seleção, considerando tanto aspectos gerais da engenharia de software quanto aspectos específicos do domínio de sistemas distribuídos.

Fu, Lin e Cai (2022) realizaram a análise de métricas em SDs, porém não com o intuito de criar exemplos trabalhados, mas sim de fornecer uma ferramenta que ajude a analisar determinados aspectos de SDs. Dessa forma, não foram encontradas pesquisas sobre a utilização da evolução de código-fonte de SDs por meio de métricas estáticas de código para a criação de exemplos trabalhados na disciplina de ES.

No mesmo contexto, o artigo propõe a ferramenta *DistFax* para a análise de métricas de IPC, que demonstraram correlação com métricas de qualidade de *software*. Dois modelos de Inteligência Artificial (IA) foram desenvolvidos para predizer a condição de qualidade de um SD, dado como entrada as métricas IPC coletadas, onde o modelo supervisionado utiliza a técnica de *Bagging* e o modelo não supervisionado utiliza o algoritmo *K-Means*. As métricas utilizadas na ferramenta são dinâmicas, as quais requerem o uso de relações *happened-before* que, como já explicado no começo desse capítulo, são um desafio por não haver um relógio global.

Entre as métricas de acoplamento empregadas no *DistFax*, destacam-se:

- ***Run-time Message Coupling (RMC)***: Contagem de mensagens enviadas de um processo para outro.
- ***Run-time Class Coupling (RCC)***: Razão entre o número de métodos em uma classe dependentes de métodos em outra classe.
- ***Class Central Coupling (CCC)***: Acoplamento agregado de uma classe em relação a classes em processos remotos.
- ***Interprocess Reuse (IPR)***: Acoplamento entre processos com base em métodos comuns.
- ***Class Communication Load (CCL)***: Carga de comunicação de uma classe individual com outras classes em processos remotos.

A única métrica de coesão utilizada foi a *Process-Level Cohesion* (PLC), que mede as conexões internas dentro de um processo individual.

A complexidade ciclomática representa a quantidade de caminhos distintos no código. Ela pode ser interpretada como o número mínimo de casos de teste necessários para cobrir todos os caminhos do programa sem repetição, servindo para garantir uma cobertura adequada durante os testes. Portanto, quanto menor a complexidade ciclomática, melhor.

No *DistFax*, seis métricas de IPC (cinco de acoplamento e uma de coesão) foram correlacionadas com cinco métricas de qualidade de *software*: tempo de execução, complexidade ciclomática, contagem de caminhos de fluxo de informações, comprimento do caminho do fluxo de informações e superfície de ataque. Observou-se correlação positiva entre: (i) CCC e o tempo de execução, (ii) RMC e CCC com a complexidade ciclomática, e (iii) CCC com a área de ataque. Houve também uma correlação negativa entre CCL e PLC com a área de ataque (FU; LIN; CAI, 2022). Adicionalmente, a correlação positiva entre CCC e o tempo de execução sugere que níveis elevados de acoplamento estão associados a um desempenho inferior.

Jesse, Kuhmuench e Sawant (2023) desenvolveram a ferramenta intitulada *RefactorScore* (JESSE, 2023b), a qual tem o intuito de prover uma métrica de avaliação automática de código que reconhece, consoante os *tokens* de um arquivo, áreas propensas a refatoração. Com

base nessa identificação, a ferramenta gera uma pontuação correspondente, refletindo o potencial de melhoria nessas áreas específicas. A ferramenta *RefactorScore* está presente no modelo *RefactorBert*, o qual aprendeu com a mudança de códigos entre refatorações, mostrando-se eficiente em detectar erros de qualidade de código. Esse tipo de erro está, por exemplo, associado a código fora do padrão de *design* de código adotado.

O modelo *RefactorBert* foi desenvolvido com base em repositórios de linguagem C/C++, contendo mais de um milhão de *commits* ao todo. Contudo, os repositórios examinados nesta monografia não estão presentes no *dataset* (JESSE, 2023a) utilizado pela ferramenta *RefactorScore*, o que torna impraticável seu uso para a coleta de dados relacionados à refatoração nos contextos abordados. Adicionalmente, a falta de documentação sobre a utilização da ferramenta, juntamente com a limitada presença de apenas três *commits* em seu repositório, reforça a inviabilidade de seu emprego. Para adaptar a ferramenta ao contexto de repositórios Java, seria necessário criar um novo *dataset* e treinar o modelo para reconhecer padrões de refatoração em código-fonte do tipo almejado.

## 2.5 Considerações finais

Este capítulo explorou o referencial teórico necessário para compreender os SDs e seus desafios, fornecendo uma base conceitual sólida para a análise e discussão dos resultados desta pesquisa. Os principais conceitos foram destacados, como a definição de SDs, os desafios inerentes à sua implementação, e a interseção entre esses desafios e os princípios da ES.

Ao analisar os desafios específicos enfrentados pelos SDs, destacou-se a importância da heterogeneidade, a ausência de um relógio global, a necessidade de tolerância a falhas e escalabilidade. A discussão sobre o Teorema CAP ressaltou os *trade-offs* cruciais na busca por consistência, disponibilidade e tolerância a partções em um SD.

Explorou-se também o conceito de exemplos trabalhados e sua relevância no contexto da educação em ES. Os exemplos trabalhados oferecem uma abordagem eficaz para ensinar conceitos complexos, proporcionando uma ponte entre a teoria e a prática. No entanto, a criação e seleção de exemplos trabalhados apresentam desafios, especialmente no contexto de SDs.

Além disso, trabalhos relacionados que abordam ferramentas e métricas para análise de qualidade em SDs foram discutidos, destacando o *DistFax* e suas métricas de acoplamento, coesão e correlações significativas com desempenho e segurança do *software*.

No contexto educacional, evidenciou-se a escassez de pesquisas sobre exemplos trabalhados em SDs, ressaltando a importância de abordagens pedagógicas que integrem de maneira eficaz os desafios específicos desses sistemas.

No próximo capítulo, é apresentada a metodologia adotada para a realização deste estudo, detalhando os procedimentos e técnicas utilizados na coleta e análise dos dados.

### 3 METODOLOGIA

Este capítulo delineia as questões de pesquisa e a abordagem da pesquisa. Além disso, não só explica as métricas e critérios utilizados, mas também as ferramentas e critérios de seleção de repositórios analisados.

#### 3.1 Questões de pesquisa

As questões de pesquisa foram formuladas com base nos objetivos geral e específicos desta monografia, para direcionar a investigação sobre a seleção de códigos representativos de melhorias de *software* em projetos de SDs, utilizando métricas específicas de qualidade de código da ES, como apresentado abaixo:

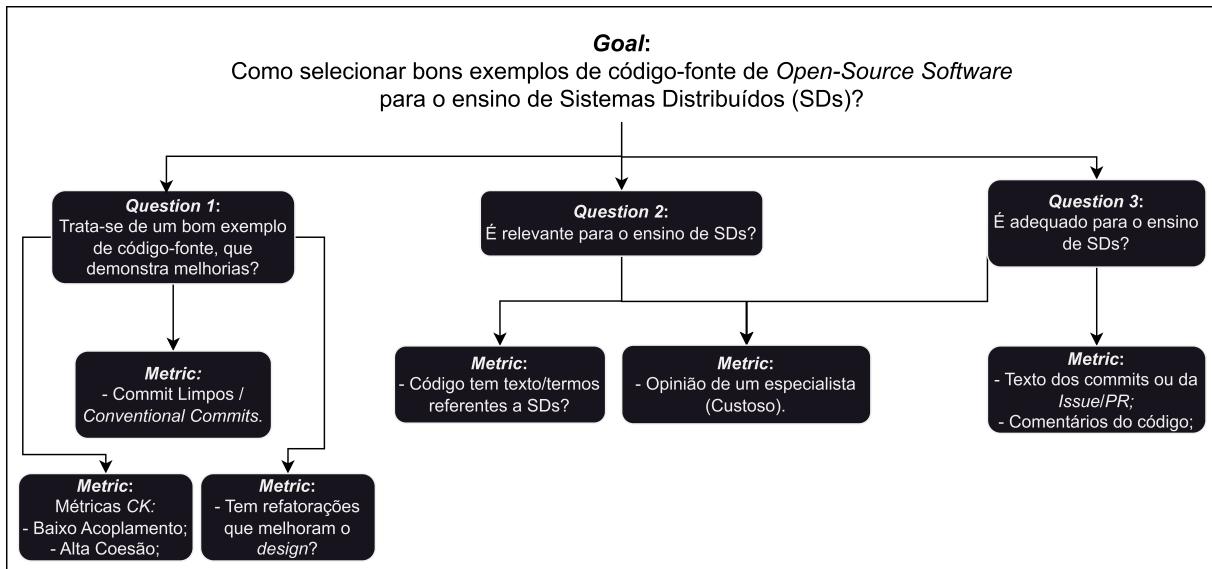
1. Q1: O exemplo de código-fonte demonstra melhorias na qualidade do *design* do código?
2. Q2: O exemplo é relevante para o ensino de SD?
3. Q3: O exemplo é didaticamente adequado para o ensino de SD?

Para garantir que essas etapas estejam alinhadas com os objetivos educacionais e técnicos do estudo, o processo de avaliação dos candidatos é estruturado com base no modelo *Goal, Question, Metric* (GQM). Este modelo é organizado em três níveis distintos. No nível conceitual, define-se um objetivo específico para o objeto de estudo, considerando diferentes modelos de qualidade e perspectivas relativas ao ambiente em questão. No nível operacional, são formuladas questões que desenvolvem modelos para avaliar o progresso em relação ao objetivo definido, focando na caracterização e avaliação desse objetivo. Finalmente, no nível quantitativo, associam-se métricas específicas a cada questão, permitindo a obtenção de respostas mensuráveis para as perguntas formuladas. Essa abordagem estruturada ajuda a garantir que a avaliação das métricas seja tanto relevante quanto precisa, alinhando os objetivos do estudo com as ferramentas e métodos de medição apropriados. Portanto, o método pode ser caracterizado com base na abordagem GQM, conforme ilustrado no diagrama da Figura 1.

#### 3.2 Abordagem proposta

O presente trabalho define uma heurística para identificar exemplos de código-fonte representativos da melhoria da qualidade de *software* em SDs, por meio da análise de métricas de código utilizadas na ES. A heurística visa auxiliar na seleção de trechos de código que permitam a criação de exemplos trabalhados que demonstrem como o código em SDs se adapta e evolui ao longo do tempo.

**Figura 1 – Relação entre objetivo, questões e métricas, conforme a abordagem GQM.**



**Fonte: Autoria própria.**

A heurística desenvolvida nesta pesquisa baseia-se nas melhorias que podem ser identificadas através das métricas selecionadas. As métricas devem ser capazes de capturar informações relacionadas à evolução do código em SDs. Logo, são empregadas ferramentas específicas em repositórios de código aberto selecionados conforme indicado na Seção 3.3.

### 3.2.1 Métricas e critérios de qualidade e relevância utilizados

Para viabilizar a seleção de exemplos trabalhados, foram selecionadas várias métricas e critérios, aplicados com o intuito de avaliar a qualidade e a relevância dos trechos de código:

- **Métricas Chidamber-Kemerer (CK)** (Acoplamento e Coesão) (Q1): Essas métricas buscam entender como componentes do código interagem entre si e como funções e classes são internamente coesas. Melhorias nessas métricas após revisões ou refatorações são evidências de que o código evoluiu para uma estrutura mais robusta e manutenível, ressaltando práticas de *design* sólidas úteis para o ensino de ES e SD.
- **Métrica McCabe** (Q1): Esta métrica avalia a complexidade ciclomática do código. Exemplos de código que mostram uma redução nesta métrica após refatorações são particularmente valiosos, pois indicam uma simplificação do código, tornando-o mais fácil de entender e manter. Este indicador permite verificar melhorias qualitativas que facilitam o ensino de conceitos de *design* eficiente e gestão de complexidade.
- **Padrões de Refatoração** (Q1, Q3): A detecção de refatorações significativas é um indicativo de esforços conscientes para melhorar o *design* do código. Tais códigos podem ser usados didaticamente para mostrar como problemas comuns de *design* podem ser

resolvidos de maneira eficaz, facilitando o aprendizado através da revisão de práticas reais de ES.

- **Análise de Commits** (Q1): A análise dos *commits* fornece um histórico direto das mudanças implementadas. *Commits* que refletem refatorações substanciais e melhorias no *design* do código são indicativos de uma melhoria na qualidade do código, entretanto podem não refletir uma refatoração por completo.
- **Termos e Textos Relacionados a SD** (Q2, Q3): A presença de termos específicos de SD nos comentários do código, na documentação, e nos nomes de métodos e classes destaca a relevância temática do exemplo, facilitando a correlação entre o contexto do código e o currículo de SD. Essa abordagem garante que o exemplo seja contextualmente apropriado para ensinar conceitos específicos de SD. Além disso, a métrica *MethodInvocationCounter* (SILVA, 2024b), desenvolvida nesta pesquisa e adicionada ao conjunto de métricas CK (ANICHE, 2015), desempenha um papel crucial neste processo. Esta métrica analisa a frequência de invocação de métodos dentro de uma classe, fornecendo *insights* detalhados sobre as interações entre métodos e a complexidade das dinâmicas de chamadas.
- **Avaliação de Especialistas** (Q2, Q3): As opiniões de especialistas em SD fornecem uma validação externa da relevância e adequação didática do código. Especialistas podem avaliar se o exemplo demonstra práticas recomendadas e se explica de forma eficaz os conceitos de SD, o que é fundamental para sua utilização em um contexto educacional.
- **Comentários no Código** (Q3): Comentários claros e educativos no código ajudam a esclarecer a lógica e as decisões de *design*, tornando o código mais compreensível para estudantes e educadores. A qualidade dos comentários é, portanto, diretamente relacionada à utilidade didática do código.
- **Texto dos Commits** (Q3): Os textos dos *commits* buscam oferecer um contexto sobre o raciocínio por trás das alterações no código, servindo para identificar *commits* que potencialmente envolvem refatorações significativas. O texto também permite correlacionar a refatoração com algum relatório de *Common Vulnerabilities and Exposures* (CVE) (DETAILS, 1999), visando explorar questões de segurança no código.
- **Texto dos Pedidos de Alteração (*Pull Requests*)** (Q3): Assim como os *commits*, os textos nos *pull requests* podem esclarecer as motivações para mudanças específicas, destacando discussões que podem ser usadas didaticamente para ilustrar como e por que certas decisões de *design* são tomadas em ambientes colaborativos de desenvolvimento de *software*, além do problema que elas resolvem.

Com relação às métricas CK, Chidamber e Kemerer (1994) apresentaram um conjunto de métricas para avaliação do *design* de código orientado a objetos, introduzindo métricas que permitem compreender a complexidade e qualidade do código-fonte. As principais métricas apresentadas no artigo são listadas a seguir.

- **Coupling Between Object (CBO)**: Reflete o grau de acoplamento, ou seja, as dependências que uma classe específica possui, indicando a quantidade de classes diretamente associadas a ela, já que utiliza métodos dessas classes. Um valor elevado sugere maior complexidade e menor flexibilidade, pois mudanças na classe podem impactar várias outras. Não suficiente, demasiado acoplamento diminui o grau de modularidade de uma classe. Observar uma redução no CBO ao longo da evolução do código é um indicativo positivo de melhorias na qualidade do código. Um esforço para reduzir o CBO pode levar a um *design* de software mais modular, onde as mudanças em uma parte do sistema têm menos probabilidade de exigir mudanças em outra.

Uma variação da implementação do CBO, também denominada *Coupling Between Object Modified (CBOM)*, considera a dependência de classes como uma referência a um objeto do mesmo tipo. Isso significa que, ao adicionar uma simples chamada de método da classe, a métrica é incrementada.

- **Depth of Inheritance Tree (DIT)**: Reflete a profundidade da árvore de herança à qual uma classe depende, indicando a quantidade de ancestrais que uma classe tem. Um valor elevado de DIT pode indicar uma maior complexidade e potencial para efeitos colaterais ao realizar alterações em alguma das superclasses.
- **Lack of Cohesion in Methods (LCOM)**: Representa o coeficiente de falta de coesão em métodos, mensurando a coesão em uma classe. O valor varia de 0 a 1, onde 0 indica alta coesão e 1 indica falta de coesão. Baixos valores são desejados, indicando que os métodos estão fortemente inter-relacionados, contribuindo para uma classe mais coesa. Uma classe altamente coesa tem responsabilidades e propósitos claros e bem definidos, enquanto uma classe com baixa coesão tem muitas responsabilidades diferentes e pouco relacionadas.
- **Number of Children (NOC)**: Indica o número de filhas diretas de uma classe. Um valor alto pode indicar maior reusabilidade. Isso implica que a referida classe desempenha uma função importante, dada a dependência de outras classes sobre ela.
- **Response for a Class (RFC)**: Refere-se ao tamanho do conjunto de respostas de uma classe, ou seja, o número de métodos que podem ser potencialmente executados em resposta a uma mensagem recebida por um objeto dessa classe. Um valor menor de RFC indica que uma classe tem menos comportamentos e, potencialmente, um menor grau de complexidade. Classes com RFC baixo tendem a ser mais coesas e menos

acopladas, pois interagem com menos outras classes e componentes, facilitando assim a manutenção e a testabilidade.

- **Weight Method Class (WMC) ou McCabe's Complexity:** Simboliza a soma dos valores de complexidade dos métodos de uma classe específica. Um valor elevado indica que a classe pode ser complexa, com múltiplos métodos, implicando em um custo significativo para o desenvolvimento e manutenção. Além disso, a existência de vários métodos associados a uma classe sugere que ela pode ser menos genérica, sem mencionar o possível impacto nas filhas dessa classe. Classes com WMC baixo tendem a ser mais coesas, o que significa que elas têm um propósito bem definido e limitado. A redução do WMC pode ser um sinal de que a classe está se tornando mais focada em suas responsabilidades, o que facilita o entendimento, a manutenção e a extensão do código.

No estágio inicial desta pesquisa, optou-se por selecionar um conjunto específico de métricas, incluindo CBO, CBOM, RFC e WMC. A escolha dessas métricas foi motivada pela necessidade de obter uma visão abrangente dos diferentes aspectos da qualidade do código. As métricas CBO e CBOM foram escolhidas para fornecer *insights* sobre o nível de acoplamento em classes ou métodos, uma característica frequentemente associada à dificuldade de manutenção e evolução do sistema. Por outro lado, a métrica RFC foi selecionada para medir o grau de comunicação entre os elementos do código, o que pode influenciar a complexidade e a modularidade do sistema. A métrica WMC foi incluída para avaliar a complexidade dos métodos em uma classe, um indicador importante da comprehensibilidade e da facilidade de manutenção do código.

Contudo, durante o desenvolvimento do estudo, decidiu-se remover a métrica CBOM da análise. Essa decisão foi tomada devido à percepção de que a inclusão de CBOM poderia introduzir poluição nos dados, aumentando a complexidade artificialmente. Isso ocorre, por exemplo, quando a adição de uma chamada a um método da própria classe afeta negativamente a métrica. Dessa forma, a exclusão de CBOM visou manter a integridade e a precisão das métricas escolhidas, proporcionando uma análise mais clara e objetiva da evolução do código.

Essa seleção de métricas foi fundamentada na literatura existente, como demonstrado nos estudos de Tonhão, Colanzi e Steinmacher (2023) e Tonhão *et al.* (2022), que analisaram a importância de métricas de acoplamento e complexidade para identificar melhorias em sistemas de software e em contextos educacionais. Em particular, Tonhão, Colanzi e Steinmacher (2023) realizaram um *survey* com docentes de ES para entender o uso de exemplos trabalhados, destacando as dificuldades enfrentadas na criação desses exemplos e os benefícios para a motivação e o aprendizado dos alunos. Adicionalmente, Tonhão *et al.* (2022) implementaram uma plataforma de exemplos trabalhados integrados com gamificação, que provou ser eficaz para melhorar o engajamento e a compreensão dos conceitos complexos em ES. Esses estudos sublinham a eficácia de utilizar métricas como CBO e WMC para analisar a qualidade do código.

e promover boas práticas de desenvolvimento, contribuindo significativamente para o ensino e a prática da ES.

### 3.2.2 Ferramentas

As ferramentas utilizadas na pesquisa são apresentadas a seguir, descrevendo brevemente sua utilidade para o desenvolvimento do trabalho.

O CK (ANICHE, 2015) é utilizada para analisar a qualidade do código-fonte em projetos Java (GOSLING; JOY; STEELE, 1996) por meio de métricas estáticas. Essas métricas fornecem informações valiosas sobre características como complexidade, acoplamento e coesão de classes. Essa ferramenta em questão oferece mais de 35 métricas, como CBO, DIT, LCOM, RFC e WMC. Entretanto, se destaca uma métrica específica, denominada *MethodInvocationCounter*, que foi implementada neste trabalho e adicionada ao conjunto de métricas da ferramenta CK (SILVA, 2024b). Ao integrar a métrica *MethodInvocationCounter*, é possível obter uma compreensão mais profunda do comportamento dos métodos das classes e aprimorar a análise da relevância do código com o intuito de refletir conceitos de SD.

A biblioteca *PyDriller* (SPADINI; ANICHE; BACCHELLI, 2018), disponível para a linguagem de programação *Python*, é um recurso eficiente para percorrer e analisar repositórios hospedados no *GitHub*, feitos em qualquer linguagem. Ela possibilita aos pesquisadores e desenvolvedores percorrer a árvore de *commits* de um repositório e extrair informações como histórico, autoria e mensagens de *commits*, além de detalhes específicos do estado do código em um determinado momento, contribuindo assim para a pesquisa e prática em ES no contexto de sistemas complexos e interconectados. À medida que se percorre a árvore de *commits*, a biblioteca modifica o clone local do repositório para corresponder à *branch* associada ao *commit* atual, possibilitando a integração com a execução do *CK* para gerar métricas correspondentes ao estado do código naquele momento.

O *RefactoringMiner* (TSANTALIS *et al.*, 2018) é uma ferramenta no contexto da ES, especializada em identificar e analisar refatorações de código-fonte. Essa ferramenta fornece uma compreensão aprofundada das mudanças realizadas em um código ao longo do tempo. Com a sua capacidade de reconhecer padrões de refatoração, o *RefactoringMiner* permite que os desenvolvedores analisem como o código foi modificado, de modo a melhorar a qualidade, manutenibilidade e eficiência do *software*.

Dentre os padrões de refatorações identificados pelo *RefactoringMiner*, destacam-se as refatorações do tipo *Extract Method*, *Extract Class*, *Pull Up Method*, *Push Down Method* e *Extract Superclass*. Esses tipos de refatorações são fundamentais para a melhoria da qualidade do código, promovendo a modularização, a reutilização e a manutenibilidade do *software*.

A refatoração *Extract Method* visa reduzir a complexidade de métodos longos, isolando partes do código em novos métodos para melhor compreensão e reuso. *Extract Class* ajuda a combater o antipadrão de classes nomeadas “*God Classes*”, distribuindo responsabilidades de

uma classe sobrecarregada para novas classes, o que facilita a manutenção e a expansão do sistema. As refatorações *Pull Up Method* e *Push Down Method* são importantes para otimizar a hierarquia de classes, movendo métodos para a classe base ou derivada, respectivamente, promovendo um melhor aproveitamento da herança e da especialização. Por fim, *Extract Superclass* permite a criação de abstrações mais genéricas ao identificar e extrair um conjunto comum de características e comportamentos, facilitando o gerenciamento de dependências e a evolução do *software*.

As práticas explicadas nos parágrafos anteriores são essenciais para a ES, pois contribuem diretamente para a coesão, o acoplamento e encapsulamento do código, pilares para o desenvolvimento de sistemas de *software* robustos, escaláveis e facilmente manutencíveis.

Outra ferramenta utilizada foi o *Gemini* (GOOGLE, 2023a) é um modelo de IA generativa desenvolvido pela *Google* que permite processar e analisar grandes volumes de dados textuais de maneira eficiente e precisa. Utilizando técnicas avançadas de aprendizado de máquina, ele é capaz de compreender contexto e gerar respostas coerentes e pertinentes para questões a ele apresentadas, com base nos dados fornecidos. A principal aplicação da ferramenta (GOOGLE, 2023b) reside na análise de dados textuais, extração de informações e geração de conteúdos, sendo amplamente utilizada em áreas como processamento de linguagem natural, geração de texto e análise semântica.

A escolha por sua utilização em comparação com outros modelos de IA justifica-se pelo fato de que, na versão 1.5 Flash, o *Gemini* oferece uma *Application Programming Interface* (API) gratuita com um maior limite de tokens de entrada, totalizando 1.048.576 tokens (AI, 2024). Em contraste, a API do ChatGPT (OPENAI, 2024a), na versão 3.5, permite apenas 4.096 tokens de entrada (OPENAI, 2024b). Esta maior capacidade do *Gemini* é importante para a presente abordagem devido à necessidade de processar como entrada não apenas as métricas extraídas, como também os currículos de referência, permitindo uma análise mais abrangente e integrada dos dados. Ambas as APIs representam versões dos modelos que, embora não sejam os melhores disponíveis, destacam-se como as melhores opções gratuitas para uso.

### 3.2.3 Critérios de seleção de repositórios

Os critérios para a seleção de repositórios foram determinados pelos seguintes fatores:

- **Ser um repositório Java:** Critério obrigatório para viabilizar o uso da ferramenta Java que avalia métricas CK.
- **Ser de software-livre (*open-source*):** A escolha por repositórios abertos, hospedados em plataformas como o *GitHub*, tem como propósito assegurar a transparência e acessibilidade do código-fonte. Ao selecionar um repositório disponível no *GitHub*, é viável, por meio da biblioteca *PyDriller*, percorrer a árvore de *commits*. Desse modo, em cada *commit*, possibilita-se gerar as métricas CK.

- **Ser ativamente mantido e atualizado:** A ativa manutenção é necessária para garantir que os repositórios estejam alinhados com as práticas e avanços mais recentes.
- **Ser utilizado em aplicações do mundo real:** A seleção de repositórios empregados em contextos práticos proporciona uma análise mais relevante e aplicável às situações reais.

### 3.3 Métodos

Este trabalho de pesquisa adota uma abordagem exploratória com o propósito de desenvolver uma heurística que possibilite a seleção de códigos representativos de melhorias de *software*, de acordo com métricas específicas de código em projetos de SDs. O objetivo principal é aprofundar a compreensão desses casos, proporcionando uma análise detalhada da evolução do código em resposta a mudanças nas métricas escolhidas. Esse estudo visa contribuir diretamente para a seleção de candidatos que após todo o processo de avaliação, de forma manual, pode ser utilizados para a criação de exemplos trabalhados aplicáveis ao contexto da ES do domínio de SD.

A natureza exploratória da pesquisa pretende proporcionar uma visão mais ampla sobre como as métricas de código em projetos de SDs refletem melhorias significativas. A heurística desenvolvida visa identificar casos exemplares de evolução de código, considerando métricas e padrões de refatoração específicos, selecionados com base nas melhores práticas e no entendimento profundo das características inerentes aos SDs.

Para alcançar os objetivos propostos, foram analisados repositórios de código aberto relevantes para a área de SDs. As análises consideraram a evolução do código-fonte por meio das métricas selecionadas e as mudanças no *design* do código, identificadas através da detecção de padrões de refatoração desejados. Entretanto, como a metodologia se baseia exclusivamente em métricas estáticas, o comportamento real do sistema não é avaliado. Isso limita a observação de aspectos como desempenho, quantidade e eficiência da troca de mensagens, e taxa de exceções geradas.

A introdução da métrica *MethodInvocationCounter*, desenvolvida e incorporada ao conjunto de métricas da ferramenta CK (SILVA, 2024b), possibilitou o uso da API do Gemini (GOOGLE, 2023b) para identificar candidatos que abordam tópicos de SD. A análise dos termos permite compreender o comportamento dos métodos e permite ao modelo de IA refinar a lista de candidatos, correlacionando termos e comportamentos com tópicos de SD presentes nos currículos do NSF/IEEE-TCPP (NSF/TCPP, 2023a; NSF/TCPP, 2023b).

Os trechos de código identificados pelo *Gemini* como relevantes para SD foram avaliados por analistas de forma cega, utilizando os formulários apresentados no Apêndice A. Esses especialistas avaliaram os trechos em termos de SD e ES, baseando-se apenas no *commit* e na classe alvo, e responderam a uma série de perguntas para compreender a relevância e a

utilidade educacional dos candidatos, validando a qualidade da heurística desenvolvida. Assim, a metodologia adotada possibilita uma compreensão holística da relação entre a qualidade do código e as métricas específicas selecionadas.

### 3.3.1 Processo de seleção

O processo de seleção dos exemplos de código para uso didático em SD segue uma abordagem iterativa, projetada para garantir a relevância e a qualidade educacional dos exemplos escolhidos. Este processo envolve várias etapas, iniciando com a coleta de dados de projetos de *software* livre que são amplamente utilizados na indústria para desenvolver aplicações distribuídas.

Inicialmente, selecionamos manualmente projetos reconhecidos, como Kafka (Apache Software Foundation, 2011) e ZooKeeper (ZK) (Apache Software Foundation, 2008), com base nas recomendações de um professor especialista no domínio de SD. Ambos os sistemas são extensivamente utilizados no mercado de trabalho para o desenvolvimento de aplicações distribuídas, e foram escolhidos devido à sua complexidade técnica, aplicação extensiva no mercado e histórico de manutenção ativa. Essas características os tornam ideais para estudar a evolução da qualidade do código e demonstrar a aplicação de refatorações significativas, fornecendo assim uma base sólida para a criação de exemplos didáticos em SD.

Em um estágio mais avançado da pesquisa, após a análise do Kafka e do ZK, identificou-se uma limitação associada ao fato de ambos serem produtos da *Apache*. Essa constatação levantou a possibilidade de um viés na análise dos projetos. Para mitigar essa questão, foi desenvolvido o *script* denominado *repositories\_picker.py* (SILVA, 2024c), que aplica os critérios mencionados anteriormente. Portanto, esse *script* busca repositórios *open-source* no *Github* (GITHUB, 2024) relacionados ao escopo de SD, desenvolvidos na linguagem Java e que tenham sido atualizados nos últimos 180 dias. Mais detalhes serão fornecidos na Seção 4.2.

Após a seleção dos repositórios, procedemos com a extração das métricas de *software* utilizando a ferramenta CK. Em seguida, todo o histórico das métricas CK são agrupadas e analisadas, de modo a ajudar a identificar componentes de código que demonstram características importantes como alta complexidade, alto acoplamento ou baixa coesão. Estes componentes são potenciais candidatos para refatoração e são usados para ilustrar princípios de ES em um contexto real de SD.

O histórico agrupado das métricas de cada classe e método permite então verificar os trechos de código que tiveram uma melhoria nas métricas ao longo do tempo. Não suficiente, concomitantemente, padrões de refatoração são detectados nesses códigos, resultando em uma lista de candidatos que, do ponto de vista da ES, demonstram evolução no *design* do código. Esta lista detalha o nome da classe ou método, o percentual de melhoria das métricas, os valores das métricas antes e após a refatoração, além dos termos extraídos dos métodos invocados pela métrica *MethodInvocationCounter* e os padrões de refatoração identificados.

Os nomes e termos extraídos pela métrica *MethodInvocationCounter* de cada candidato são então fornecidos como entrada ao Google Gemini, juntamente com uma lista de currículos de interesse definidos pelo NSF/IEEE-TCPP (NSF/TCPP, 2023a; NSF/TCPP, 2023b). Isso permite analisar a relação dos termos com tópicos de SD e entender seu comportamento, filtrando a lista de candidatos para incluir aqueles que, além de relevantes para a ES, também possuem relevância no contexto de SDs.

Por fim, a lista de candidatos com potencial relevância para ES e SD é revisada por um especialista em SD. Esta revisão é crucial, pois a ferramenta WEM, que implementa a heurística, não é totalmente automatizada. A análise do especialista aborda a subjetividade do processo de avaliação, garantindo não apenas a adequação técnica dos candidatos, mas também sua capacidade de ilustrar claramente conceitos-chave de *design* e manutenção de *software* em ambientes distribuídos.

Este processo iterativo de seleção e revisão assegura que os exemplos finais escolhidos para ensino sejam não apenas tecnicamente sólidos, mas também extremamente relevantes e úteis para estudantes que buscam entender os desafios e soluções na ES.

### 3.3.2 Refinamento da Heurística

O processo de refinamento da heurística envolveu uma série de iterações de definição para assegurar a seleção precisa de exemplos de código que representassem melhorias significativas em projetos de SDs. Inicialmente, foram selecionadas métricas que melhor representassem aspectos críticos da qualidade do código, tais como CBO, RFC e WMC. Estas métricas foram escolhidas devido à sua capacidade de refletir a complexidade e o acoplamento do código, aspectos frequentemente relacionados com melhorias substanciais no código. Embora a métrica CBOM também tenha sido considerada, ela foi excluída do conjunto devido ao seu comportamento indesejado: a adição de uma simples chamada de método à classe pode incrementar significativamente essa métrica, resultando em uma poluição dos dados. Outras métricas, como DIT, LCOM e LOC, foram avaliadas, mas o trio CBO, RFC e WMC demonstrou já seria, na teoria, eficaz na identificação de melhorias significativas.

Para detectar quedas nas métricas, inicialmente foram definidos *thresholds* absolutos e gerais para todas as métricas, além da detecção de padrões específicos de refatoração. No entanto, observou-se que a métrica CBO sozinha era mais adequada para identificar boas refatorações, pois uma redução na complexidade é um forte indicativo de melhorias no código. A aplicação de múltiplos *thresholds* com valores absolutos revelou-se problemática, uma vez que não refletia de maneira confiável as melhorias devido à variação significativa nas quedas das métricas conforme o tamanho do repositório. Portanto, a remoção dos *thresholds* absolutos e a adoção de limiares relativos, baseados em percentuais, contribuiu para uma heurística mais eficaz, capaz de se adaptar às variações entre diferentes repositórios e identificar melhorias reais no código.

No caso do ZK, não se observou uma queda superior a 15% entre os *commits*, mantendo os padrões de refatoração desejados. O mesmo resultado foi observado para um limiar de 10%. Em contraste, a consideração de *thresholds* relativos ofereceu uma abordagem mais flexível e adaptativa. No entanto, para simplificar a heurística, optou-se por não utilizar limitares percentuais e, em vez disso, aceitar qualquer queda nas métricas como um critério para inclusão dos trechos de código na lista de candidatos.

Adicionalmente, foram analisados padrões de refatoração. No estágio inicial, a heurística era restritiva, incluindo apenas os candidatos que apresentavam padrões específicos de refatoração, tais como *Extract Method*, *Extract Class*, *Pull Up Method*, *Push Down Method* e *Extract Superclass*, que são tradicionalmente associados a melhorias na qualidade do design do projeto. Contudo, essa abordagem revelou-se excessivamente restritiva, resultando em uma quantidade zero de candidatos no ZK e considerando que haveria queda no CBO e apenas os padrões de refatoração desejados eram detectados. Em resposta a essa limitação, a heurística foi flexibilizada para permitir a detecção de todos os tipos de padrões de refatoração, não se limitando apenas aos mais prováveis de estar associados aos melhores tipos de refatoração.

O resultado dessas iterações é uma heurística que, embora menos rigorosa na seleção baseada exclusivamente nas métricas, integra de forma mais robusta a API do Gemini (SILVA, 2024a). Ao utilizar como *input* os tópicos de HiPC do NSF/IEEE-TCPP, a heurística não se limita apenas ao nome das classes e aos padrões de refatoração detectados, mas também incorpora os termos extraídos pela métrica *MethodInvocationCounter* (SILVA, 2024b). Essa abordagem torna a heurística mais adaptativa e eficaz, permitindo ao Gemini analisar o comportamento das classes e os termos associados. Dessa forma, a lista de candidatos é filtrada com base na relevância dos tópicos de SD, proporcionando uma seleção mais refinada e alinhada com as melhores práticas da área.

### 3.3.3 Avaliação de candidatos

Os candidatos selecionados passaram por um rigoroso processo de avaliação, que envolveu diversas etapas delineadas para garantir sua relevância educacional e técnica. Esse processo confirmou que os trechos de código podem ser utilizados efetivamente como exemplos trabalhados. Esse processo envolve:

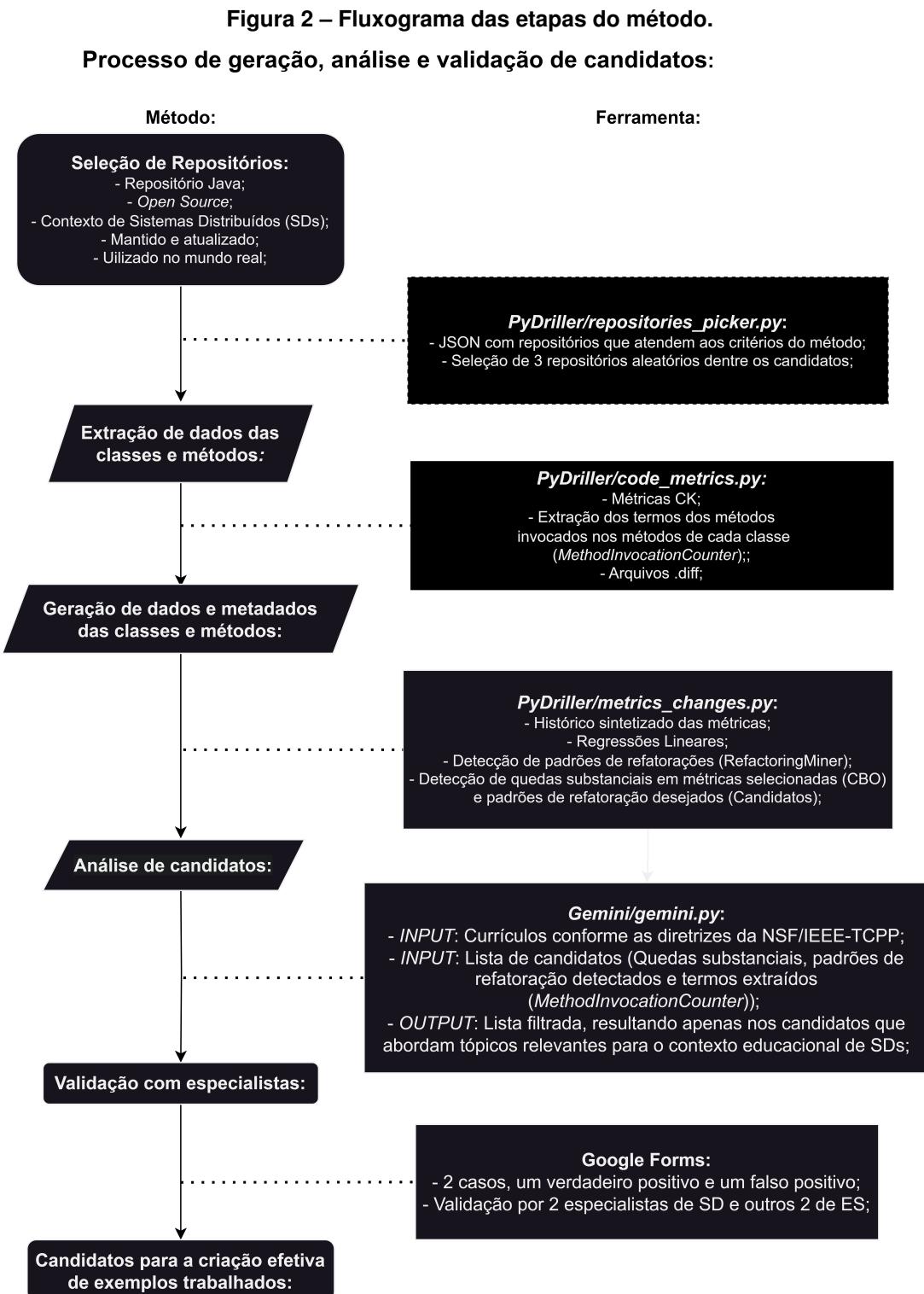
- **Análise das métricas** (Q1): Esta etapa compreende a análise detalhada das métricas de qualidade do código geradas pelo CK, desde o começo do projeto, até que melhorias substanciais nas métricas sejam identificadas.
- **Análise de atividades de refatoração** (Q1): Após a identificação de uma melhoria significativa nas métricas, o padrão de refatoração detectado pela ferramenta *RefactoringMiner* é analisado. Esta análise do tipo e da extensão da refatoração implementada

visa compreender seu impacto sobre o *design* do código e avaliar sua relevância no contexto da ES.

- **Análise de termos do código-fonte** (Q2 e Q3): Nesta etapa, analisamos os termos dos métodos das classes candidatas. Esses termos refletem a estrutura e o comportamento das classes, pois incluem os métodos na ordem em que são invocados dentro de cada classe, bem como a frequência com que esses métodos são chamados. A integração com a API do Google Gemini permite que os tópicos de SD mais relevantes em currículos sejam usados como entrada, relacionando-os com os termos extraídos do código. A extração desses termos é possibilitada pela implementação de uma nova métrica na ferramenta CK (SILVA, 2024b), que utiliza uma *Abstract Syntax Tree* (AST) para percorrer o código-fonte analisado.
- **Análise de código-fonte** (Q2 e Q3): Aprofundou-se a análise investigando os arquivos *.diff* que documentam a refatoração, a qual é realizada de forma manual. Avaliou-se a relevância das mudanças considerando diversos aspectos: o impacto das métricas antes e após a refatoração, a natureza do padrão de refatoração adotado, a relevância dos nomes das classes e métodos modificados para conceitos fundamentais de SD e os impactos que aquela refatoração gera para o sistema.
- **Avaliação por especialistas** (Q2 e Q3): Finalmente, dois especialista em SD e outros dois de ES foram selecionados para revisar os candidatos, confirmando a relevância das refatorações, tanto do ponto de vista técnico quanto educacional. Os especialistas em SD também avaliaram se as mudanças são didaticamente apropriadas para serem utilizadas como exemplos no ensino de SD, garantindo que os exemplos não apenas ilustrem conceitos teóricos, mas também apresentem aplicações práticas de como resolver desafios enfrentados por SD.

Este processo de seleção e avaliação garante que cada código candidato para uso como exemplo trabalhado não apenas contribua para o entendimento técnico dos alunos, mas também estimule a aplicação prática desses conhecimentos em cenários reais de desenvolvimento de *software* distribuído. As etapas de seleção, extração, geração e análise de candidatos podem ser visualmente representadas conforme o diagrama de fluxo da Figura 2.

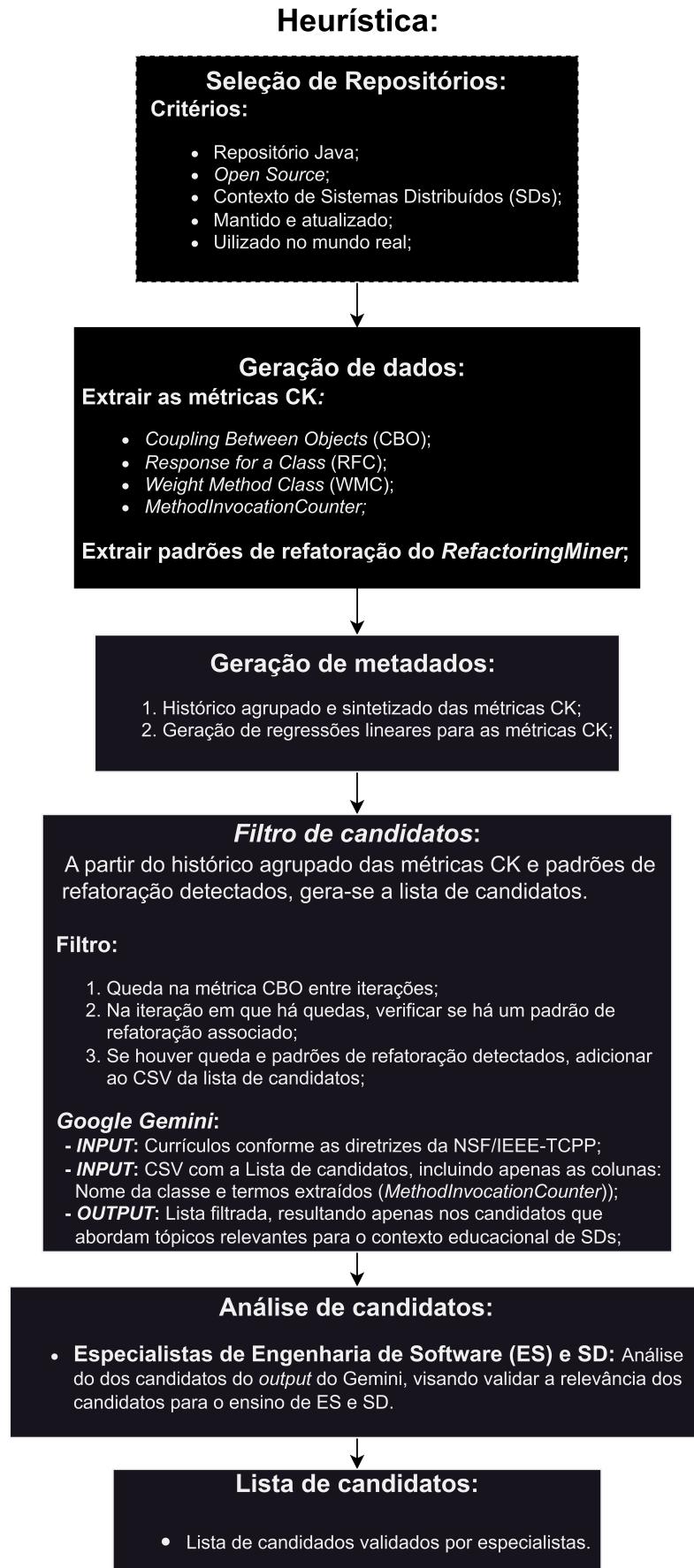
A heurística empregada no processo de geração de dados, metadados, seleção e avaliação de candidatos para exemplos trabalhados é projetada para assegurar que os códigos selecionados não apenas promovam uma compreensão de conceitos de SD pelos alunos, mas também incentivem a aplicação prática desses conhecimentos em cenários reais de desenvolvimento de *software* distribuído. O processo de seleção é estruturado em várias etapas, conforme detalhado no diagrama da Figura 3. Essa abordagem iterativa e adaptativa permite refinar a escolha dos exemplos trabalhados, garantindo que eles sejam representativos de melhorias significativas e alinhados com as melhores práticas da área. A heurística não só contribui para



**Fonte: Autoria própria.**

a compreensão teórica, mas também para a aplicação prática, facilitando o aprendizado efetivo e a preparação dos alunos para enfrentar desafios reais em projetos de SD.

**Figura 3 – Diagrama da heurística.**



## 4 RESULTADOS

Neste capítulo são apresentados e discutidos os resultados obtidos a partir da geração de dados e análises de código realizadas no *Apache Kafka*, ZK e *CorfuDB*. A investigação foi orientada por critérios rigorosos de seleção de repositórios e pela aplicação de princípios de ES para avaliar a eficácia das refatorações no contexto de SD. Além disso, exploramos as limitações encontradas durante a análise, como a impossibilidade de utilizar métricas dinâmicas e a presença de *commits* muito extensos e poluídos, e discutimos as implicações dessas restrições para a validade e o alcance das conclusões.

### 4.1 Repositórios selecionados

Conforme mencionado anteriormente, os projetos *Apache Kafka* (Apache Software Foundation, 2011) e ZK (Apache Software Foundation, 2008) foram inicialmente escolhidos de forma manual, com base na sugestão de um especialista em SD. Ambos atendem a todos os critérios de seleção definidos na Seção 3.2.3 e também estão presentes na lista de repositórios gerada pelo script *PyDriller/repositories\_picker.py* (SILVA, 2024c). Na data de escrita desta monografia, o programa identificou 1.263 repositórios relacionados a SD; entretanto, apenas 53 desses repositórios atendem aos critérios de seleção estabelecidos. Entre os repositórios selecionados e analisados neste estudo, destacam-se *CorfuDB* (CORFUDB, 2013), *Scalecube-Services* (SCALECUBE, 2018) e *Molecular-Java* (JAVA, 2017).

A seguir são apresentados os projetos de código aberto disponíveis no GitHub (GITHUB, 2008), que foram analisados neste trabalho de pesquisa.

#### 4.1.1 *Apache Kafka*

O *Apache Kafka* (Apache Software Foundation, 2011) é um sistema de mensagens distribuídas, baseado no modelo *publish-subscribe*, amplamente utilizado em infraestruturas de processamento de eventos em tempo real. Segundo Kreps Neha Narkhede (2010), o Kafka é projetado para lidar com fluxos de dados em grande escala, permitindo que organizações processem, armazenem e transmitam dados de maneira eficiente. Ele opera em um modelo de *log* distribuído, onde as mensagens são registradas em *logs* (ou tópicos) que podem ser divididos em partícipes para permitir a distribuição e o processamento paralelo de mensagens.

A estrutura do Kafka inclui vários componentes essenciais:

- **Tópicos:** Canais de mensagens nos quais os dados são publicados e armazenados.
- **Produtores:** Responsáveis por publicar mensagens nos tópicos do Kafka.
- **Partições:** Dividem os tópicos para facilitar a distribuição e o processamento paralelo.

- **Corretores (Brokers):** Servidores que armazenam e distribuem as mensagens, desempenhando um papel central na arquitetura do Kafka.
- **Consumidores:** Aplicativos que se inscrevem nos tópicos para receber e processar mensagens.
- **ZK (substituído por KRaft em 2023):** O Kafka originalmente dependia do Apache ZK para gerenciamento de metadados e coordenação. Em 2023, no entanto, o ZK foi substituído pelo *KRaft*, visando um gerenciamento mais eficiente e robusto.

Essa arquitetura permite ao Kafka escalar, tolerar falhas e distribuir mensagens entre várias máquinas e servidores, tornando-o uma escolha popular em empresas como LinkedIn, X (Twitter), Uber e Foursquare para a construção de pipelines de dados em tempo real e sistemas de processamento de eventos (KAFKA, 2023). A combinação de sua arquitetura robusta e capacidade de lidar com grandes volumes de dados em tempo real faz do *Kafka* um excelente candidato para estudos aprofundados.

#### 4.1.2 ZooKeeper (ZK)

O ZK (Apache Software Foundation, 2008) é um serviço de coordenação distribuída amplamente utilizado em sistemas de grande escala na Internet. Ele oferece um ambiente confiável e altamente disponível para a coordenação de tarefas entre diversos nós em um cluster distribuído. De acordo com Hunt *et al.* (2010), o ZK é baseado em um conjunto de servidores distribuídos que formam o chamado “ensemble”. Cada servidor nesse conjunto mantém uma cópia completa dos dados de configuração e estado do sistema. Os clientes podem se conectar a qualquer servidor do *ensemble* para ler ou gravar informações. Quando uma gravação é realizada, os dados são replicados para a maioria dos servidores antes de serem confirmados, garantindo a consistência das informações.

O ZK atua como um serviço centralizado de coordenação para SDs, utilizando um conjunto de servidores para assegurar a consistência dos dados através do *Two-Phase Commit Protocol*. Esse protocolo opera em duas fases: na primeira, o coordenador propõe a execução de uma transação aos participantes, que respondem indicando se estão prontos para prosseguir; na segunda, se todos os participantes estiverem de acordo, o coordenador emite o comando de *commit* para a conclusão da transação, ou, caso contrário, o comando de *rollback*, revertendo as alterações.

Além disso, o ZK implementa o algoritmo *Zookeeper Atomic Broadcast Protocol* (ZAB), que garante a consistência dos dados distribuídos utilizando o *Two-Phase Commit Protocol* para replicar todas as transações em todos os nós do cluster (VINKA, 2018). O ZK também oferece um sistema de coordenação sem espera (“*wait-free*”), permitindo que os clientes armazenem dados em *cache* sem a necessidade de gerenciar diretamente esse *cache*.

O ZK é amplamente empregado na coordenação e gerenciamento de serviços distribuídos, proporcionando um serviço de consenso altamente confiável para SDs. Sua capacidade de garantir a consistência e a sincronização entre os nós permite a implementação de serviços distribuídos confiáveis e escaláveis, o que torna o ZK uma escolha valiosa para este estudo.

#### 4.1.3 *CorfuDB*

O *CorfuDB* é uma plataforma que se baseia na ideia de um *log* compartilhado para garantir consistência em SDs. Ao contrário das abordagens tradicionais, que muitas vezes optam por particionar dados para alcançar escalabilidade em detrimento da consistência global, o *CorfuDB* oferece uma solução que mantém a consistência e a atomicidade das operações em larga escala. A visão do *CorfuDB* é especialmente relevante para aplicações distribuídas que precisam de um equilíbrio entre escalabilidade e consistência, oferecendo uma abordagem que permite a construção de plataformas de dados e controle com garantias de Atomicidade, Consistência, Isolamento e Durabilidade (ACID) em grande escala.

#### 4.1.4 *ScaleCube*

O *ScaleCube* é uma biblioteca projetada para microserviços, focada em oferecer alto desempenho e baixa latência em sistemas escaláveis e reativos. Sua arquitetura modular e distribuída facilita a integração com *gateways* de API, descoberta de serviços e balanceamento de carga. Utilizando o protocolo *SWIM* para gerenciamento eficiente de *clusters*, o *ScaleCube* suporta uma variedade de módulos de comunicação intercambiáveis, permitindo implantações flexíveis e adaptáveis às necessidades específicas de SDs.

Uma característica distintiva do *ScaleCube* é sua abordagem baseada em *streams* reativos, que permite uma comunicação assíncrona entre microserviços. Isso melhora a capacidade de resposta e escalabilidade do sistema, permitindo que os serviços processem e reajam aos dados em tempo real, sem bloqueios. O *ScaleCube* é otimizado para ambientes de processamento de dados em tempo real e SDs de grande escala, onde a minimização da latência é crucial.

Além disso, a arquitetura *mesh* completa do *ScaleCube* elimina pontos únicos de falha e gargalos, promovendo uma alta disponibilidade e resiliência. Suas características incluem descoberta automática de serviços sem configuração adicional, detecção de falhas, e suporte para diversos protocolos de transporte e codificação. O *ScaleCube* também oferece suporte para um modelo de programação assíncrono e não bloqueante, adequado para aplicações que exigem alta eficiência e escalabilidade em ambientes dinâmicos.

#### 4.1.5 Moleculer for Java

O *Moleculer for Java* implementa o *Moleculer Microservices Framework* para a *Java Virtual Machine* (JVM), projetado para facilitar o desenvolvimento de aplicações distribuídas não bloqueantes. Esse *framework* é altamente relevante para o estudo de SDs, pois oferece diversas funcionalidades essenciais, como APIs de mensagens e eventos não bloqueantes, serviços *REST* assíncronos, e tolerância a falhas com recursos como *circuit breaker* e *retry*. A estrutura do *Moleculer for Java* inclui transporte e serialização plugáveis, permitindo a comunicação eficiente entre serviços em ambientes distribuídos por meio de diferentes protocolos, como *Transmission Control Protocol* (TCP), *Message Queuing Telemetry Transport* (MQTT), *Kafka*, e *Redis*. Além disso, o *framework* suporta balanceamento de carga, *cache* integrado, e descoberta dinâmica de serviços, características que são fundamentais para a construção de SDs escaláveis e resilientes. A capacidade do projeto de integrar-se com outros serviços, através de *WebSockets*, torna-o uma ferramenta robusta para o desenvolvimento de aplicações modernas que exigem alta disponibilidade e desempenho em ambientes distribuídos.

#### 4.1.6 Conclusões da seleção de repositórios

Os dois repositórios selecionados manualmente têm muito a agregar em termos de implementação e evolução, tanto para a área de SDs quanto para a ES. Assim, a escolha deles propicia a análise da evolução do código em contextos práticos e desafiadores, contribuindo para uma compreensão mais abrangente das práticas de desenvolvimento em SDs.

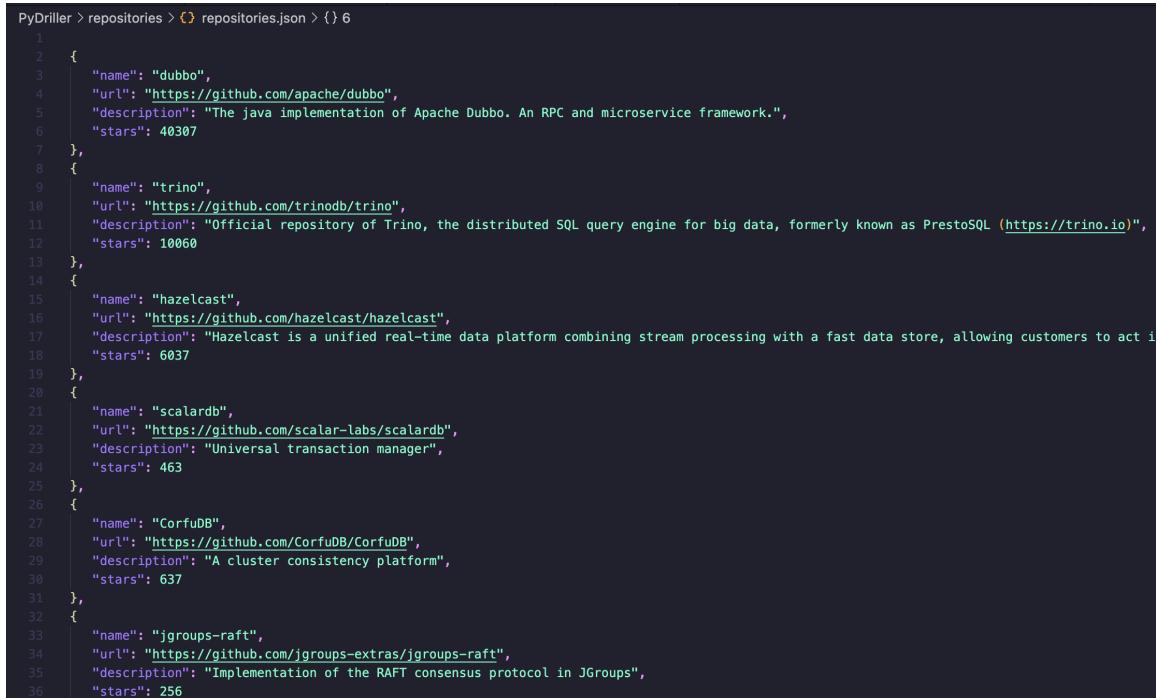
Da mesma forma, os repositórios adicionais analisados relacionam-se diretamente com a área de SDs. A escolha desses repositórios permite uma análise detalhada da evolução do código em contextos práticos e desafiadores de SDs. Essa abordagem contribui para uma compreensão mais completa das técnicas e ferramentas utilizadas no desenvolvimento de aplicações escaláveis e resilientes, enriquecendo o estudo das práticas de ES em cenários complexos e em constante evolução.

## 4.2 Ferramenta *Worked-Example-Miner* (WEM)

A aplicação do método proposto na Seção 3.3 foi apoiada por meio da ferramenta WEM (SILVA, 2023c), que integra as ferramentas citadas na Seção 3.2.2. Dentro da ferramenta, encontram-se algoritmos que representam diferentes fases de extração de métricas, geração de dados e metadados, estando as fases divididas nos diretório “*PyDriller*”, “*RefactoringMiner*” e “*Gemini*”, consecutivamente, sendo que todas foram feitas usando a linguagem de programação *Python* (ROSSUM, 1991).

A primeira etapa, que já foi concluída e discutida na Seção 4.1, consiste na execução do algoritmo *PyDriller/repositories\_picker.py*. Dentre os repositórios encontrados pelo *script*, ele seleciona, de forma aleatória - a fim de evitar qualquer tipo de viés e garantir a imparcialidade no processo - três candidatos para análise. A Figura 4 mostra parte do arquivo gerado no formato *json*, contendo todos os repositórios que atendem aos critérios de seleção e a Figura 5 apresenta os candidatos escolhidos.

**Figura 4 – Arquivo *json* gerado pelo *repositories\_picker.py***



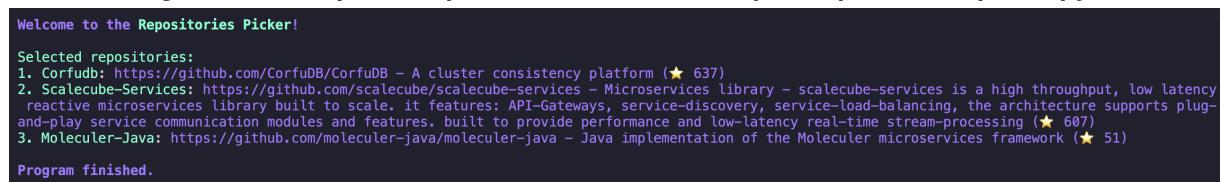
```

PyDriller > repositories > repositories.json > {} 6
1
2 {
3     "name": "dubbo",
4     "url": "https://github.com/apache/dubbo",
5     "description": "The java implementation of Apache Dubbo. An RPC and microservice framework.",
6     "stars": 40307
7 },
8 {
9     "name": "trino",
10    "url": "https://github.com/trinodb/trino",
11    "description": "Official repository of Trino, the distributed SQL query engine for big data, formerly known as PrestoSQL (https://trino.io)",
12    "stars": 10060
13 },
14 {
15     "name": "hazelcast",
16     "url": "https://github.com/hazelcast/hazelcast",
17     "description": "Hazelcast is a unified real-time data platform combining stream processing with a fast data store, allowing customers to act i",
18     "stars": 6037
19 },
20 {
21     "name": "scalardb",
22     "url": "https://github.com/scalar-labs/scalardb",
23     "description": "Universal transaction manager",
24     "stars": 463
25 },
26 {
27     "name": "CorfuDB",
28     "url": "https://github.com/CorfuDB/CorfuDB",
29     "description": "A cluster consistency platform",
30     "stars": 637
31 },
32 {
33     "name": "jgroups-raft",
34     "url": "https://github.com/jgroups-extras/jgroups-raft",
35     "description": "Implementation of the RAFT consensus protocol in JGroups",
36     "stars": 256

```

Fonte: Autoria própria.

**Figura 5 – Exemplo de repositórios selecionados pelo *repositories\_picker.py***



```

Welcome to the Repositories Picker!
Selected repositories:
1. Corfudb: https://github.com/CorfuDB/CorfuDB - A cluster consistency platform (★ 637)
2. Scalecube-Services: https://github.com/scalecube/scalecube-services - Microservices library - scalecube-services is a high throughput, low latency
   reactive microservices library built to scale. it features: API-Gateways, service-discovery, service-load-balancing, the architecture supports plug-
   and-play service communication modules and features. built to provide performance and low-latency real-time stream-processing (★ 607)
3. Moleculer-Java: https://github.com/moleculer-java/moleculer-java - Java implementation of the Moleculer microservices framework (★ 51)

Program finished.

```

Fonte: Autoria própria.

A seguir, são apresentados os três repositórios selecionados, de forma aleatória, pelo *script repositories\_picker.py* que estão expostos na Figura 5.

A segunda etapa envolve a geração dos arquivos com as métricas CK para todas as classes e métodos em cada *commit* de um repositório específico. O algoritmo responsável por essa função encontra-se em *PyDriller/code\_metrics.py* (SILVA, 2023a). Durante esse processo, a ferramenta percorre toda a árvore de *commits* do repositório selecionado. Para cada *commit*, o CK é executado em sua versão que inclui a métrica *MethodInvocationCounter* desenvolvida

neste projeto, permitindo a extração de termos dos métodos invocados em cada classe. Isso resulta na geração das métricas CK para cada estado do repositório ao final da execução.

Adicionalmente, a ferramenta produz um arquivo no formato *diff* para cada refatoração realizada no repositório, possibilitando a investigação das refatorações realizadas no código-fonte quando forem identificadas uma tendência de melhoria das métricas. Não suficiente, considerando que este processo é oneroso e prolongado, um arquivo de *log* é criado para documentar os estados já processados. Isso permite que a geração dos arquivos CSV do CK seja retomada do ponto de interrupção anterior, seja ela causada por um comando de parada de execução do programa ou por uma falha de energia.

A implementação da métrica *MethodInvocationCounter* foi realizada com o objetivo de fornecer uma análise detalhada dos padrões de invocação de métodos em classes Java. Essa métrica, introduzida como uma *ClassLevelMetric* no CK, capta a frequência com que cada método é invocado dentro de uma classe, oferecendo uma visão detalhada das dependências e da utilização dos métodos. Para isso, várias modificações foram feitas no código-fonte da ferramenta CK (SILVA, 2024b), incluindo a adição de um novo campo privado `methodInvocations` na classe `CKClassResult.java`, responsável por armazenar os dados formatados das invocações. Além disso, a classe utilitária `MethodCounter.java` foi introduzida para auxiliar no armazenamento e formatação desses dados, garantindo consistência e reutilização na implementação da métrica. A classe `MethodInvocationCounter.java` foi desenvolvida para visitar declarações e invocações de métodos, populando a estrutura de dados `MethodInfo` com as informações coletadas, que posteriormente são formatadas e associadas ao resultado final através do método `setResult`. Por fim, o arquivo `ResultWriter.java` foi adaptado para incluir esses dados na saída gerada pelo CK, proporcionando uma visão sintetizada das interações entre os métodos. Essa métrica é útil para identificar métodos-chave, oportunidades de refatoração, otimização de desempenho e para a documentação do sistema.

A terceira etapa é a geração de metadados conforme a análise dos arquivos CSV gerados na etapa anterior. O algoritmo responsável por essa função encontra-se em *PyDriller/metrics\_changes.py* (SILVA, 2023b), o qual depende da execução da segunda etapa para operar. Essencialmente, esta ferramenta lê o valor das métricas de todos os *commits* do repositório processado, produzindo os seguintes resultados:

- **Evolução das métricas:** Geração de arquivos que sintetizam os dados, organizados por classes e métodos, contendo o histórico completo de alterações e as métricas correspondentes. A Figura 14 ilustra o resultado gerado.
- **Predição das métricas:** Geração de gráficos que segmentam as métricas por classe e método. Esses gráficos mostram, por meio de pontos, os valores de cada métrica em diferentes estados do código. Além disso, uma linha de regressão linear é aplicada para ilustrar a tendência dessas métricas ao longo do tempo, permitindo observar se o

comportamento da classe indica uma melhoria ou não. A Figura 11 ilustra o resultado gerado.

- **Padrões de refatoração:** Listagem e filtragem, conforme o interesse, dos padrões de refatoração identificados em cada arquivo associado às alterações correspondentes a cada *commit* do repositório.

A saída dos algoritmos que utilizam o *RefactoringMiner* consiste em um objeto *JSON*, semelhante à Figura 6, que permite a análise das refatorações por tipo. Destaca-se que, a partir do *JSON* geral, é possível aplicar filtros para armazenar apenas refatorações dos tipos desejados. Essa abordagem proporciona uma visão detalhada e personalizável das transformações efetuadas no código-fonte.

- **Estatísticas das métricas:** Geração de dois arquivos, um para classes e outro para métodos, organizados de forma decrescente de acordo com o número de alterações feitas em cada classe ou método. Esses arquivos contêm estatísticas sobre cada métrica, incluindo o valor mínimo, máximo, mediana, média e terceiro quartil. A Figura 15 ilustra o resultado gerado.
- **Alterações substanciais nas métricas:** Identificação de padrões de declínio na métrica selecionada, como o CBO. Esse processo oferece uma análise detalhada das classes ou métodos que mostram tendências de melhoria ao longo do tempo, classificando-os com base no percentual de redução do CBO entre diferentes estados de cada classe ou método. Os trechos de código que tiveram queda na métrica CBO e onde foram detectados padrões de refatoração fazem parte do conjunto de candidatos armazenados no arquivo *PyDriller/{repository\_name}/substantial\_{metric\_name}\_{class\_or\_method}\_changes.csv*. A Figura 7 ilustra o resultado gerado. Vale destacar que diversos campos foram removidos ou reduzidos para facilitar a visualização nesta imagem.

Nessa funcionalidade é possível configurar um *threshold* (limiar) para definir o que é uma queda substancial, por exemplo, 0.10, que significa que só entra para a lista de candidatos aqueles com uma queda igual ou superior a 10% entre *commits* distintos. Todavia, após iterações de ajuste na heurística, esse limiar foi colocado como 0, pois um valor percentual só é viável conforme o tamanho da classe.

A quarta e última etapa realizada pela ferramenta, descrita no arquivo *Gemini/gemini.py* (SILVA, 2024a), utiliza semáforos para limitar a no máximo três requisições simultâneas, a fim de evitar bloqueios de endereço *Internet Protocol* (IP) pela API do *Gemini*. Esta consiste em fornecer à API do *Gemini* um contexto baseado em currículos de SD, conforme as diretrizes estabelecidas pelo NSF/IEEE-TCPP (NSF/IEEE-TCPP, 2024). O *Gemini* recebe esse currículo juntamente com a lista de candidatos que apresentaram redução nos valores de CBO e onde

**Figura 6 – Refatorações da classe org.apache.zookeeper.server.quorum.Learner**

```
{
  "type": "Pull Up Method",
  "description": "Pull Up Method public getPendingRevalidationsCount() : int from\n  \"leftSideLocations\": [\n    {\n      \"filePath\": \"src/java/main/org/apache/zookeeper/server/quorum/Follower.java\", \n      \"startLine\": 332,\n      \"endLine\": 334,\n      \"startColumn\": 5,\n      \"endColumn\": 6,\n      \"codeElementType\": \"METHOD_DECLARATION\", \n      \"description\": \"original method declaration\", \n      \"codeElement\": \"public getPendingRevalidationsCount() : int\"\n    }\n  ],\n  \"rightSideLocations\": [\n    {\n      \"filePath\": \"src/java/main/org/apache/zookeeper/server/quorum/Learner.java\", \n      \"startLine\": 78,\n      \"endLine\": 80,\n      \"startColumn\": 5,\n      \"endColumn\": 6,\n      \"codeElementType\": \"METHOD_DECLARATION\", \n      \"description\": \"pulled up method declaration\", \n      \"codeElement\": \"public getPendingRevalidationsCount() : int\"\n    }\n  ]\n}
```

Fonte: Autoria própria.

**Figura 7 – Lista de candidatos com quedas substanciais e padrões de refatorações identificados ZK**

Class	Type	From CBO	To CBO	% Variation	Method Invocations	Refactoring Patterns
org.apache.zookeeper.ZooKeeperMain\$MyCommandOptions	innerclass	1.0	0.0	100.0	getCmdArgument[ get():1 ]	ZooKeeperMain.java: [Extract Method(34) Replace V](1)
org.apache.zookeeper.server.quorum.flexible.QuorumVerifier	interface	2.0	1.0	50.0	containsQuorum[ ]	QuorumVerifier.java: [Change Return Type(4) Rename I](1)
org.apache.zookeeper.server.quorum.Follower	class	21.0	13.0	38.1	readPacket[ logQuorumPacket():1 ]	FollowerSessionTracker.java: [Rename Class(1)] se
org.apache.zookeeper.server.SyncRequestProcessor	class	11.0	7.0	36.4	SyncRequestProcessor[ allocateDirect():1 ]	SyncRequestProcessor.java: [Change Attribute Access(1)]
org.apache.zookeeper.server.NIOServerCnxn\$ConsCommand	innerclass	3.0	2.0	33.3	ConsCommand[ ]	NIOServerCnxn.java: [Extract Method(144) Change A](1)

Fonte: Autoria própria.

foram identificados padrões de refatoração relativos aos arquivos de *commits* selecionados. Contudo, somente as colunas contendo os nomes das classes são enviadas, acompanhadas pelos termos extraídos e armazenados na coluna da métrica *MethodInvocationCounter*. O objetivo dessa integração é relacionar os candidatos com sua relevância em SD, considerando que eles já demonstraram comportamentos desejados do ponto de vista da ES, de acordo com as métricas e padrões de refatoração.

De forma simplificada, o algoritmo *PyDriller/code\_metrics.py* extrai as métricas CK. Em seguida, o algoritmo *PyDriller/metrics\_changes.py* gera dados e metadados que facilitam a análise do código, reduzindo significativamente o escopo de busca por trechos de código relevantes sob a perspectiva educacional de SD. Por último, o algoritmo *Gemini/gemini.py* recebe a lista de códigos com quedas substanciais na métrica CBO e os padrões de refatoração identificados. Ele então analisa os termos dos métodos invocados desses códigos e busca correlacioná-los com tópicos educacionais de SD. A ferramenta se limita a essas funcionalidades, uma vez que a

**Tabela 1 – Características dos Repositórios: Quantidade de Classes, Linhas de Código (LOC), Número de *Commits* e Tempo de Execução**

Nome do Re却itório	Número de Classes	Linhas de Código (LOC)	Número de Commits	Tempo de Execução
Corfudb	1,397	82,126	3,563	25h 47min
Kafka	29	1,303	13,688	5h 40min
Molecular-Java	309	24,597	607	43min
Scalcube-Services	279	9,501	4,702	2h 55min
Zookeeper	987	66,203	2,569	6h 55min

**Fonte:** Autoria própria.

validação da relevância educacional dos candidatos gerados pelo *Gemini* é realizada por especialistas. Dessa forma, a ferramenta não gera exemplos trabalhados, mas sim uma lista pequena de candidatos.

A ferramenta automatiza todo o processo de extração e geração de dados e metadados, entretanto não automatiza todo o processo de análise de candidatos devido as questões subjetivas, como a análise de um especialista. Todavia, a redução de escopo no processo de análise de código representa bons resultados. O processo envolve a diminuição significativa da quantidade de casos a serem analisados, simplificando a seleção de exemplos relevantes para a análise. Portanto, a ferramenta reflete a heurística desenvolvida, a qual pode ser observada na Figura 2.

Os dados apresentados na Tabela 1 mostram uma variabilidade significativa entre os re却itórios analisados. O re却itório *Kafka*, por exemplo, possui um número relativamente baixo de classes (29) e linhas de código (LOC) (1.303), mas um número elevado de *commits* (13.688) e um tempo de execução considerável, de aproximadamente 340 minutos. Em contraste, o re却itório *Corfudb*, com um número substancial de classes e LOC (1.397 e 82.126, respectivamente), apresenta o maior tempo de execução (25 horas e 47 minutos), refletindo a complexidade e o tamanho do código analisado. O re却itório *Molecular-Java*, que possui um número moderado de classes (309) e LOC (24.597), apresenta um tempo de execução menor (43,39 minutos), indicando uma análise mais rápida. Já o *Scalcube-Services* e o *Zookeeper* demonstram uma relação proporcional entre o número de classes, linhas de código e o tempo de execução, com o *Scalcube-Services* exibindo um tempo de execução intermediário e o *Zookeeper* um tempo mais elevado devido à sua maior complexidade. Esses dados evidenciam a relação entre a quantidade de código e o tempo necessário para a execução das ferramentas utilizadas no processo de extração de métricas e análise de mudanças, refletindo como a complexidade e o volume de código influenciam a duração da análise.

#### 4.3 Análises Preliminares dos Projetos Selecionados

Nesta seção são apresentados casos analisados antes da definição completa da heurística, bem como casos que se ajustam à heurística estabelecida. A análise concentrou-se

majoritariamente no ZK, uma vez que este repositório foi utilizado durante todas as iterações de refinamento da heurística. Após a definição precisa da heurística, foi realizada uma última análise no ZK antes de expandir a investigação para os outros repositórios selecionados, com o objetivo de validar a heurística e evitar o viés associado a um único repositório.

Como a heurística ainda não estava completamente definida, a análise focou na identificação de termos-chave relevantes para SD. Todos os termos foram sugeridos por um especialista na área e incluíram: “*algorithm*”, “*security*” (ou “*CVE*”), “*improvements*”, “*consensus*”, “*election*” e “*log*”. Desde o início das análises no ZK, observou-se, por meio de regressões lineares, uma tendência de queda em métricas como o CBO em diversos *commits* do histórico do repositório do ZK. Entretanto, verificou-se que apenas uma pequena parcela desses *commits* continha mensagens relacionadas a aspectos como substituição de algoritmos, consistência e replicação, comunicação e processos, e desempenho. Estes aspectos são fundamentais para refatorações que explicam conceitos de SDs e que podem proporcionar melhorias significativas no *design* do código.

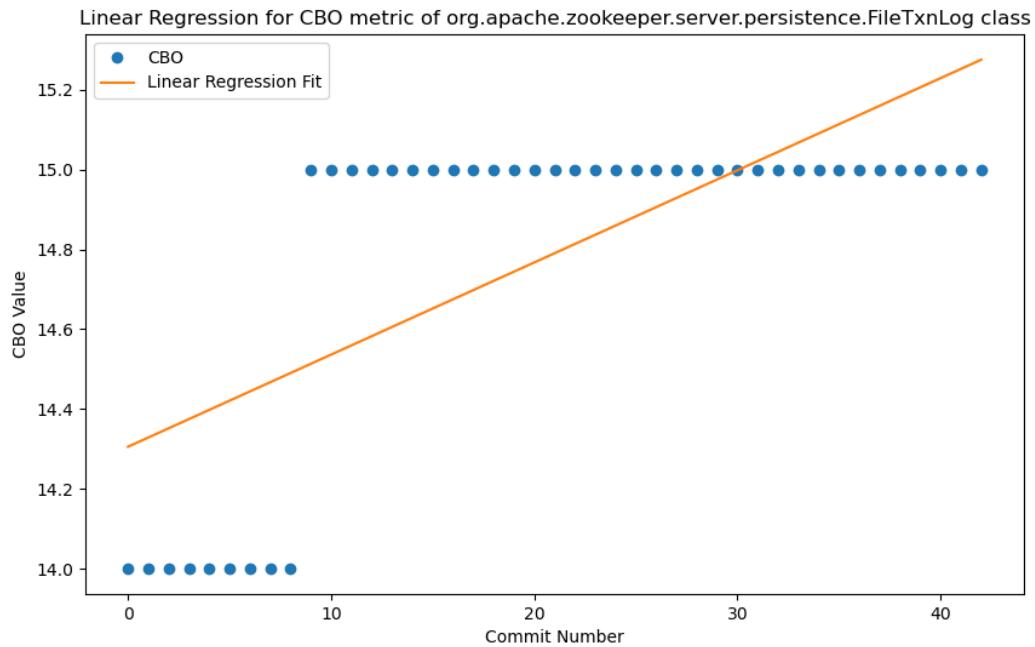
A análise do *commit* 83cf0a93c37759334fab885c2010fa0b7d953f52, cuja mensagem é “*ZOOKEEPER-308. improve the atomic broadcast performance 3x*”, foi aprofundada. Apesar da sugestão inicial de um ganho de desempenho de três vezes, uma investigação mais detalhada revelou que a alteração se limitava à substituição da classe *FileOutputStream* por *BufferedOutputStream*. Embora essa mudança tenha de fato resultado em uma melhoria de desempenho, ela não constitui uma refatoração significativa. Não suficiente, visto que o *commit* modificou 19 arquivos, sendo relevante apenas a alteração no arquivo *FileTxnLog.java*, observa-se muitas refatorações não relacionadas. No entanto, o nome da classe indica que está relacionada a um sistema de *log*, um tópico de interesse no contexto de SD. No que diz respeito às métricas, conforme ilustrado nas Figuras 8, 9, e 10, todas apresentaram incrementos, o que não é desejável do ponto de vista da ES.

Os outros arquivos também foram alterados no *commit* 83cf0a93c37759334fab885c2010fa0b7d953f52 e abordam refatorações relacionadas à validação de objetos nulos antes de operações, mas estão fora do escopo da mensagem do *commit*. Apesar disso, a melhoria de desempenho é reconhecida como válida, embora sua magnitude não possa ser verificada, mas esse caso não se enquadra na heurística, afinal não houve queda ou padrões de refatoração detectados que sejam interessantes.

A falta de uma refatoração mais substancial e a existência de uma melhoria de desempenho que não pode ser validada tornam este *commit* inadequado para servir como objeto de exemplo trabalhado. Além disso, uma vez que a classe “*BufferedOutputStream*” está presente desde a versão inicial do Java (lançada em 1996) e o *commit* em questão é de 2009, ou seja, o uso de um *buffered input* devia ter sido feita desde o começo do projeto.

A análise realizada nos parágrafos anteriores indica que, em certas situações, melhorias no código relacionadas à implementação mais abrangente de um conceito podem não resultar na redução das métricas. Entretanto, é necessário diferenciar entre melhorias que visam aumen-

**Figura 8 – CBO da classe *org.apache.zookeeper.server.persistence.FileTxnLog*.**



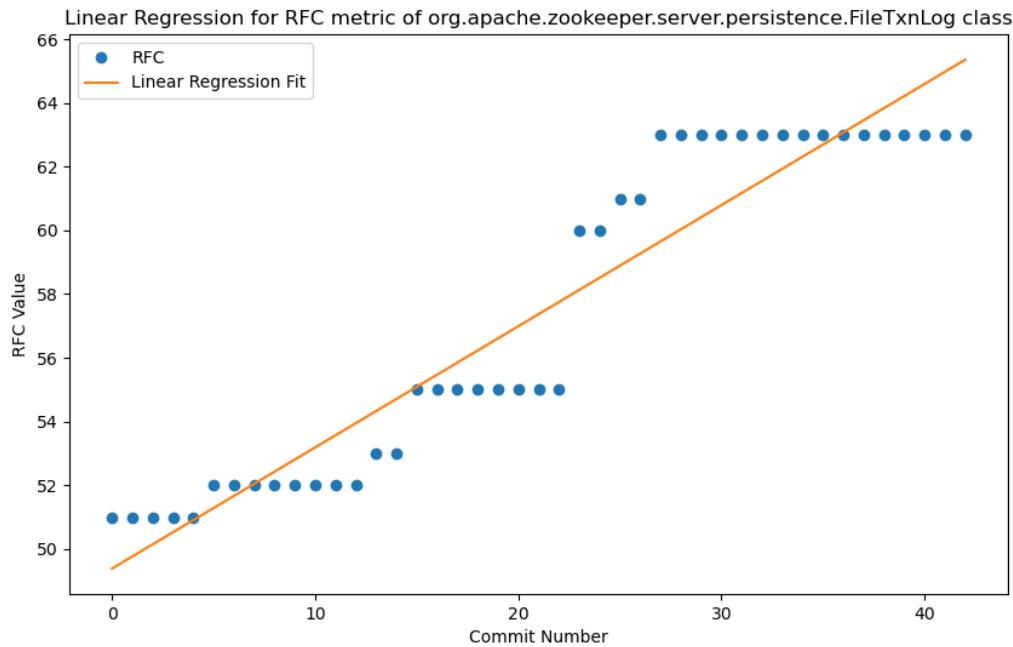
**Fonte:** Autoria própria.

tar a completude e a eficiência e aquelas que, de fato, elevam a qualidade do *design* do código. Refatorações de completude bem executadas podem simplesmente introduzir uma nova funcionalidade ou modificar um método sem alterar o comportamento final do sistema. Em alguns casos, essas alterações podem não ocasionar uma diminuição em métricas como o CBO, ainda que tal redução seja desejável para garantir a relevância do trecho de código na ES.

Embora a mudança promova efetivamente uma melhoria de desempenho, as métricas tradicionais de código, como CBO, RFC e WMC não podem refletir diretamente a magnitude do ganho de desempenho obtida. Isto porque estas são métricas estáticas de código, conforme discutido na Seção 3.3. Entretanto, apesar da magnitude do aprimoramento não poder ser totalmente verificada por meio das métricas convencionais, é válida a melhoria de desempenho proporcionada pela otimização do uso da classe de E/S, mas apresenta-se como um caso irrelevante para a criação de um exemplo trabalhado.

Lamentavelmente, apenas um *commit* apresentou uma correlação direta entre as vulnerabilidades identificadas no banco de dados do “*CVE-Details*” e os *commits* no repositório do ZK. O único caso encontrado foi para o *commit* 9213f7353b1e6ce4d0fdb1dca963ace1fd32cec, o qual se relaciona com a vulnerabilidade *CVE-2021-21295*. Entretanto, por mais que o *commit* resolva um problema de vulnerabilidade, constatamos que esta refatoração específica consiste apenas na atualização da versão da biblioteca *Netty* no arquivo *pom.xml* do Java. Essa biblioteca provê uma estrutura cliente-servidor de E/S não bloqueante, logo ela não se mostra uma

**Figura 9 – RFC da classe *org.apache.zookeeper.server.persistence.FileTxnLog*.**



**Fonte:** Autoria própria.

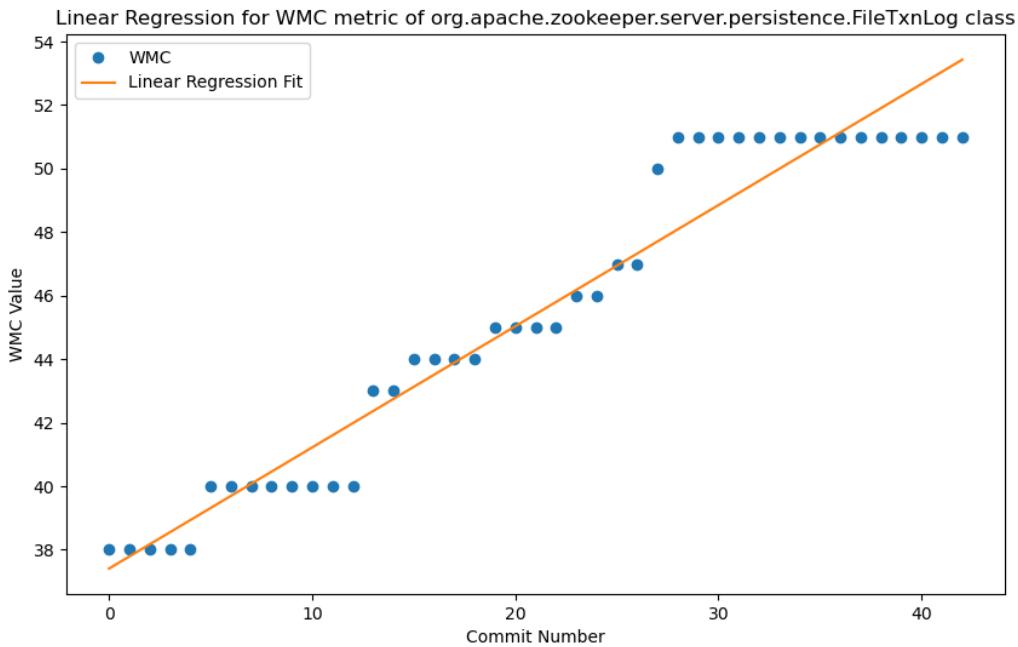
boa candidata para correlacionar métricas de código com a segurança, pois se trata de uma ação direta para resolver uma vulnerabilidade conhecida.

A partir deste ponto, conclui-se a análise do subconjunto de *commits* que contêm palavras-chave pertinentes a SDs na mensagem do *commit*. Posteriormente, a investigação é ampliada para identificar palavras-chave presentes no nome das classes e métodos do ZK, tais como “*election*”, “*quorum*”, “*replication*”, “*consensus*” e “*log*”, também sugeridas por um especialista em SD. Esses termos são indicativos de tópicos relevantes do âmbito educacional de SDs sendo, portanto, essenciais para a compreensão e análise dessa área.

Dentro deste novo conjunto, identifica-se novamente a classe *org.apache.zookeeper.-server.persistence.FileTxnLog*, associada ao *commit* 83cf0a93c37759334fab885c20-10fa0b7d953f52, que indicava um ganho de desempenho de três vezes. Esse candidato é repetido, pois está relacionado à palavra-chave *log*. No entanto, mais uma vez, esse caso não se revelou interessante para ilustrar qualquer conceito de SDs, nem apresentou melhorias nas métricas que o tornassem uma refatoração relevante para a ES.

Para entender melhor o comportamento das métricas em classes específicas, a seleção da classe *org.apache.zookeeper.server.quorum.Follower* foi feita com base na sua nomenclatura, que sugere que a classe está relacionada ao aspecto do consenso no protocolo. Em particular, a classe lida com o processo pelo qual os participantes (*followers*) alcançam consenso durante uma votação. Com essa premissa, realizou-se uma análise detalhada da classe e observou-se uma tendência de queda nas métricas CBO, RFC e WMC.

**Figura 10 – WMC da classe *org.apache.zookeeper.server.persistence.FileTxnLog*.**



**Fonte:** Autoria própria.

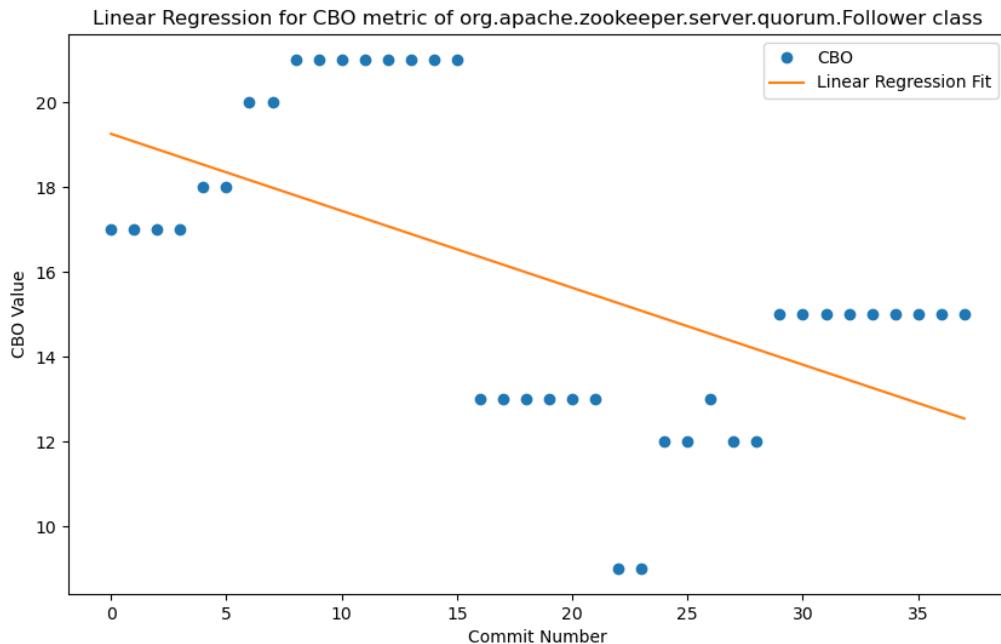
As Figuras 11, 12 e 13 ilustram as saídas geradas pelo algoritmo *PyDriller/metrics\_changes.py* para a classe previamente mencionada, *org.apache.zookeeper.server.quorum.Follower*. Essas três figuras proporcionam uma visão detalhada do comportamento evolutivo desta classe em relação às métricas CBO, RFC e WMC, respectivamente.

A partir das Figuras 11, 12 e 13 observa-se uma tendência decrescente nos valores das métricas CBO, RFC e WMC. Enquanto isso, a Figura 14 delineia a evolução das métricas de tal classe e a Figura 15 apresenta um vislumbre das cinco primeiras linhas dos CSVs gerados, delineando a evolução das métricas de tal classe e as estatísticas gerais das classes no contexto do ZK.

As métricas afetadas e os seus impactos são descritos a seguir.

- **CBO:** Conforme a Figura 11, a métrica CBO reduziu de 21 para 13, indicando um desacoplamento efetivo da classe *Follower* de outras classes com as quais previamente interagia. A introdução de uma classe base, *Learner*, ajudou a centralizar a comunicação e reduzir o acoplamento.
- **RFC:** Conforme a Figura 12, a redução de 79 para 35 no RFC reforça que a quantidade de métodos que podem ser invocados pela classe *Follower* foi consideravelmente diminuída, indicando uma melhor definição de responsabilidades dentro da hierarquia de classes.

**Figura 11 – Regressão linear da métrica CBO da classe *org.apache.zookeeper.server.quorum.Follower* do ZK**



**Fonte:** Autoria própria.

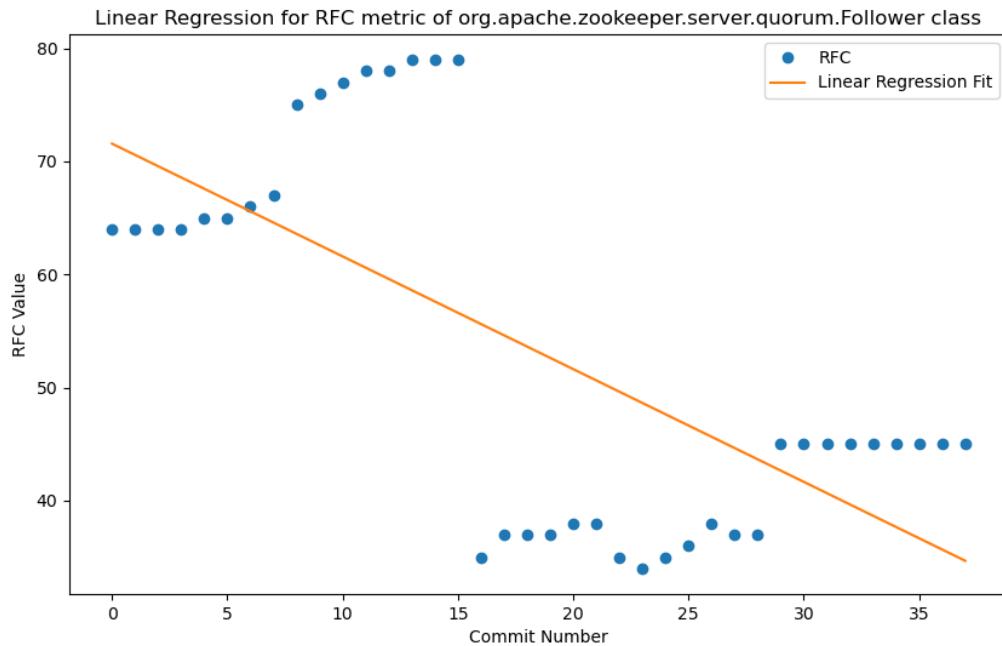
- **WMC:** Conforme a Figura 13, a redução de 45 para 18 no WMC sugere uma simplificação considerável na complexidade interna da classe *Follower*, através da realocação de parte da lógica para a nova classe *Learner*.

O comportamento de redução das métricas CBO, RFC e WMC apontado pelas Figuras 11, 12 e 13 sugere uma melhoria no *design* do código, acompanhada por diversos padrões de refatoração. Destacam-se os padrões “*Extract Superclass*” e “*Pull-Up Method*”, ocorrendo 2 e 4 vezes, respectivamente, na 16<sup>a</sup> alteração da classe, associada ao *commit* 0957b8404e1ecfc5703d7c2827752773b7dc23be. Embora essa refatoração seja muito promissora para a ES, ela é extremamente extensa e complexa.

Após avaliação por um especialista, constatou-se que se trata de uma mudança na hierarquia e na relação entre *peers* e *followers*. Apesar de estar diretamente relacionada ao currículo de SD, a refatoração foi considerada pelo especialista como inadequada para explicar conceitos de SD, além de estar em um nível de complexidade superior ao esperado para um aluno de graduação.

Do ponto de vista da ES, as refatorações realizadas na classe *org.apache.zookeeper.-server.quorum.Follower* são de grande relevância, pois abordam conceitos fundamentais de encapsulamento e organização do código, que são essenciais para a melhoria contínua e a manutenção da qualidade do software. O encapsulamento foi aprimorado com a transferência de parte da lógica para a classe *org.apache.zookeeper.server.quorum.Learner*.

**Figura 12 – Regressão linear da métrica RFC da classe *org.apache.zookeeper.server.quorum.Follower* do ZK**



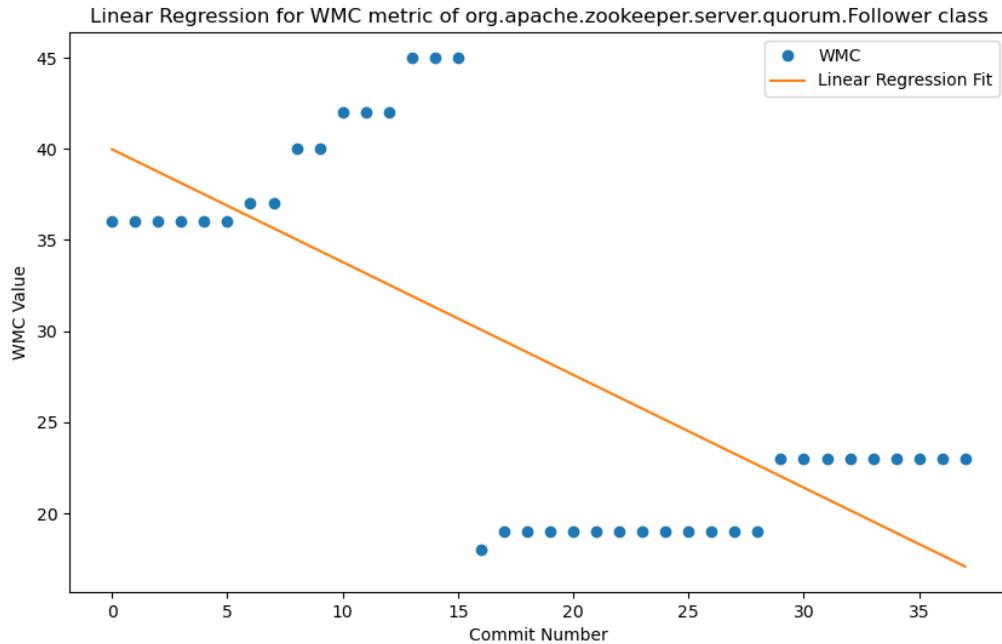
**Fonte:** Autoria própria.

Os impactos dessa refatoração podem também ser relacionados aos princípios SOLID, que buscam promover um *design de software* mais manutenível, escalável e compreensível:

- **Manutenibilidade:** A melhoria na modularidade e a redução do acoplamento, refletidas na diminuição da métrica CBO, facilitam a manutenção e a futura expansão do código. Esses benefícios estão diretamente ligados ao Princípio da Responsabilidade Única (*Single Responsibility Principle* (SRP)) e ao Princípio Aberto-Fechado (*Open-Closed Principle* (OCP)). O SRP é evidenciado pela separação de responsabilidades excessivas da classe *Follower*, enquanto o OCP é aplicado ao permitir que novas funcionalidades sejam adicionadas com menor impacto nas classes existentes, utilizando herança e polimorfismo com a introdução da classe *Learner*.
- **Reusabilidade:** A potencial movimentação de lógica comum para a superclasse *Learner* não só aumenta a reusabilidade do código entre as subclasses, mas também se alinha ao Princípio da Segregação de Interfaces (*Interface Segregation Principle* (ISP)). Isso ocorre porque as interfaces ou classes abstratas podem ser projetadas para agrupar métodos relevantes para seus respectivos utilizadores, evitando que subclasses dependam de interfaces que não utilizam.

Avançando, outro candidato em potencial era justamente o resultado da operação “Extract Superclass” que ocorreu na classe *org.apache.zookeeper.server.quorum.Follower*, a

**Figura 13 – Regressão linear da métrica WMC da classe *org.apache.zookeeper.server.quorum.Follower* do ZK**



**Fonte:** Autoria própria.

**Figura 14 – Evolução das métricas da classe *org.apache.zookeeper.server.quorum.Follower* do ZK**

Class	Commit Hash	CBO	WMC	RFC
org.apache.zookeeper.server.quorum.Follower	175-cfd2ecb8d049c1b51b2c1e6d5fd02edd5d3c26e8	17.0	36.0	64.0
org.apache.zookeeper.server.quorum.Follower	176-b4b73a37d43413efdf50427f96aab2720af1baf	17.0	36.0	64.0
org.apache.zookeeper.server.quorum.Follower	177-0a81c16c4c921ef52179b943dd046fc72022ccc6	17.0	36.0	64.0
org.apache.zookeeper.server.quorum.Follower	208-435bc0ff5db851d341c373bc515f6a18691e6c4	17.0	36.0	64.0
org.apache.zookeeper.server.quorum.Follower	211-3a6a4ba7fc538f066d7b691777320fb9b2a5f93	18.0	36.0	65.0

**Fonte:** Autoria própria.

**Figura 15 – Estatísticas das métricas do ZK.**

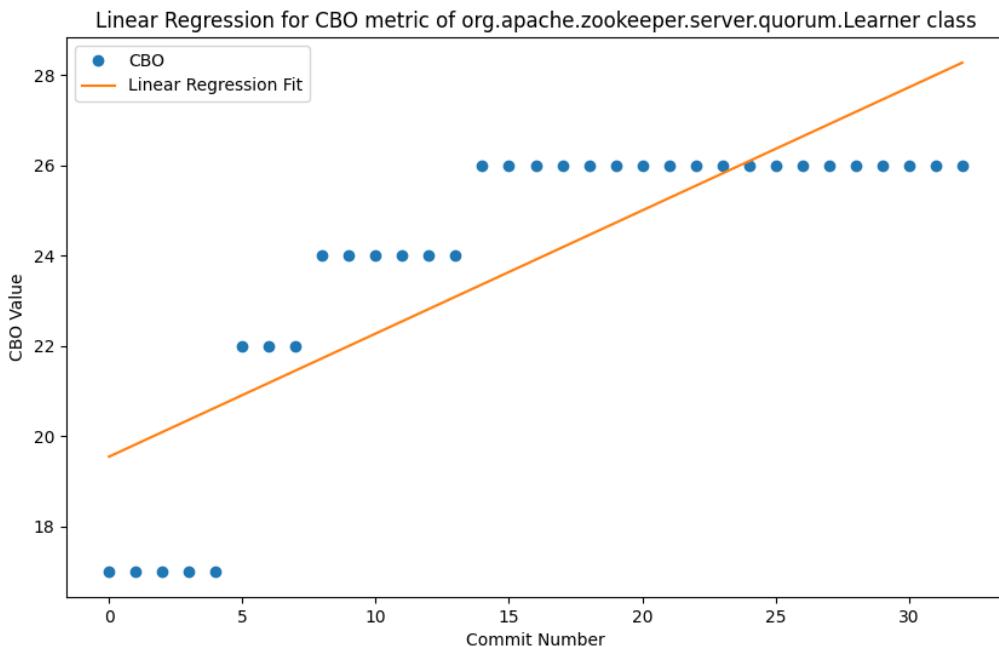
Class	Type	Changed	CBO Min	CBO Max	CBO Avg	CBO Q3	WMC Min	WMC Max	WMC Avg	WMC Q3	RFC Min	RFC Avg	RFC Max	RFC Avg
org.apache.zookeeper.server.quorum.QuorumPeerServerState	enum	115	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
org.apache.zookeeper.server.quorum.QuorumPeersQuorumServer	innerclass	115	0.0	4.0	2.139	4.0	1.0	41.0	21.835	41.0	0.0	13.0	6.061	13.0
org.apache.zookeeper.server.quorum.QuorumPeer	class	115	21.0	41.0	35.948	41.0	67.0	228.0	161.887	226.0	33.0	137.0	92.53	137.0
org.apache.zookeeper.server.quorum.QuorumPeersResponderThread	innerclass	115	2.0	5.0	4.304	5.0	10.0	12.0	11.374	12.0	11.0	15.0	14.078	15.0
org.apache.zookeeper.ClientInnsPacket	innerclass	89	5.0	8.0	6.697	8.0	4.0	8.0	6.135	8.0	11.0	16.0	14.761	16.0

**Fonte:** Autoria própria.

*org.apache.zookeeper.server.quorum.Learner*. Infelizmente, ela não sofreu reduções nas métricas analisadas, conforme exposto pelas Figuras 16, 17 e 18.

Observou-se que parte da complexidade na análise de um *commit* propriamente dito reside na existência de uma considerável poluição nos dados. Muitos *commits* no repositório do ZK abrangem alterações em inúmeros arquivos, frequentemente incorporando refatorações não diretamente correlacionadas às mensagens de *commit*. Essa complexidade aumenta a dificul-

**Figura 16 – Regressão linear da métrica CBO da classe *org.apache.zookeeper.server.quorum.Learner* do ZK**



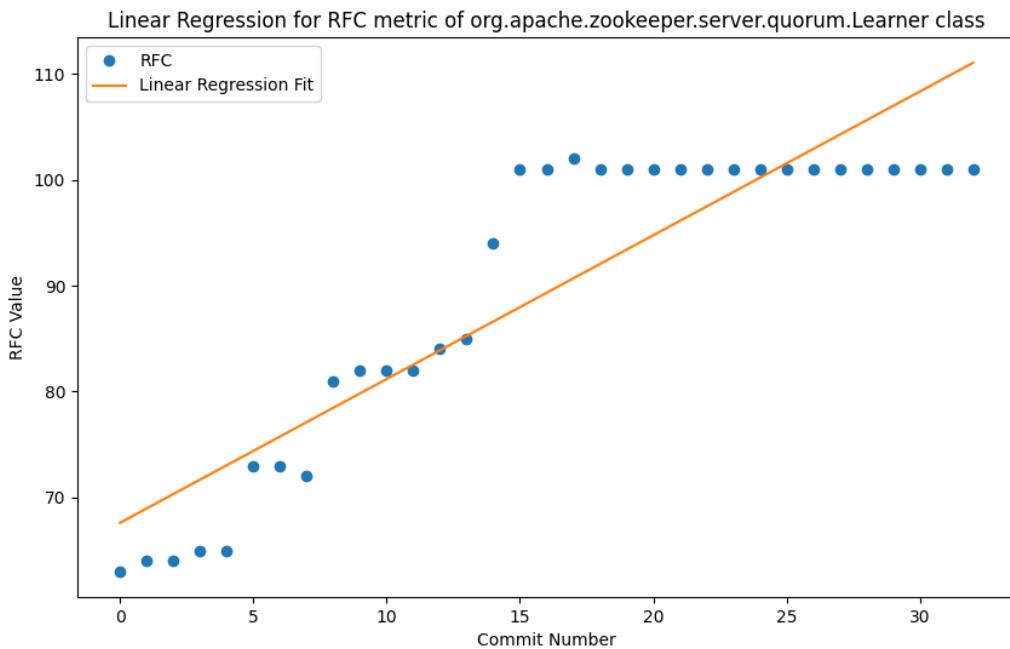
**Fonte:** Autoria própria.

dade de discernir as motivações subjacentes por trás das mudanças no código-fonte, tornando desafiador o processo de vincular essas alterações a possíveis vulnerabilidades identificadas externamente. Este fenômeno destaca a importância de uma abordagem criteriosa na interpretação do histórico de *commits* para extrair conclusões significativas sobre a evolução do *software* em relação à qualquer aspecto desejado.

Com a complexidade da análise dos *commits* em mente, a etapa seguinte concentra-se na aplicação de uma heurística definida e implementada por meio da ferramenta WEM (SILVA, 2023c). Após várias iterações de modulação, a heurística está agora refinada e não exige mais a busca manual por *keywords* em mensagens de *commit* ou nomes de classes. A nova abordagem inclui o uso do *script* de geração das métricas CK (localizado em *PyDriller/code\_metrics.py*) e o aprimoramento do algoritmo *PyDriller/metrics\_changes.py*, que agora reflete a heurística desenvolvida e apresenta melhorias significativas em desempenho. Com isso, o algoritmo produz uma lista de candidatos relevantes do ponto de vista da ES, que ainda precisam ser validados quanto à sua relevância para SDs, como ilustrado na Figura 7 e na Tabela 2.

Dentre os candidatos interessantes para a ES apontados pelo *script*, encontra-se novamente a classe *Follower* do quórum do ZK e poucas dezenas de outros candidatos também. Após dar esses candidatos ao *script* de integração do *Google Gemini* (encontrado em *Gemini/gemini.py*), a lista é reduzida novamente conforme exposto na Tabela 4 e um novo candidato chama a atenção, denominado *BookKeeper*.

**Figura 17 – Regressão linear da métrica RFC da classe *org.apache.zookeeper.server.quorum.Learner* do ZK**



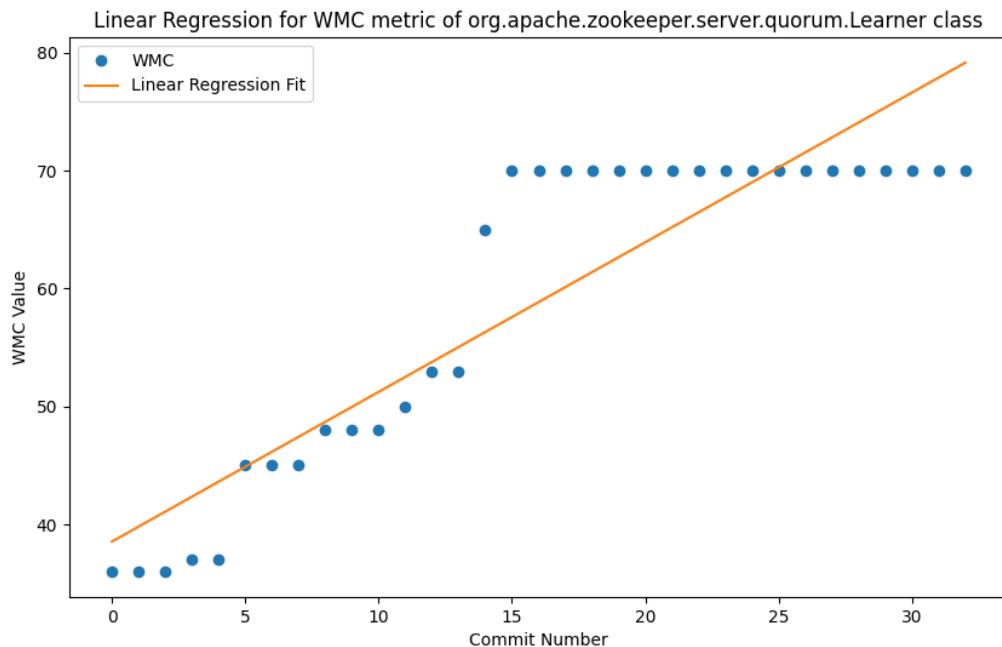
**Fonte:** Autoria própria.

O candidato identificado na classe *org.apache.bookkeeper.client.BookKeeper*, no commit *c41bd3f83e6ad99f03261ba21295eed5151eadda*, apresentou um comportamento interessante conforme análise realizada pelo *Gemini* sobre os termos extraídos dos métodos invocados dessa classe. A análise da refatoração correspondente, contida no arquivo *.diff* associado ao *commit*, foi revisada em conjunto com um especialista em SD, buscando esclarecer a relevância e a eficácia das alterações realizadas.

Durante essa revisão, foi encontrado um comentário na refatoração sugerindo a introdução de uma API de *streaming* para o *BookKeeper*, que teria como objetivo facilitar o armazenamento de *checkpoints/snapshots*. Contudo, a análise do código mostrou que as mudanças efetivamente implementadas não concretizaram essa funcionalidade, ou pelo menos não ainda. Em vez disso, o *commit* removeu uma quantidade significativa de código relacionado ao mecanismo de quórum e simplificou substancialmente a classe *BookKeeper*. Isso evidenciou uma desconexão entre a intenção expressa na mensagem do *commit* e as modificações reais introduzidas, sugerindo que refatorações relevantes poderiam não estar sendo aplicadas de forma consistente ao nível de *commit*. Essa percepção foi corroborada pelo especialista, que confirmou que, apesar das refatorações de encapsulamento na classe *BookKeeper*, não houve implementação de mudanças consideradas significativas do ponto de vista de SD.

Outro ponto relevante é que a refatoração realizada envolveu dezenas de arquivos, introduzindo modificações extensas que, na maioria dos casos, não estavam diretamente rela-

**Figura 18 – Regressão linear da métrica WMC da classe *org.apache.zookeeper.server.quorum.Learner* do ZK**



**Fonte:** Autoria própria.

cionadas à mensagem do *commit*. A discrepância entre o que é prometido na mensagem e o que é efetivamente modificado pode indicar que as boas práticas de refatoração são mais bem aplicadas em um nível mais elevado, como em um *Pull Request* (PR). Para investigar essa hipótese, uma análise preliminar foi conduzida, sugerindo que a revisão de *labels* e descrições de PRs/*Issues* poderia oferecer *insights* mais precisos sobre a natureza das refatorações.

Diante do exposto, o ZK não se mostra um repositório promissor para identificar candidatos que sejam simultaneamente relevantes para ES e para o contexto educacional de SD, considerando a abordagem proposta e aplicada. Além disso, o processo de análise tornou-se significativamente mais automatizado em comparação ao início do estudo do ZK e várias simplificações serão feitas devido à extensão deste texto. Dessa forma, a análise será direcionada para outros repositórios.

No que diz respeito ao *Kafka*, o *Gemini* identificou um número ainda menor de candidatos, especificamente 13, e a análise desses candidatos não foi aprofundada neste estudo. Isso se deve ao fato de que, assim como o ZK, o *Kafka* é um produto da *Apache* (FOUNDATION, 1999), e há uma intenção de evitar possíveis viéses associados a esses produtos. Portanto, conforme já discutido na seção anterior 4.2, foi implementado o algoritmo *PyDriller/repositories\_picker.py* com o objetivo de aplicar critérios de seleção mais abrangentes e identificar repositórios alternativos que atendam aos critérios estabelecidos na Seção 3.2.3.

Dentre os candidatos do *Kafka*, alguns são os mesmos encontrados no ZK, afinal o ZK foi inicialmente desenvolvido dentro do *Kafka* e só recentemente ele foi substituído pelo *KRaft*. Os outros candidatos apresentaram relação com a classes que tratam da leitura e processamento de fila de mensagens (do inglês, *message queues*), mas após breves investigações, decidiu-se realmente trocar para outros repositórios que não fossem produtos *Apache*.

#### 4.4 Análises com a Heurística Proposta

Esta seção apresenta as análises realizadas após o estabelecimento da heurística proposta com base nos critérios definidos. Aqui, são discutidos os resultados obtidos a partir da aplicação da heurística por meio do uso da ferramenta WEM (SILVA, 2023c), destacando as melhorias na seleção de candidatos para exemplos trabalhados.

Dada a execução do algoritmo de seleção de repositórios, foram sugeridos pela ferramenta três repositórios, o *CorfuDB*, *ScaleCube-Services* e *Molecular-Java*, conforme exposto pela Figura 5. A análise desses repositórios ocorreu no fim do prazo da pesquisa, a qual foi intercalada pela: (i) busca e leitura de artigos, (ii) refinamento da heurística, (iii) implementação da ferramenta WEM (SILVA, 2023c), (iv) criação da métrica *MethodInvocationCounter* no conjunto do CK (SILVA, 2024b), e (v) escrita de artigos, como o resumo já publicado no EduComp 2024 (SILVA *et al.*, 2024). A quantidade de candidatos em cada uma das etapas de filtro da heurística pode ser visualizada conforme as Tabelas 2 e 4, além dos ganhos na redução de escopo ao incluir a análise do *Gemini* expostos na Tabela 5.

A Tabela 2 apresenta a quantidade de classes em cada repositório, a quantidade final de candidatos gerados após a execução da terceira etapa (*PyDriller/metrics\_changes.py*), e a redução de escopo obtida com esta etapa da ferramenta. A redução de escopo é calculada pela razão entre a diferença na quantidade de classes e a quantidade de candidatos, dividida pela quantidade de candidatos. Este cálculo pode ser visualizado da seguinte forma:

$$\text{Redução de Escopo} = \frac{\text{Quantidade de Classes} - \text{Quantidade de Candidatos}}{\text{Quantidade de Candidatos}}$$

Esta fórmula busca demonstrar a eficácia do processo em reduzir o número de trechos de código a serem avaliados.

Ao término da execução do terceiro algoritmo, obtém-se uma lista de *commits* candidatos que apresentaram quedas nas métricas, sendo que uma queda maior que 0 já é considerada válida. Além disso, são identificados padrões de refatoração associados. Inicialmente, a heurística utilizada era extremamente restritiva, considerando apenas quedas superiores a 15% e detectando padrões específicos de refatoração. No entanto, essa abordagem rigorosa resultou em uma lista vazia de candidatos, então sucessivas iterações foram feitas de modo a flexibilizar a heurística.

**Tabela 2 – Quantidade de classes, candidatos e a redução de escopo dos repositórios após execução do terceiro algoritmo *PyDriller/metrics\_changes.py*.**

Nome do Re却tório	Quantidade de Classes	Quantidade de Candidatos (3 <sup>a</sup> etapa)	Redução de Escopo
<i>Apache Kafka</i>	5007	319	93.628%
<i>CorfuDB</i>	954	590	38.155%
<i>Moleculer-java</i>	224	57	74.554%
<i>Scalercube-services</i>	246	122	50.407%
<i>ZooKeeper</i>	941	43	95.430%

Fonte: Autoria própria.

Até esta etapa, conforme exposto pela Tabela 2, observa-se um certo grau de inconsistência no percentual de redução de escopo entre os repositórios selecionados, o que reflete a variação na efetividade da heurística conforme o repositório em análise. Tome-se como exemplo o *ZooKeeper*, com suas 941 classes e mais de 2500 *commits*. Ao aplicar um filtro rigoroso, como a definição de um limiar superior a 10% para o *threshold* de quedas e a seleção de *commits* candidatos apenas por refatorações específicas, como *Extract Method*, *Extract Class*, *Pull Up Method*, *Push Down Method* e *Extract Superclass*, não se obtém nenhum candidato.

Diante desse cenário, foram realizadas algumas modificações na heurística para flexibilizá-la, a fim de evitar resultados vazios. Essas alterações incluíram a redefinição do limiar de quedas para qualquer valor maior que 0, de modo a capturar qualquer redução, bem como a remoção do filtro restritivo para padrões específicos de refatoração.

Devido às características dos resultados gerados pela IA generativa, que podem sofrer alterações a cada execução, inúmeras rodadas foram realizadas para efetuar um *fine-tuning* do modelo, tornando os resultados consistentes e reproduutíveis. O aprimoramento dos parâmetros do modelo de IA foi feito com base na leitura da documentação oficial (AI, 2023), compreendendo seus parâmetros e adicionando uma *feature* que realiza múltiplas execuções com o mesmo *input*. Para avaliar a consistência dos resultados gerados, foi implementada uma função que calcula a similaridade dos *outputs* utilizando a similaridade do cosseno.

Essa similaridade é calculada utilizando a distância do cosseno entre os resultados produzidos pela IA generativa. Esses resultados são representados em um modelo vetorial *bag of words*, com os pesos dos termos calculados através da métrica *Term Frequency - Inverse Document Frequency* (TF-IDF).

Essa abordagem permitiu verificar a consistência dos *outputs* e definir um intervalo de confiança para a similaridade média obtida, aumentando a confiabilidade das saídas geradas pelo modelo após o *fine-tuning*. O grau de similaridade entre os *outputs* das 5 execuções foi de 84%, com um intervalo de confiança de 95%, indicando que a similaridade estava entre 79% e 89%. Mais detalhes estão na Tabela 3.

A Tabela 2 apresentou a quantidade de classes em cada repositório, a quantidade final de candidatos gerados após a execução do terceiro algoritmo, e a redução de escopo obtida com esta etapa da ferramenta. Em seguida, a Tabela 4 mostra os resultados obtidos após a

**Tabela 3 – Similaridade média e intervalos de confiança para diferentes números de execuções.**

Execuções	Similaridade Média	Intervalo de 95% de Confiança
5	84%	79% a 89%
10	82%	79% a 86%
15	78%	73% a 85%
20	77%	70% a 79%
25	75%	71% a 77%

**Fonte:** Autoria própria.

execução do quarto algoritmo (*Gemini/gemini.py*), evidenciando uma redução de escopo ainda mais significativa. Dessa forma, a sequência apresentada reflete a ordem das etapas de processamento.

**Tabela 4 – Quantidade de classes, candidatos e a redução de escopo dos repositórios após execução do quarto algoritmo *gemini.py*.**

Nome do Repositório	Quantidade de Classes	Quantidade de Candidatos (4 <sup>a</sup> etapa)	Redução de Escopo
<i>Apache Kafka</i>	5007	12	99.760%
<i>CorfuDB</i>	954	23	97.589%
<i>Moleculer-java</i>	224	25	89.238%
<i>Scalcube-services</i>	246	16	93.496%
<i>ZooKeeper</i>	941	13	98.618%

**Fonte:** Autoria própria.

A Tabela 5 apresenta a quantidade de candidatos na 3<sup>a</sup> e 4<sup>a</sup> etapas para cada repositório, bem como a redução de escopo calculada entre essas duas etapas. Esta tabela ilustra o impacto da redução de escopo após a aplicação dos algoritmos, evidenciando uma redução significativa em relação ao número de candidatos da 3<sup>a</sup> etapa. Os resultados destacam melhorias adicionais na filtragem de candidatos, demonstrando um avanço no processo de refinamento da heurística desenvolvida, se comparado à Tabela 2 (que apresenta os dados referentes à 3<sup>a</sup> etapa da ferramenta).

**Tabela 5 – Redução de escopo da quantidade de candidatos da 3<sup>a</sup> e 4<sup>a</sup> etapas.**

Nome do Repositório	Quantidade de Candidatos (3 <sup>a</sup> etapa)	Quantidade de Candidatos (4 <sup>a</sup> etapa)	Redução de Escopo
<i>Apache Kafka</i>	319	12	96.238%
<i>CorfuDB</i>	590	23	96.102%
<i>Moleculer-java</i>	57	25	56.140%
<i>Scalcube-services</i>	122	16	86.885%
<i>ZooKeeper</i>	43	13	69.767%

**Fonte:** Autoria própria.

Os resultados apresentados na Tabela 6 fornecem uma visão abrangente da redução de escopo obtida nos repositórios analisados. A média de 95,740% reflete uma redução substancial na quantidade de candidatos após a quarta etapa do processo de avaliação, indicando que, em média, uma grande parte do escopo foi efetivamente reduzido.

Esses resultados indicam que a heurística de seleção aplicada foi bastante eficaz em reduzir o escopo dos dados, o que é um indicativo positivo da capacidade da metodologia em filtrar trechos de código relevantes e reduzir a complexidade dos exemplos trabalhados.

**Tabela 6 – Estatísticas descritivas da redução de escopo para os repositórios analisados.**

Estatística	Redução de Escopo
Média	95.740%
Mediana	97.589%
Desvio Padrão	3.877%
Máximo	99.760%
Mínimo	89.238%

**Fonte:** Autoria própria.

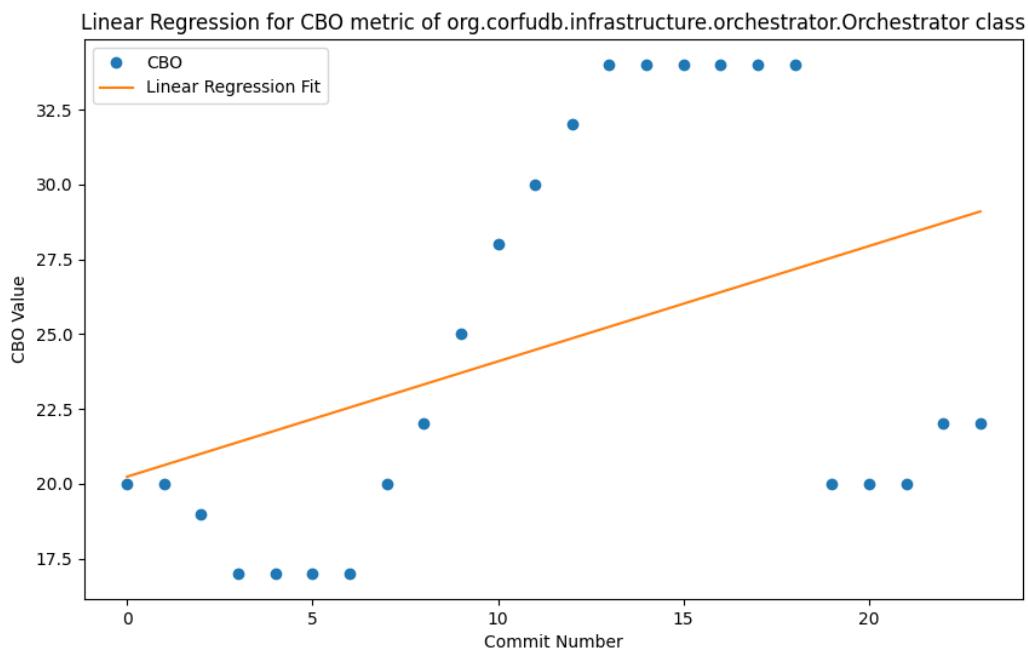
O *CorfuDB* apresentou um total de vinte e três candidatos expostos pelo *Gemini*. Esse candidatos são classes e métodos que ilustram conceitos centrais na área de SDs, oferecendo valiosos exemplos educativos. A seguir, discorre-se sobre cada um desses candidatos, relacionando-os com temas relevantes para o ensino de SDs.

Entre os candidatos, o mais relevante refere-se à classe *org.corfudb.infrastructure.orchestrator.Orchestrator*, que é responsável por gerenciar a adição de nós (*addNode*) e tratar requisições de consulta (*handleQuery*). Este exemplo reflete a importância de um orquestrador centralizado na manutenção do estado do *cluster*. Em questão, trata-se do *commit 805ef812d278dbec2a6f3bba458114b1264ac009*, que introduz a implementação de *Remote Procedure Call* (RPC)s de gerenciamento utilizando *Protobuf*. Este *commit* adiciona suporte para roteamento de mensagens *Protobuf* e a implementação de RPCs de gerenciamento com base nessas mensagens, como parte de um esforço para suportar atualizações contínuas. No entanto, é importante observar que, apesar das melhorias introduzidas, o *commit* alterou mais de 50 arquivos e mais de mil linhas de código, não conseguindo refletir toda a refatoração completa em um único *commit*. Ou seja, mesmo com os ganhos observados na Tabela 4, cada *commit* apresenta-se extremamente denso, complexo e poluído.

Este candidato destaca a complexidade envolvida na implementação de novas funcionalidades em SDs e a necessidade de considerar a amplitude das modificações em análise de código. Com relação às métricas, todas tiveram incremento ao longo do tempo, todavia elas encontram-se na heurística pois houve uma queda substancial no CBO entre *commits* que alteravam a classe em questão, conforme exposto na Figura 19.

Entre os candidatos, um deles é pouco relevante para explicar conceitos de SD, mas aparenta ser válido talvez mostrar questões como tratamento de falhas. Esse candidato é referente ao *commit ec64ef74891b32d341b2d3ec553d9e64605c87e4* da classe *org.corfudb.infrastructure.ManagementAgent*, que lida com épocas fora de fase (*correctOutOfPhaseEpochs*) e com a recuperação do *cluster* (*runRecoveryReconfiguration*). Em SDs, uma “época” é um intervalo de tempo em que um sistema opera sob um conjunto específico de con-

**Figura 19 – Regressão linear da métrica CBO da classe *org.corfudb.infrastructure.orchestrator.Orchestrator* do CorfuDB**



**Fonte:** Autoria própria.

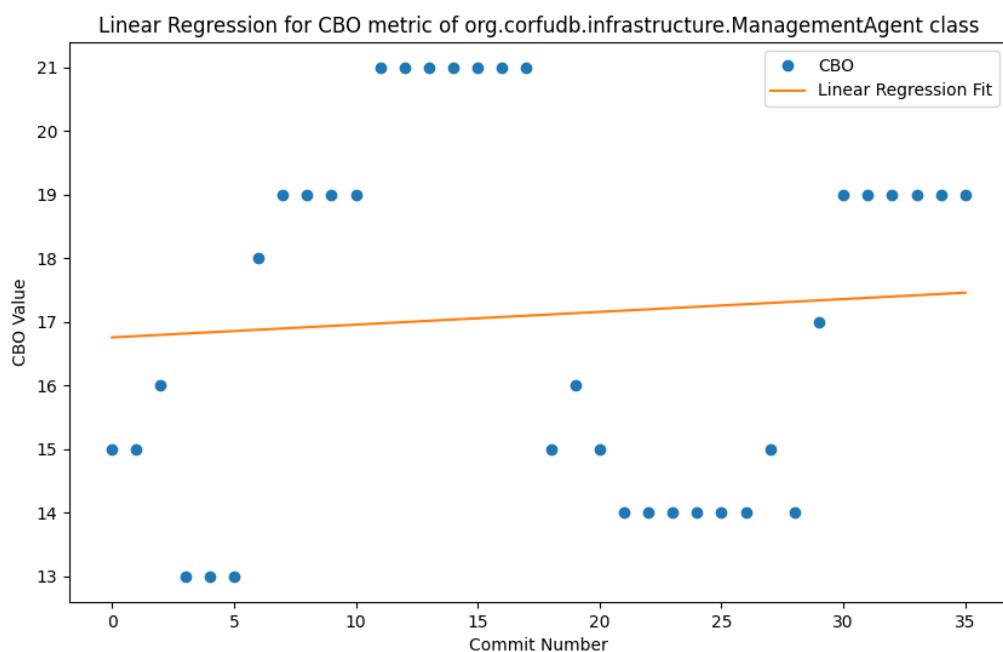
dições ou configurações. A gestão de épocas é fundamental para garantir a consistência e a sincronização entre os diferentes componentes do sistema, especialmente após falhas ou mudanças na configuração. Portanto, a capacidade de corrigir épocas fora de fase é essencial para manter a integridade do sistema e garantir a continuidade do serviço. Este exemplo ilustra o processo de recuperação e a tolerância a falhas, aspectos fundamentais em SDs, onde a resiliência e a capacidade de recuperação são essenciais para a continuidade do serviço diante de falhas de componentes.

Outro conceito relevante encontrado na análise de candidatos é o uso de *heartbeats*, que desempenha um papel relevante em SD. Nesta refatoração, a classe responsável por gerenciar as requisições de *heartbeat* é destacada na função `handleHeartbeatRequest`. O conceito de *heartbeat* refere-se a mensagens periódicas enviadas entre os componentes de um SD para monitorar a saúde e a disponibilidade de cada nó. Essas mensagens permitem que um nó verifique a presença e a resposta de outros nós, ajudando a detectar falhas e a garantir a comunicação contínua. No código analisado, a implementação do *heartbeat* envolve o envio de uma resposta que inclui as métricas do servidor local e a visão do nó sobre o estado do cluster, permitindo que o sistema avalie seu estado atual e ajuste suas operações conforme necessário. Esse mecanismo é vital para manter a coesão e a eficiência em um ambiente distribuído, pois assegura que todos os nós estejam operacionais e sincronizados, facilitando a

recuperação rápida e a continuidade do serviço em caso de falhas. Portanto, a implementação e o gerenciamento de *heartbeats* são essenciais para garantir a resiliência e a robustez dos SD.

Novamente, trata-se de um *commit* extremamente complexo e poluído. Com relação às métricas, todas tiveram incremento ao longo do tempo, mas bem sutil. Todavia, elas encontram-se na heurística pois houve quedas substanciais no CBO entre *commits* que alteravam a classe em questão, conforme exposto na Figura 20.

**Figura 20 – Regressão linear da métrica CBO da classe *org.corfudb.infrastructure.ManagementAgent* do CorfuDB**



**Fonte:** Autoria própria.

Por fim, embora os repositórios *ScaleCube-Microservices* e *Molecular-Java* tenham sido inicialmente selecionados, a análise dos candidatos extraídos desses repositórios não foi realizada devido às restrições de tempo e à necessidade de priorizar a validação para a conclusão da parte qualitativa da pesquisa.

#### 4.5 Validação de candidatos

A redução de repositórios analisados visou reduzir o escopo da análise e concentrar os esforços na validação da heurística desenvolvida, selecionando dois candidatos: um com relevância para ES e SDs, e outro aparentemente menos promissor. Este procedimento visa responder às questões formuladas na Seção 3.1 e são respondidos no Capítulo 5.

Para alcançar esse objetivo, foram elaborados formulários destinados a avaliar a relevância educacional dos candidatos, tanto no contexto de ES quanto de SD. Os formulários foram

projetados com o intuito de avaliar trechos de código provenientes de repositórios de *software* livre, com foco em identificar seu potencial educacional, tanto em termos de competências em ES quanto em SDs. Para cada candidato, foram criados dois questionários: um voltado para a análise sob a perspectiva da ES e outro sob a ótica de SDs. As perguntas desses formulários podem ser encontradas nos Apêndices A e B.

O questionário relacionado à ES foi estruturado para investigar a legibilidade, organização e aderência às boas práticas de programação do trecho de código avaliado. Perguntas como “Quão bem estruturado está o trecho de código em termos de legibilidade e organização?” e “O código segue boas práticas de programação?” visavam aferir a qualidade do código em aspectos fundamentais para a manutenibilidade e reutilização, aspectos relevantes no ensino de ES. Ademais, o formulário solicitava ao respondente que considerasse a facilidade de testabilidade do código, a clareza da documentação e a potencialidade de reutilização do trecho em outros projetos ou contextos. A intenção era assegurar que o candidato selecionado não apenas exemplificasse boas práticas de ES, mas também fosse um recurso didático eficaz.

No contexto de SDs, o questionário focou na capacidade do trecho de código de ilustrar conceitos específicos de SDs, conforme delineados pelo currículo *NSF/IEEE-TCPP*. Questões como “Quais conceitos específicos de SDs são melhor ilustrados por este trecho de código?” e “Quão relevante é o trecho de código para o ensino de conceitos de SDs?” buscaram validar a relevância do código em termos de didática de SDs. Além disso, foram abordadas a facilidade de adaptação do código para fins educacionais e a necessidade de adaptações adicionais para sua efetiva utilização em sala de aula.

Esses formulários foram feitos de forma cega, sem dar indícios da qualidade de cada candidato e desempenharam um papel essencial na validação da heurística proposta, permitindo uma avaliação sistemática e criteriosa dos candidatos selecionados. Através da aplicação desses instrumentos, foi possível: (i) validar a qualidade da heurística desenvolvida e (ii) identificar trechos de código que, além de serem tecnicamente relevantes, possuem um grande potencial didático, tanto no ensino de ES quanto de SDs, contribuindo assim para a criação de exemplos trabalhados que facilitam a compreensão de conceitos complexos nessas áreas.

As classes candidatas selecionadas foram as duas do *CorfuDB* discutidas na Seção 4.4, *org.corfudb.infrastructure.orchestrator.Orchestrator* e *org.corfudb.infrastructure.-ManagementAgent*, representando os candidatos 1 e 2, respectivamente.

O resultado das respostas dadas pelos especialistas de ES e SD mostrou que ambos os candidatos apresentam características importantes para fins didáticos, porém com nuances distintas em termos de aplicabilidade e clareza conceitual.

A Tabela 7 apresenta a média das respostas dos especialistas de ES para os candidatos analisados.

No caso do candidato 1, referente à classe *Orchestrator*, os especialistas de ES destacaram que, apesar de o código seguir boas práticas de programação e ser testável e reutilizável, ele foi considerado mal estruturado e de difícil compreensão devido à documentação limitada

**Tabela 7 – Média das respostas dos especialistas de ES para os candidatos**

<b>Perguntas</b>	<b>Classe Orchestrator</b>	<b>Classe ManagementAgent</b>
<b>Quão bem estruturado está o trecho de código em termos de legibilidade e organização?</b>	Decente	Bem Estruturado
<b>O código segue boas práticas de programação (ex.: nomeação de variáveis, modularidade, tratamento de erros)?</b>	Concordo	Concordo plenamente
<b>Este trecho de código pode ser reutilizado em outros projetos ou contextos?</b>	Reutilizável	Pouco Reutilizável
<b>O trecho de código é facilmente testável?</b>	Testável	Testável
<b>O código está bem documentado, com comentários e explicações claras?</b>	Documentado	Documentado
<b>Que melhorias na documentação poderiam ser feitas para facilitar a compreensão do código?</b>	Difícil avaliar devido à falta de explicações detalhadas no <i>commit</i>	A documentação está parcialmente clara, mas faltam detalhes sobre a arquitetura geral
<b>Você considera este trecho de código um bom exemplo para ilustrar práticas de Engenharia de Software em um curso?</b>	Sim, com limitações	Sim, para disciplinas mais avançadas
<b>Que outras características de Engenharia de Software poderiam ser exploradas neste trecho de código para fins didáticos?</b>	Estrutura de dados, sistemas distribuídos	Padrões de projeto e princípios <i>SOLID</i>

**Fonte: Autoria própria.**

e ao contexto complexo do *commit*. A mensagem principal do *commit* foi considerada muito breve, sem explicar adequadamente as alterações realizadas. Assim, a dificuldade em entender as modificações essenciais e acessórias compromete a utilização do código como exemplo didático em sua forma atual. Embora haja aspectos positivos, como a relação com casos de teste, a falta de informações adicionais sobre medidas de qualidade e resumo de refatorações torna a exploração de outras características do código difícil. Portanto, o candidato 1 pode ser usado em cursos de ES com cautela, considerando a complexidade do código e a necessidade de uma documentação mais acessível.

Os especialistas de SD avaliaram o candidato 1 e identificaram que o código ilustra conceitos fundamentais de SD, incluindo gerenciamento de *threads*, orquestração de tarefas, tratamento de falhas e registro de ações (*logs*). No entanto, a adaptação do código para o ensino em sala de aula foi considerada difícil devido à sua extensão e ao fato de envolver muitas alterações em diversos arquivos (50 arquivos alterados). Apesar das dificuldades, os especialistas sugeriram que o código poderia ser utilizado como exemplo de conceitos teóricos, como o gerenciamento de *threads* e *logs*, em vez de ser desenvolvido em sala de aula. Para uma avaliação mais precisa, seria útil um resumo das principais classes alteradas para entender

melhor a efetividade do código. Apesar das dificuldades e da necessidade de adaptações, o candidato 1 foi considerado relevante para o ensino de conceitos de SD e recomendado para ser mostrado em aula, especialmente para ilustrar como esses conceitos são aplicados em sistemas reais e complexos.

Por outro lado, o candidato 2, referente à classe *ManagementAgent*, recebeu avaliações ligeiramente diferentes. Este código foi considerado bem estruturado e alinhado com boas práticas de programação, especialmente em termos de modularidade e uso de padrões de projeto, o que o torna um exemplo didático robusto para disciplinas de ES. No entanto, sua reutilização em outros projetos ou contextos foi vista como limitada devido às dependências de outros componentes do sistema. A documentação foi considerada adequada, mas o escopo do *commit*, que envolveu alterações em vários arquivos, dificultou a avaliação completa. Apesar dessas limitações, o candidato 2 é recomendado como exemplo didático para explicar conceitos complexos, especialmente em cursos avançados que abordam tópicos como resiliência e recuperação em SDs.

Para especialistas de SD, o candidato 2 se destacou pela ilustração de conceitos fundamentais, como tratamento de falhas e verificação de estados via *heartbeat*. Embora a adaptação desse código para um exemplo trabalhado em sala de aula possa ser desafiadora, devido à sua complexidade e dependências, ele foi considerado relevante para o ensino de SD. Portanto, o candidato 2 é recomendado como um exemplo a ser mostrado em aula para a explicação de conceitos complexos de SD, sendo particularmente útil para cursos que abordam tópicos avançados, como resiliência e recuperação em SDs.

Durante a avaliação dos trechos de código, foram utilizadas perguntas baseadas em uma escala *Likert* de cinco pontos, permitindo a conversão de opiniões subjetivas em valores numéricos para posterior comparação. Os especialistas foram questionados sobre aspectos como a clareza, a estrutura, e a aplicabilidade didática dos trechos de código selecionados. Perguntas como “Quão bem estruturado está o trecho de código?” e “Você considera este trecho de código um bom exemplo para ilustrar práticas de Engenharia de Software em um curso?” servem para avaliar os candidatos, mas também para medir o grau de concordância dos avaliadores.

A análise das respostas revelou uma consistência significativa, já que as variações observadas entre as avaliações foram mínimas. A uniformidade nas respostas sugere que a heurística desenvolvida foi capaz de identificar exemplos claros e apropriados para uso educacional, corroborando sua eficácia na seleção de candidatos para o contexto de SD. A categorização dessas respostas qualitativas permitiu identificar padrões, o que resultou na extração de pontos relevantes sem a necessidade de incluir todas as respostas individuais no apêndice.

Os resultados obtidos com a aplicação dos formulários evidenciam a eficácia da heurística proposta para a seleção de candidatos, uma vez que ela foi capaz de reduzir significativamente o escopo de análise, conforme já exposto nas Tabelas 2 e 4. A heurística identifica classes que, embora apresentem níveis variados de complexidade e aplicabilidade didática, possuem características essenciais para a ilustração de conceitos tanto de ES quanto de SDs.

A seleção dos candidatos, como evidenciado pelas respostas dos especialistas, permitiu a escolha de exemplos que, se bem adaptados, além de serem tecnicamente sólidos, oferecem um grande potencial didático para o ensino dessas áreas.

As respostas dos especialistas de ES demonstraram uma notável consistência, com variações de apenas um grau na escala *Likert* entre as avaliações, sublinhando a robustez e a precisão na avaliação da heurística. Isso é particularmente significativo dado o caráter subjetivo das questões abordadas, indicando um consenso sólido entre os avaliadores sobre a eficácia da heurística. A capacidade da heurística em filtrar trechos de código relevantes, alinhados aos objetivos educacionais, reflete sua eficácia na curadoria de exemplos que possam ser trabalhados em sala de aula, contribuindo assim para a formação de profissionais mais bem preparados para lidar com os desafios práticos de ES e SDs.

O processo de validação e seleção dos candidatos foi crucial para garantir a relevância educacional e a aplicabilidade prática dos exemplos de código. A abordagem adotada demonstrou a eficácia da heurística em filtrar e selecionar candidatos que possuem potencial para enriquecer o ensino de ES e SDs, permitindo um aprendizado mais aprofundado e contextualizado dos conceitos abordados. Entretanto, algumas melhorias em versões futuras acabam surgindo, como analisar as refatorações a nível de PR, de modo a conseguir abranger todo o escopo da refatoração.

#### **4.6 Limitações**

Este trabalho de pesquisa exploratória apresenta algumas limitações decorrentes da sua natureza interdisciplinar, que abrange três grandes áreas: SD, ES e a esfera educacional, além do uso de métricas objetivas para um problema com questões subjetivas.

A primeira limitação refere-se à subjetividade inerente às questões de pesquisa definidas na Seção 3.1. Embora a primeira questão possa ser respondida por meio da detecção de padrões de refatoração e do comportamento das métricas, a segunda e a terceira questões apresentam um grau maior de subjetividade. Consequentemente, a ferramenta desenvolvida não é totalmente automatizada, exigindo a avaliação de especialistas, o que torna o processo custoso e demanda uma análise cuidadosa por parte do avaliador.

A segunda limitação refere-se à análise dos repositórios descrita na Seção 4.3. Observou-se que, em alguns casos, um *commit* isolado pode não refletir todo o escopo de melhorias realizadas em uma classe. Nesse sentido, uma análise em nível de PR poderia proporcionar resultados mais abrangentes. Todavia, se em nível de *commit* já é possível encontrar refatorações extremamente extensas e complexas, envolvendo múltiplos arquivos, esse desafio se intensificaria ao se considerar o nível de PR. Além disso, poucos repositórios seguem rigorosamente a convenção de *commits* bem documentados e estruturados, o que facilitaria a identificação de refatorações significativas, sem a interferência de alterações irrelevantes (Conventional Commits, 2023).

A terceira e última limitação diz respeito à impossibilidade de realizar análises empíricas sobre certas melhorias de desempenho, como a avaliação das trocas de mensagens em tempo real, devido à restrição imposta pelo uso exclusivo de métricas estáticas de código. Essa restrição impede a realização de avaliações práticas por meio da execução de SDs e de simulações em larga escala, o que limita a obtenção de percepções detalhadas sobre o desempenho em contextos que envolvem uma quantidade massiva de requisições sendo enviadas ao SD.

## 4.7 Ameaças à validade

Este estudo, por sua natureza exploratória e interdisciplinar, está sujeito a diversas ameaças à validade, que podem afetar a robustez dos resultados e a generalização das conclusões. As ameaças foram categorizadas em quatro tipos: interna, externa, de construção e de conclusão.

### 4.7.1 Ameaças à Validade Interna

- **Subjetividade na avaliação de candidatos:** A segunda e a terceira questões de pesquisa, que se concentram na relevância e adequação didática dos candidatos, são inherentemente subjetivas. A avaliação feita por especialistas pode variar, o que pode influenciar a classificação de um candidato como ‘bom’ ou ‘ruim’.
- **Poluição de dados em commits:** A análise de *commits* individuais pode ser prejudicada pela presença de refatorações não relacionadas à mensagem do *commit*, dificultando a identificação precisa das alterações relevantes.
- **Restrição ao uso de métricas estáticas:** A impossibilidade de utilizar métricas dinâmicas de código, como as métricas IPC, limita a análise de aspectos como desempenho e troca de mensagens em tempo real.

### 4.7.2 Ameaças à Validade Externa

- **Seleção de repositórios:** A escolha dos repositórios analisados pode não ser representativa de todos os projetos de *software* livre em SD, limitando a generalização dos resultados para outros contextos.
- **Viés de repositórios Apache:** A escolha inicial de projetos *Apache* (*Kafka* e *ZooKeeper*) pode ter introduzido um viés na análise, pois esses projetos podem ter características específicas que não se aplicam a outros repositórios.

#### 4.7.3 Ameaças à Validade de Construção

- **Integração com o modelo de IA:** O modelo de IA generativa (*Gemini*) utilizado para correlacionar termos com tópicos de SD é aleatório por natureza, o que pode gerar resultados inconsistentes ou imprecisos.
- **Implementação da heurística:** A implementação da heurística na ferramenta WEM pode conter erros ou falhas que afetem a precisão dos resultados.

#### 4.7.4 Ameaças à Validade de Conclusão

- **Interpretação dos resultados:** A interpretação dos resultados da análise pode ser influenciada por vieses do pesquisador, levando a conclusões errôneas ou incompletas.
- **Generalização dos resultados:** A generalização dos resultados para outros contextos pode ser limitada pelas ameaças à validade interna e externa, especialmente pela seleção de repositórios e pela subjetividade na avaliação de candidatos.

#### 4.7.5 Mitigação das Ameaças

- **Seleção de repositórios diversificados:** A escolha de repositórios de diferentes origens e domínios ajuda a aumentar a generalização dos resultados.
- **Refinamento do modelo de IA:** Realizar um *fine-tuning* dos parâmetros do modelo de IA generativa (*Gemini*) para garantir a máxima consistência e reproduzibilidade dos resultados. Esse processo ajuda a melhorar a precisão das correlações e a estabilidade dos resultados obtidos.
- **Validação por múltiplos especialistas:** A utilização de múltiplos especialistas para avaliar os candidatos ajuda a minimizar o impacto da subjetividade na avaliação.
- **Revisão crítica dos resultados:** A revisão crítica dos resultados por outros pesquisadores ajuda a minimizar o impacto de vieses do pesquisador.
- **Análise de PR:** A análise de PR, além de *commits* individuais, pode fornecer uma visão mais abrangente das refatorações e reduzir a poluição de dados.
- **Uso de métricas dinâmicas:** A exploração de métricas dinâmicas, como as métricas IPC, pode complementar a análise de qualidade do código e fornecer informações mais completas sobre o desempenho.

A identificação e mitigação dessas ameaças à validade são cruciais para garantir a robustez e confiabilidade dos resultados deste estudo. A aplicação de estratégias de mitigação,

como a utilização de múltiplos especialistas e a exploração de métricas dinâmicas, contribui para a obtenção de resultados mais precisos e generalizáveis.

#### **4.8 Considerações finais**

A heurística desenvolvida neste trabalho tem como objetivo principal auxiliar na seleção de trechos de código no contexto de SD que sejam adequados para a criação de exemplos didáticos em ES. Esta seleção é realizada por meio de uma análise detalhada da evolução do código nos repositórios selecionados, com base em uma abordagem sistemática. Essa abordagem inclui a escolha criteriosa dos projetos, a definição de métricas apropriadas e o desenvolvimento de ferramentas específicas para a análise. Inicialmente, foram estabelecidos critérios e métricas de qualidade. Em seguida, foram definidos os critérios para a seleção dos repositórios. Por fim, o processo de avaliação dos candidatos foi conduzido com base nessas métricas e critérios.

Com a heurística definida, ela foi então implementada através da ferramenta WEM (SILVA, 2023c), cuja discussão detalhada de todas as suas etapas encontram-se na Seção 4.2. Os ganhos em termos de redução de escopo promovidos pela heurística podem ser observados nas Tabelas 2 e 4. Para atingir tamanha redução de escopo, a heurística passou por inúmeras rodadas de *fine-tuning*.

A heurística, em sua concepção teórica, fundamenta-se na utilização das métricas CK, com especial ênfase nas métricas CBO, RFC e WMC, que são capazes de refletir a complexidade e a coesão dos trechos de código analisados. Além disso, a heurística inclui a detecção simultânea de padrões de refatoração, inseridos com o intuito de identificar melhorias na qualidade do *design* do código-fonte, oferecendo indícios de refatorações que possam ser especialmente relevantes para a ES.

Finalmente, a avaliação dos candidatos selecionados é realizada por especialistas em SD e ES, com o objetivo de validar a efetividade desses candidatos.

## 5 CONCLUSÕES

Este trabalho de pesquisa teve como objetivo principal desenvolver uma heurística para a seleção de trechos de código representativos de melhorias de *software* em SDs, utilizando métricas específicas de qualidade de código em projetos de *software* livre. A pesquisa visou aprofundar a compreensão da evolução do código em resposta a mudanças nas métricas escondidas, contribuindo para a construção de exemplos trabalhados que sejam aplicáveis ao contexto da ES.

A heurística desenvolvida neste trabalho baseia-se na análise de métricas de qualidade de código, como CBO, RFC e WMC, e na detecção de padrões de refatoração. A ferramenta WEM (SILVA, 2023c) foi implementada para automatizar o processo de coleta de dados e metadados relevantes para a seleção de exemplos de código-fonte. A ferramenta utiliza a heurística desenvolvida para filtrar e identificar os candidatos mais promissores possibilitando, de forma manual, a criação de exemplos trabalhados para ES a partir dos candidatos. A heurística foi capaz de atingir uma redução substancial na quantidade de candidatos, com uma média de 95,740% e desvio padrão pequeno de 3,877% na redução de escopo entre os repositórios analisados, conforme exposto na Tabela 6. Esses resultados reafirmam o potencial da heurística em filtrar trechos de código relevantes, reduzindo consideravelmente a complexidade na busca por candidatos para a criação de exemplos trabalhados.

A pesquisa também incluiu a implementação da métrica *MethodInvocationCounter* no projeto CK para Java (SILVA, 2024b), que implementa o conjunto de métricas CK (CHIDAMBER; KEMERER, 1994). Essa métrica permite uma análise detalhada dos métodos invocados em uma classe, registrando, de forma precisa, a frequência com que cada método é chamado. Essa contribuição proporciona uma visão mais abrangente e sintetizada das interações entre os métodos, enriquecendo a análise de código com informações valiosas sobre o comportamento e a complexidade das classes.

A heurística foi aplicada em diversos repositórios de código aberto, incluindo *Apache Kafka*, ZK, *CorfuDB*, *Scalcube-Services* e *Molecular-Java*. A análise dos resultados mostrou que a heurística é eficaz na identificação de trechos de código que demonstram melhorias na qualidade do *design* do código.

A integração da heurística com o modelo de IA generativa do *Google Gemini* (GOOGLE, 2023b) permitiu a identificação de candidatos que abordam tópicos de SD, como consenso distribuído, tolerância a falhas e gerenciamento de *clusters*. A análise dos termos extraídos dos métodos invocados por cada classe, juntamente com a lista de currículos de interesse definidos pelo *NSF/IEEE-TCPP* (NSF/TCPP, 2023a; NSF/TCPP, 2023b), permitiu ao modelo de IA refinar a lista de candidatos, correlacionando termos e comportamentos com tópicos de SD presentes nos currículos.

A avaliação dos candidatos por especialistas em SD e ES confirmou a relevância dos trechos de código selecionados para o ensino de SD e ES. Os especialistas avaliaram os can-

didatos em termos de legibilidade, organização, aderência às boas práticas de programação, relevância para o ensino de SD e facilidade de adaptação para fins educacionais.

Os resultados da pesquisa demonstram que a heurística desenvolvida é um recurso promissor para a seleção de candidatos para a criação de exemplos didáticos. A heurística é capaz de identificar trechos de código que demonstram melhorias na qualidade do *design* do código e que são relevantes para o ensino de ES. A ferramenta WEM (SILVA, 2023c) automatiza o processo de coleta de dados e metadados, facilitando a identificação de candidatos promissores.

Este trabalho abre portas para diversas pesquisas futuras, visando aprimorar a heurística desenvolvida e ampliar seu escopo de aplicação. Algumas sugestões para trabalhos futuros incluem:

- Desenvolver um conjunto de exemplos trabalhados com base nos candidatos selecionados pela heurística, incluindo explicações detalhadas e atividades para os alunos, com o foco no uso prático em sala de aula.
- Realizar um estudo de caso em sala de aula para avaliar o impacto do uso de exemplos trabalhados selecionados pela heurística no aprendizado de SD e ES.
- Investigar a aplicação da heurística em outros domínios de *software*, como Algoritmo e Estrutura de Dados e *machine learning*, considerando a integração do modelo de IA para expandir o escopo, desde que os repositórios atendam aos critérios estabelecidos na Seção 3.2.3.
- Ampliar a análise de refatorações ao nível de PR, utilizando a API do *GitHub* para extrair metadados adicionais, como os *labels* atribuídos a cada PR, e priorizando a análise de alterações significativas. Para isso, será importante considerar a proporção de alterações realizadas via PRs em relação aos *commits* totais do projeto, além de priorizar repositórios com um mínimo de *commits* diretos. Essa distinção é importante, pois PRs geralmente refletem um processo de revisão mais formal. Também será relevante diferenciar PRs de novas funcionalidades daqueles focados em manutenção corretiva, uma vez que essas categorias podem ter impactos distintos nas refatorações observadas. Embora essa abordagem adicione complexidade, espera-se que permita uma identificação mais precisa de refatorações significativas.
- Investigar a utilização de métricas dinâmicas de código, como as métricas IPC utilizadas no *DistFax* (FU; LIN; CAI, 2022), para complementar a análise de qualidade do código, permitindo uma visão mais detalhada do comportamento do software em execução.
- Explorar a possibilidade de aplicar a ferramenta *WEM* na replicação de estudos sobre Desenvolvimento Guiado por Testes (*Test-Driven Development (TDD)*), avaliando sua capacidade de identificar refatorações e padrões de código comuns nesse contexto.

- Melhorar a usabilidade da heurística e da ferramenta desenvolvida para uso por um público mais amplo, criando uma interface de usuário (*User Interface (UI)*) que facilite a interação com a ferramenta e a visualização dos resultados.
- Adicionar novas métricas de qualidade de refatoração, além de estudar a viabilidade da integração da heurística com outras ferramentas de análise de código, como o *SonarQube* (SONAR SOURCE, 2006), para obter uma visão mais abrangente da qualidade do código.

## REFERÊNCIAS

- ABAD, C. L.; ORTIZ-HOLGUIN, E.; BOZA, E. F. Have we reached consensus? an analysis of distributed systems syllabi. *In: Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*. New York, NY, USA: Association for Computing Machinery, 2021. (SIGCSE '21), p. 1082–1088. ISBN 9781450380621. Disponível em: <https://doi.org/10.1145/3408877.3432409>.
- ACM. **CS2023: ACM/IEEE-CS Computing Curricula 2023**. 2023. Acessado em 21 agosto 2024. Disponível em: <https://csed.acm.org/>.
- AI, G. **Google AI API Documentation (Python)**. 2023. <https://ai.google.dev/api?lang=python>. Acessado em 9 de setembro de 2024.
- AI, G. **Gemini API Documentation: Gemini 1.5 Flash**. 2024. <https://ai.google.dev/gemini-api/docs/models/gemini#gemini-1.5-flash>. Acessado em 9 de setembro de 2024.
- ANALYST, T. **Awesome Distributed Systems**. 2015. <https://github.com/theanalyst/awesome-distributed-systems>. Acessado em 9 de setembro de 2024.
- ANICHE, M. **Java code metrics calculator (CK)**. [S.I.], 2015. Available in <https://github.com/mauricioaniche/ck/>.
- Apache Software Foundation. **Zookeeper**. 2008. Programa de computador. Acessado em 19 setembro 2023. Disponível em: <https://zookeeper.apache.org/>.
- Apache Software Foundation. **Kafka**. 2011. Programa de computador. Acessado em 19 setembro 2023. Disponível em: <https://kafka.apache.org/>.
- ATKINSON, R. K. *et al.* Learning from examples: Instructional principles from the worked examples research. **Review of Educational Research**, SAGE, EUA, v. 70, n. 2, p. 181–214, jun. 2000. ISSN 0034-6543.
- BONETTI, T. P. *et al.* Students' perception of example-based learning in software modeling education. *In: Proceedings of the XXXVII Brazilian Symposium on Software Engineering*. New York, NY, EUA: ACM, 2023. p. 67–76. ISBN 9798400707872.
- BREWER, E. Cap twelve years later: How the "rules"have changed. **Computer**, v. 45, n. 2, p. 23–29, Feb 2012. ISSN 1558-0814.
- CHIDAMBER, S.; KEMERER, C. A metrics suite for object oriented design. **IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 476–493, 1994.
- Conventional Commits. **Conventional Commits**. 2023. Acessado em 31 de outubro de 2023. Disponível em: <https://www.conventionalcommits.org/en/v1.0.0/>.
- CORFUDB. **CorfuDB - GitHub Repository**. 2013. <https://github.com/CorfuDB/CorfuDB>. Acessado em 9 de setembro de 2024.
- COULOURIS, G. *et al.* **Distributed Systems: Concepts and Design**. 5th. ed. USA: Addison-Wesley Publishing Company, 2011. ISBN 0132143011.
- DETAILS, C. **CVE Details - Vulnerability Information**. 1999. <https://www.cvedetails.com>. Acessado em 9 de setembro de 2024.

- FOUNDATION, A. S. **Apache Software Foundation**. 1999. <https://www.apache.org/>. Acessado em 9 de setembro de 2024.
- FU, X.; LIN, B.; CAI, H. Distfax: A toolkit for measuring interprocess communications and quality of distributed systems. *In: 2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*. [S.I.: s.n.], 2022. p. 51–55.
- GHAFOOR, S. *et al.* Faculty development workshops for integrating PDC in early undergraduate curricula: An experience report. *In: QASEM, A.; THIRUVATHUKA, G. (Ed.). 11th Workshop on Education for High-Performance Computing (EduHPC-23)*. [s.n.], 2023. p. 1–8. Disponível em: [https://tcpp.cs.gsu.edu/curriculum/?q=system/files/ws\\_eduhpc105s1.pdf](https://tcpp.cs.gsu.edu/curriculum/?q=system/files/ws_eduhpc105s1.pdf).
- GITHUB. **Distributed Systems in Java - GitHub Topics**. 2024. <https://github.com/topics/distributed-systems?l=java>. Acessado em 9 de setembro de 2024.
- GITHUB, I. **GitHub - Code Hosting Platform**. 2008. <https://github.com>. Acessado em 9 de setembro de 2024.
- GOOGLE. **Gemini by Google**. 2023. <https://gemini.google.com>. Acessado em 9 de setembro de 2024.
- GOOGLE. **Google AI Research**. 2023. <https://ai.google.dev>. Acessado em 9 de setembro de 2024.
- GOSLING, J.; JOY, B.; STEELE, G. **The Java Language Specification**. Addison-Wesley, 1996. (Addison-Wesley Java series). ISBN 9780201634518. Disponível em: <https://books.google.com.br/books?id=O79QAAAAMAAJ>.
- HUNT, P. *et al.* Zookeeper: Wait-free coordination for internet-scale systems. *In: Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*. USA: USENIX Association, 2010. (USENIXATC'10), p. 11.
- JAVA, M. **Molecular Java - GitHub Repository**. 2017. <https://github.com/molecular-java/molecular-java>. Acessado em 9 de setembro de 2024.
- JESSE, K. **ManyRefactors4C Dataset on Hugging Face**. 2023. <https://huggingface.co/datasets/kevinjesse/ManyRefactors4C>. Acessado em 9 de setembro de 2024.
- JESSE, K. **RefactorBERT on Hugging Face**. 2023. <https://huggingface.co/kevinjesse/RefactorBERT>. Acessado em 9 de setembro de 2024.
- JESSE, K.; KUHMUENCH, C.; SAWANT, A. Refactorscore: Evaluating refactor prone code. *IEEE Transactions on Software Engineering*, v. 49, n. 11, p. 5008–5026, 2023.
- KAFKA, A. **Powered by**. 2023. Acessado em 3 de dezembro de 2023. Disponível em: <https://kafka.apache.org/powerd-by>.
- KREPS NEHA NARKHEDE, J. R. J. Apache kafka: Next generation distributed messaging system. *International Journal of Scientific and Engineering Research*, v. 3, n. 8, p. 1–4, 2010. Disponível em: <https://ijsetr.com/uploads/436215IJSETR3636-621.pdf>.
- MCQUAIGUE, M. *et al.* Data-driven discovery of anchor points for PDC content. *In: QASEM, A.; THIRUVATHUKA, G. (Ed.). 11th Workshop on Education for High-Performance Computing (EduHPC-23)*. [s.n.], 2023. p. 1–8. Disponível em: [https://tcpp.cs.gsu.edu/curriculum/?q=system/files/ws\\_eduhpc111s1.pdf](https://tcpp.cs.gsu.edu/curriculum/?q=system/files/ws_eduhpc111s1.pdf).

- MULDNER, K.; JENNINGS, J.; CHIARELLI, V. A review of worked examples in programming activities. **ACM Trans. Comput. Educ.**, Association for Computing Machinery, New York, NY, USA, v. 23, n. 1, dec 2022. Disponível em: <https://doi.org/10.1145/3560266>.
- NSF/IEEE-TCPP. **NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing**. 2024. Disponível em: <https://tcpp.cs.gsu.edu/curriculum/?q=home>.
- NSF/TCPP. **EduHiPC: Workshop on Education for High Performance Computing**. 2023. Disponível em: <https://tcpp.cs.gsu.edu/curriculum/?q=node/21242>.
- NSF/TCPP. **EduPar (NSF/TCPP Workshop on Parallel and Distributed Computing Education)**. 2023. Disponível em: <https://tcpp.cs.gsu.edu/curriculum/?q=node/21242>.
- OPENAI. **Introducing ChatGPT and Whisper APIs**. 2024. <https://openai.com/index/introducing-chatgpt-and-whisper-apis/>. Acessado em 9 de setembro de 2024.
- OPENAI. **OpenAI API Reference Documentation**. 2024. <https://platform.openai.com/docs/api-reference>. Acessado em 9 de setembro de 2024.
- PINTO, G. H. L. *et al.* Training software engineers using open-source software: The professors' perspective. In: **2017 IEEE 30th Conference on Software Engineering Education and Training**. [S.l.: s.n.], 2017. p. 117–121.
- RAJ, R. K.; KUMAR, A. N. Toward computer science curricular guidelines 2023 (cs2023). **ACM Inroads**, Association for Computing Machinery, New York, NY, USA, v. 13, n. 4, p. 22–25, nov 2022. ISSN 2153-2184. Disponível em: <https://doi.org/10.1145/3571092>.
- ROSSUM, G. van. **Python Programming Language**. 1991. Acessado em 17 de novembro de 2023. Disponível em: <https://www.python.org/>.
- SCALECUBE. **ScaleCube Services - GitHub Repository**. 2018. <https://github.com/scalecube/scalecube-services>. Acessado em 9 de setembro de 2024.
- SILVA, B. *et al.* Abordagem para seleção de exemplos trabalhados para engenharia de software do domínio de sistemas distribuídos. In: **Anais Estendidos do IV Simpósio Brasileiro de Educação em Computação**. Porto Alegre, RS, Brasil: SBC, 2024. p. 17–18. ISSN 0000-0000. Disponível em: [https://sol.sbc.org.br/index.php/educomp\\_estendido/article/view/29466](https://sol.sbc.org.br/index.php/educomp_estendido/article/view/29466).
- SILVA, B. F. da. **Code Metrics Generator: Python Script for generating the code metrics using CK metrics and PyDriller**. 2023. [https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/blob/main/PyDriller/code\\_metrics.py](https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/blob/main/PyDriller/code_metrics.py). Acessado em 4 de dezembro de 2023.
- SILVA, B. F. da. **Metrics Changes: Python Script for generating the metrics changes using PyDriller**. 2023. [https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/blob/main/PyDriller/metrics\\_changes.py](https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/blob/main/PyDriller/metrics_changes.py). Acessado em 4 de dezembro de 2023.
- SILVA, B. F. da. **Worked Example Miner (WEM): A Comprehensive Tool for Analyzing Java Repositories**. 2023. <https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/>. Acessado em 4 de dezembro de 2023.
- SILVA, B. F. da. **Gemini API Integration: Python Script for Associating Candidates with Significant Metric Improvements and Detected Refactoring Patterns to the Curriculum of Distributed Systems**. 2024. <https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/blob/main/Gemini/gemini.py>. Acessado em 23 de agosto de 2024.

- SILVA, B. F. da. **Inclusão da métrica *MethodInvocationCounter* no projeto CK**. 2024. <https://github.com/mauricioaniche/ck/pull/120>. Acessado em 9 de setembro de 2024.
- SILVA, B. F. da. **Repositories Picker: Python Script for Selecting Java Repositories from GitHub in the Domain of Distributed Systems**. 2024. [https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/blob/main/PyDriller/repositories\\_picker.py](https://github.com/BrenoFariasdaSilva/Worked-Example-Miner/blob/main/PyDriller/repositories_picker.py). Acessado em 23 de agosto de 2024.
- SKUDDER, B.; LUXTON-REILLY, A. Worked examples in computer science. In: **Proceedings of the Sixteenth Australasian Computing Education Conference**. Darlinghurst, Austrália: Australian Computer Society, 2014. v. 148, p. 59–64. ISBN 978-1-921770-31-9.
- SONAR SOURCE. **SonarQube - Code Quality and Security**. 2006. <https://www.sonarsource.com/products/sonarqube/>. Acessado em 9 de setembro de 2024.
- SPADINI, D.; ANICHE, M.; BACCHELLI, A. PyDriller: Python framework for mining software repositories. In: **Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018**. New York, New York, USA: ACM Press, 2018. p. 908–911. ISBN 9781450355735. Disponível em: <http://dl.acm.org/citation.cfm?doid=3236024.3264598>.
- STEEN, M. van; TANENBAUM, A. S. A brief introduction to distributed systems. **Computing**, v. 98, n. 10, p. 967–1009, Oct 2016. ISSN 1436-5057. Disponível em: <https://doi.org/10.1007/s00607-016-0508-7>.
- TONHÃO, S.; COLANZI, T.; STEINMACHER, I. Using real worked examples to aid software engineering teaching. In: RIBEIRO, M. et al. (Ed.). **35th Brazilian Symposium on Software Engineering (SBES 2021)**. New York, NY, EUA: ACM, 2023. p. 133–142. ISBN 978-1-4503-9061-3.
- TONHÃO, S. et al. Uma plataforma gamificada de desafios baseados em worked examples extraídos de projetos de software livre para o ensino de engenharia de software. In: **XVII Simpósio Brasileiro de Sistemas Colaborativos**. Porto Alegre, RS, Brasil: SBC, 2022. p. 33–38.
- TSANTALIS, N. et al. Accurate and efficient refactoring detection in commit history. In: **Proceedings of the 40th International Conference on Software Engineering**. New York, NY, USA: ACM, 2018. (ICSE '18), p. 483–494. ISBN 978-1-4503-5638-1. Disponível em: <http://doi.acm.org/10.1145/3180155.3180206>.
- VINKA, E. **Zookeeper Atomic Broadcast Protocol (ZAB) and implementation of Zookeeper**. 2018. Acessado em 10 de novembro de 2023. Disponível em: <https://www.cloudkarafka.com/blog/cloudkarafka-zab.html>.

## **APÊNDICES**

**APÊNDICE A – QUESTIONÁRIO DE VALIDAÇÃO DO CANDIDATO 1 PARA EXEMPLO  
DIDÁTICO EM ENGENHARIA DE SOFTWARE**

# Questionário Avaliação de Trechos de Código - Engenharia de Software - Candidato 1.

Este questionário tem como objetivo avaliar trechos de código de repositórios de software livre relacionados a Sistemas Distribuídos (SD), com foco em determinar seu valor educacional quanto a competências em engenharia de software. A intenção é identificar códigos que exemplos práticos que possam ser eficazmente utilizados em atividades didáticas em sala de aula, tornando-se exemplos trabalhados (ET) para auxiliar na explicação de conceitos de SD e que tenham uma implementação de qualidade, em especial quanto à manutenibilidade em engenharia de software.

Trecho de código: <https://github.com/CorfuDB/CorfuDB/commit/805ef812d278dbec2a6f3bba458114b1264ac009>

Classe Alvo: Orchestrator

---

\* Indicates required question

1. Quão bem estruturado está o trecho de código em termos de legibilidade e organização? \*

*Mark only one oval.*

- Muito Mal Estruturado
- Mal Estruturado
- Decente
- Bem Estruturado
- Muito Bem Estruturado

2. O código segue boas práticas de programação (ex.: nomeação de variáveis, modularidade, tratamento de erros)? \*

*Mark only one oval.*

Discordo plenamente

Discordo

Mediano

Concordo

Concordo plenamente

3. Este trecho de código pode ser reutilizado em outros projetos ou contextos? \*

*Mark only one oval.*

Nada Reutilizável

Pouco Reutilizável

Reutilizável

Bastante Reutilizável

Muito Reutilizável

4. O trecho de código é facilmente testável? \*

*Mark only one oval.*

Nada Testável

Pouco Testável

Testável

Bastante Testável

Muito Testável

5. O código está bem documentado, com comentários e explicações claras? \*

*Mark only one oval.*

- Nada Documentado
- Pouco Documentado
- Documentado
- Bastante Documentado
- Muito Documentado
- Other: \_\_\_\_\_

6. Que melhorias na documentação poderiam ser feitas para facilitar a compreensão do código? \*

---

7. Você considera este trecho de código um bom exemplo para ilustrar práticas \* de Engenharia de Software em um curso?

---

8. Que outras características de Engenharia de Software poderiam ser exploradas neste trecho de código para fins didáticos? \*

---

---

This content is neither created nor endorsed by Google.

Google Forms

**APÊNDICE B – QUESTIONÁRIO DE VALIDAÇÃO DO CANDIDATO 2 PARA EXEMPLO  
DIDÁTICO EM SISTEMAS DISTRIBUÍDOS**

# Questionário Avaliação de Trechos de Código - Sistemas Distribuídos - Candidato 2.

Este questionário tem como objetivo avaliar trechos de código de repositórios de software livre relacionados a Sistemas Distribuídos (SD), com foco em determinar seu valor educacional quanto a competências em sistemas distribuídos. A intenção é identificar códigos que exemplos práticos que possam ser eficazmente utilizados em atividades didáticas em sala de aula, tornando-se exemplos trabalhados (ET) para auxiliar na explicação de conceitos de SD e que tenham uma implementação de qualidade, em especial quanto à manutenibilidade em engenharia de software.

Trecho de código: <https://github.com/CorfuDB/CorfuDB/commit/ec64ef74891b32d341b2d3ec553d9e64605c87e4>

Classe Alvo: ManagementServer

---

\* Indicates required question

- Quais conceitos específicos de Sistemas Distribuídos são melhor ilustrados por este trecho de código, conforme o currículo NSF/IEEE-TCPP? \*

*Mark only one oval.*

- Concorrência e Paralelismo
- Modelos Cliente-Servidor e Peer-to-Peer
- Tarefas e Threads
- Sincronização
- Comunicação

- O exemplo facilita a compreensão de algum outro conceito teórico de Sistemas Distribuídos? \*

3. Quão fácil seria adaptar este código para ajudar na explicação do conceito abordado do ensino em Sistemas Distribuídos? \*

*Mark only one oval.*

- Muito Difícil
- Difícil
- Mediano
- Fácil
- Muito Fácil

4. (Opcional) Quais adaptações precisariam ser feitas para utilizar o código em um exemplo trabalhado?

---

5. Mais informações seriam necessárias para conseguir avaliar corretamente a \* efetividade do trecho de código apresentado? Se sim, quais?

---

---

---

---

6. Quão relevante é o trecho de código para o ensino de conceitos de Sistemas \* Distribuídos?

*Mark only one oval.*

- Nada Relevante
- Pouco Relevante
- Relevante
- Bastante Relevante
- Extremamente Relevante

7. Considerando todos os aspectos, você recomendaria o uso deste trecho de código em um curso de Sistemas Distribuídos? \*

---

---

---

---

---

---

---

This content is neither created nor endorsed by Google.

Google Forms