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

Uma abordagem para apoiar a seleção de repositórios de software

Guilherme Marques de Oliveira

JUIZ DE FORA
DEZEMBRO, 2024

Uma abordagem para apoiar a seleção de repositórios de software

Guilherme Marques de Oliveira

Universidade Federal de Juiz de Fora

Instituto de Ciências exatas

Departamento de Ciência da Computação

Bacharelado em Ciência da Computação

Orientador: Gleiph Ghiotto Lima de Menezes

JUIZ DE FORA

DEZEMBRO, 2024

Uma abordagem para apoiar a seleção de repositórios de software

Guilherme Marques de Oliveira

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

Aprovada por:

Gleiph Ghiotto Lima de Menezes
Prof. Dr. em Computação

André Luiz de Oliveira
Prof. Dr. em Ciência da Computação

Leonardo Vieira Dos Santos Reis
Prof. Dr. em Ciência da Computação

JUIZ DE FORA
12 DE DEZEMBRO, 2024

Resumo

Repositórios de software permitem a centralização do armazenamento de artefatos produzidos durante o desenvolvimento de um software. Distinguir esses repositórios de software entre aqueles que trarão dados relevantes daqueles que não trarão um valor final para a pesquisa é difícil. A proporção de ruído, como repositórios vazios, pode distorcer o estudo sobre uma amostra de repositórios e pode levar os pesquisadores a chegar a conclusões irreais e potencialmente imprecisas. Existem também limitações nas ferramentas presentes na literatura que auxiliam o processo de seleção de repositórios, por haver uma falta de um processo sistemático para escolha dos repositórios, inviabilizando a reprodutibilidade dos experimentos. Portanto, este trabalho implementa uma abordagem para auxiliar na seleção de repositórios de software para a área de Mineração de Repositórios de Software (MSR), resolvendo limitações de ferramentas presentes na literatura. Também foi conduzido um experimento replicando uma busca de repositórios do *GitHub* realizada por outro trabalho, e comparar as mudanças ocorridas entre os repositórios de ambas as pesquisas.

Palavras-chave: Repositórios de Software, Mineração de Repositórios de Software e Seleção de Repositórios de Software

Abstract

Software repositories make it possible to centralize the storage of artifacts produced during software development. Distinguishing these software repositories between those that will provide relevant data from those that will not contribute to the final value to research is difficult. The proportion of noise, such as empty repositories, can distort the study of a repository sample and potentially lead researchers to draw unrealistic and potentially inaccurate conclusions. There are also limitations in the tools found in the literature that help with the repository selection process, as there is a lack of a systematic process for choosing repositories, making it difficult to reproduce experiments. Therefore, this work implement an approach to assist in the selection of software repositories for the MSR field and solving limitations of tools in the literature.An experiment was also conducted replicating a search of *GitHub* repositories carried out by another work, and compare the changes that occurred between the repositories of both searches.

Keywords: Software Repositories, Mining Software Repositories, Selection of Software Repositories.

Agradecimentos

A tudo e a todos que me guiaram até aqui.

"Don't resist, accept, everything is connected".

Hideki Arai

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Abreviações

DCC Departamento de Ciência da Computação

UFJF Universidade Federal de Juiz de Fora

1 Introdução

1.1 Apresentação do tema

A mineração de repositórios de software (MSR) tem evoluído graças à abundante disponibilidade de dados presentes em repositórios, permitindo a análise e a evolução de sistemas de software por meio de técnicas de mineração dos dados históricos dos desenvolvedores (HASSAN, 2008). Ela tem sido amplamente utilizada para extrair informações valiosas de conjuntos de dados que antes eram apenas valores estáticos, transformando esses valores em informações que podem ser utilizadas para embasar decisões que impactarão diretamente na mudança e evolução dos sistemas de software. Por exemplo, repositórios de código-fonte podem ser analisados a fim de identificar características de conflitos em *merges* de software (GHIOTTO et al., 2018), bem como para predizer potenciais *bugs* nos códigos-fonte, evitando a introdução de possíveis defeitos (MOUSTAFA et al., 2018). Com a crescente disponibilidade de dados e a evolução destas técnicas, a expectativa é que a mineração de repositórios continue a se desenvolver e fornecer informações cada vez mais valiosas para a área de desenvolvimento de software. Para isso, é necessário haver a possibilidade de uma seleção dos repositórios, dentre os milhões disponíveis, de forma organizada, mantendo o histórico de decisões, para obter resultados alinhados com o objeto de pesquisa e permitir uma filtragem dos repositórios que gere informações valiosas.

1.2 Contextualização

Durante o ciclo de vida de desenvolvimento de um software, vários artefatos (*e.g.*, código-fonte, documentação) são gerados e manipulados. O acompanhamento da evolução desses artefatos pode ser registrado em sistemas de controle de versões, possibilitando que as equipes de desenvolvimento rastreiem as alterações realizadas por cada desenvolvedor (CHACON; STRAUB, 2010) e gerenciem o histórico de desenvolvimento dos artefatos armazenados nos repositórios de software (BERCZUK; APPLETON, 2002). Através des-

ses repositórios, pesquisadores podem encontrar e investigar grandes volumes de dados brutos.

Há diversos tipos de repositórios disponíveis na literatura, como os repositórios de implantação, que registram informações sobre a execução da implantação de um projeto de software (HASSAN, 2008), e os repositórios de *bugs*, também chamados de *Issue tracking*, que armazenam o histórico de resolução de problemas relatados pelos usuários e permite o compartilhamento dessas informações com a equipe de desenvolvimento, dando uma visão geral do estado atual do software presente no repositório (JANÁK, 2009). Estes são apenas alguns exemplos de repositórios de software que podem ser utilizados para obtenção de informações pela mineração de repositórios (HASSAN, 2008).

Existem plataformas que permitem o armazenamento e gerenciamento de repositórios de software, como o *GitLab*, *Bitbucket* e o *GitHub*. O *GitHub*, mais popular dentre eles, armazena mais de 400 milhões de repositórios públicos contendo artefatos de software [1], repositórios esses baseados no sistema de controle de Versão *Git*. Plataformas como o *GitHub* têm o potencial de fornecer aos pesquisadores diversos artefatos de software, através de seus repositórios armazenados em um mesmo ambiente, tornando a investigação científica mais acessível (ALMARZOUQ; ALZAIDAN; ALDALLAL, 2020).

1.3 Descrição do problema

Devido ao grande número de repositórios com artefatos de software distribuídos digitalmente em plataformas de hospedagem de código-fonte como o *GitHub*, tornou-se complexo para os pesquisadores da área de mineração de repositórios selecionarem quais repositórios podem ser candidatos para trazer dados brutos que possam ser convertidos em informações valiosas (MUNAIAH et al., 2017). Portanto, um dos desafios da mineração de repositórios é a seleção dos repositórios que serão utilizados em uma pesquisa.

Na literatura, existem diversas ferramentas que auxiliam os pesquisadores na seleção dos repositórios mais apropriados para a mineração de repositórios, como a *GitHub API* e o *GitHub Search*. Essas ferramentas permitem que os usuários realizem filtragens baseadas em características específicas dos repositórios, como quantidade de *estrelas* e de

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

forks, para, assim, obterem repositórios alinhados com seus objetivos de pesquisa. Ao utilizar essas ferramentas é possível reduzir a complexidade na seleção dos repositórios, tornando-as importantes instrumentos para os pesquisadores da área de mineração de repositórios de software. Porém, é possível também encontrar limitações em tais ferramentas que podem afetar a seleção de repositórios do usuário, como a falta de alguns metadados na pesquisa, e o limite máximos de repositórios por requisição.

1.4 Justificativa e motivação

Existem diversas ferramentas que podem auxiliar na seleção de repositórios de software, incluindo as oferecidas por plataformas de hospedagem de repositórios, como o *GitHub*. O *GitHub* oferece a busca avançada e a *GitHub API*, que permitem a seleção de repositórios de software. Além das ferramentas oficiais, existem outras opções, como o *GitHub Search*, que utiliza a *GitHub API* e um *WebCrawler* para obter repositórios e armazená-los em um banco de dados próprio (DABIC; AGHAJANI; BAVOTA, 2021).

No entanto, essas ferramentas possuem certas limitações que podem prejudicar a experiência do usuário. Por exemplo, a *GitHub API* retorna no máximo 100 repositórios por requisição e no máximo 1000 repositórios se houver a paginação das requisições. Essa limitação pode exigir que o usuário tenha que alterar seus parâmetros de busca diversas vezes, caso a quantidade de repositórios retornados for superior a 1000, tornando mais complexo o processo de obtenção destes repositórios. Outra limitação é a falta de campos que podem ser utilizados como parâmetros de pesquisa, como os campos de quantidade de *commits* e *watchers*, a falta destes campos limita a pesquisa por repositórios do usuário.

Com relação ao *GitHub Search*, uma das limitações está atrelada as linguagens disponíveis. Ele seleciona apenas os repositórios que possuem pelo menos uma dentre 10 linguagens pré-definidas pela ferramenta, limitando a possibilidade do usuário obter repositórios que possuam códigos-fontes em uma linguagem fora do escopo da ferramenta (DABIC; AGHAJANI; BAVOTA, 2021). Portanto, é importante que o usuário considere as limitações de cada ferramenta, pois assim, ele pode escolher a mais adequada para as suas necessidades.

Devido a essas limitações encontradas, a abordagem desenvolvida por este traba-

lho é motivada a reparar algumas dessas limitações presentes em outras ferramentas, como as linguagens pré-definidas do *GitHub Search* e o retorno de no máximo 1000 repositórios por requisições paginadas da *GitHub API*.

1.5 Objetivos

Com o intuito de contribuir para a área de mineração de repositórios, facilitando o acesso aos repositórios que serão utilizados pelos pesquisadores da área, este trabalho possui como objetivo principal superar as limitações impostas por ferramentas que permitem a seleção de repositórios de software, como o limite de 1000 repositórios e falta de atributos da *GitHub API* e as 10 linguagens pré-definidas do *GitHub Search*.

Como objetivos específicos, a pesquisa oferta uma ferramenta semi-automatizada que auxilie na escolha dos repositórios mais adequados para sua pesquisa, permitindo que o usuário selecione as características desejadas dos repositórios e os obtenha através de uma busca. De início foi realizado uma análise da literatura a respeito de softwares que apoiam a seleção de repositórios, analisando as características e limitações que essas ferramentas apresentam. Após isso, foi definido os requisitos da a abordagem deste trabalho, para contemplar a seleção de repositórios e superar as limitações das outras ferramentas. Com a abordagem estabelecida, foi realizado uma implementação dela, e por fim, feito um experimento para validar essa implementação.

Como nova funcionalidade, a abordagem permite que o usuário possa, em sua seleção, estabelecer critérios que o possibilitem incluir ou excluir, dentre os repositórios obtidos, aqueles que não estiverem alinhados com os seus ideais de seleção. Por fim, foi realizada uma validação da abordagem, replicando uma busca sob os repositórios do *GitHub* realizada anteriormente na literatura, com o intuito de responder às seguintes questões de pesquisa levantadas:

QP1 — O conjunto de repositórios do experimento base contém os mesmos repositórios do experimento derivado?

QP2 — Quais foram os repositórios que não estavam no experimento base que entraram no experimento derivado? Quais os motivos?

QP3 — Quais foram os repositórios que estavam no experimento base

que não se encontram no experimento derivado? Quais os motivos?

QP4 — Houve repositórios que sofreram modificações em sua URL do experimento base para o experimento derivado? Quais os motivos?

Estas questões visam descobrir se houve mudanças nos repositórios resultantes de uma pesquisa para outra, e caso tenham ocorrido mudanças, quais repositórios saíram e entraram entre os resultados das pesquisas e o porquê.

1.6 Organização deste trabalho

Além desta Introdução, este trabalho está organizado em seis capítulos e um Apêndice. O Capítulo 2 apresenta os conceitos necessários para o entendimento do trabalho proposto. O Capítulo 3 apresenta uma comparação dos trabalhos relacionados obtidos na revisão sistemática da literatura deste trabalho. O Capítulo 4 apresenta a ideia da abordagem que visa auxiliar pesquisadores da área de mineração de repositórios a selecionarem repositórios alinhados com sua pesquisa. O Capítulo 5 é destinado à apresentação de uma implementação da abordagem proposta. O Capítulo 6 descreve um experimento realizado, replicando de um trabalho anterior, uma pesquisa no *GitHub*. O Capítulo 7 compila os resultados e apresenta os trabalhos futuros. Por fim, o Apêndice A destaca como foi feita a revisão sistemática da literatura para buscar por trabalhos relacionados a seleção de repositórios de software.

2 Fundamentação teórica

Neste capítulo, são apresentados os conceitos que apoiam o entendimento deste Trabalho de Conclusão de Curso. A Seção 2.1 apresenta os conceitos relacionados aos Sistemas de controle de versão e seus tipos. A Seção 2.2 descreve o que são as plataformas de hospedagem de repositórios, discute quais são as plataformas presentes na literatura e quais os metadados presentes nos repositórios dessas plataformas. A Seção 2.3 apresenta o campo de pesquisas chamado Mineração de Repositórios de Software, cita as evoluções obtidas, e conceitua modelos de repositórios de software que podem ser objetos de pesquisa pela área. A Seção 2.4 apresenta uma descrição das versões v3 e v4 da API do *GitHub*, destacando suas diferenças e disserta sobre algumas limitações que podem ser encontradas ao utilizar a API v4 do *GitHub*. A Seção 2.5 descreve o que é a área da seleção de repositórios de software e as dificuldades encontradas. Por fim, a Seção 2.6 realiza um apanhado geral das seções anteriores.

2.1 Sistemas de controle de versão (SCV)

Os Sistemas de Controle de Versão (SCV) são essenciais no desenvolvimento de software, por permitirem o gerenciamento eficiente do histórico de desenvolvimento de artefatos de software (BERCZUK; APPLETON, 2002). Esses artefatos incluem arquivos como documentações e códigos-fonte, produzidos durante o processo de desenvolvimento de software. Com o uso de um SCV, é possível acompanhar as alterações feitas em artefatos ao longo do tempo, comparar diferentes versões e restaurar uma versão anterior, se necessário. Isso torna o processo de desenvolvimento de software mais organizado e colaborativo.

Algumas informações que os SCVs armazenam através do histórico de evolução dos repositórios de software são o número de alterações que um repositório sofreu, a data de realização destas alterações e quais destas alterações podem ter sido feitas para corrigir um determinado *bug*, e quais e quantos contribuidores estão atrelados a um dado repositório (CASALNUOVO et al., 2017).

Existem dois tipos principais de SCV: os centralizados e os distribuídos. Os SCV centralizados, como o *Subversion* (COLLINS-SUSSMAN; FITZPATRICK; PILATO, 2003), seguem uma topologia em estrela, no qual um único repositório central é utilizado, mas cada desenvolvedor tem sua própria cópia de trabalho. Os SCV distribuídos, como o *Mercurial* (O'SULLIVAN, 2009) e o *Git* (CHACON; STRAUB, 2010), possuem repositórios autônomos e independentes, um para cada desenvolvedor, com cada repositório contendo sua própria área de trabalho acoplada, permitindo uma maior flexibilidade e independência (ZOLKIFLI; NGAH; DERAMAN, 2018).

Nos SCV centralizados, a comunicação entre as máquinas passa pelo repositório central e o ciclo de trabalho pode ser dividido em quatro atividades: o *checkout*, que busca uma versão do repositório e a traz para a área de trabalho do desenvolvedor; as alterações realizadas pelo desenvolvedor por meio de adições, remoções ou edições de informações nos artefatos de software; o *commit*, que permite ao desenvolvedor enviar essas mudanças para o repositório e criar uma nova versão, e o *update*, que traz as mudanças do repositório para a área de trabalho do desenvolvedor (OTTE, 2009). A topologia dos SCV centralizados é representada pela Figura 2.1. Ela demonstra que, mediante um repositório central, tanto o usuário A quanto o usuário B conseguem obter os artefatos do repositório para sua máquina local através do método de *checkout*, e, se necessário, levar as atualizações para o repositório do servidor central através do método *commit*, tais mudanças poderão ser obtidas por outros usuários, e por fim, podem atualizar os artefatos desatualizados com relação ao repositório central, através do método *update*.

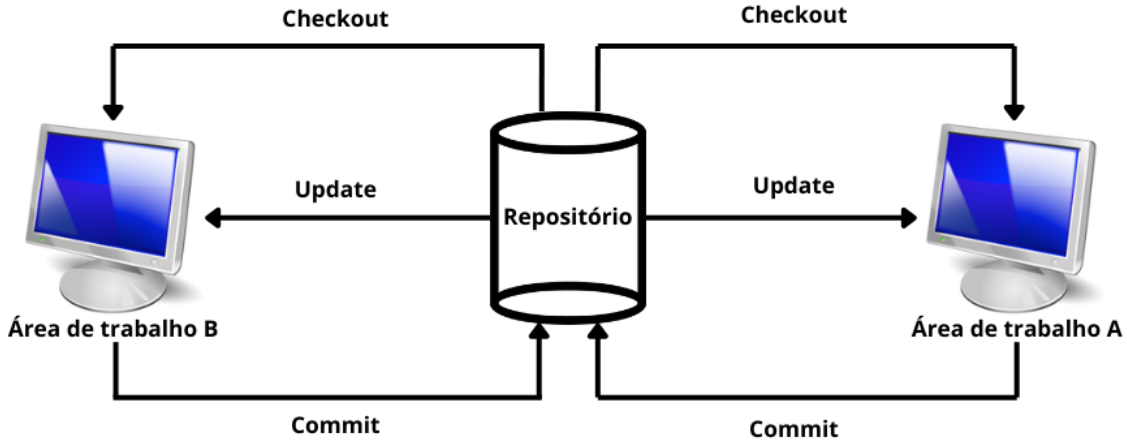

Figura 2.1: Sistema de controle de versão centralizado, adaptado de Ghiotto et al. (2018)

Os SCVs distribuídos possuem um ciclo de trabalho parecido dos SCVs centralizados quando se fala sobre os comandos de *checkout* e *commit*. No entanto, tais SCVs possuem a operação de *clone*, que realiza uma cópia completa do repositório para a área de trabalho do desenvolvedor e não só uma configuração específica, como ocorre com os SCV centralizados. (CHACON; STRAUB, 2010). Os repositórios de SCV distribuídos podem receber e enviar versões para os demais sem a necessidade de uma topologia pré-definida; eles realizam tal comunicação através das operações de *pull* e *push*. O comando *pull* atualiza o repositório local com as alterações feitas no repositório de origem e o *push* envia as alterações realizadas localmente para o repositório de destino, permitindo uma colaboração descentralizada entre os desenvolvedores (OTTE, 2009).

A topologia dos SCV distribuídos é representada pela Figura 2.2. Ela mostra que tanto o usuário A quanto o B possuem seu próprio repositório local em suas áreas de trabalho e que é possível realizar a comunicação tanto entre os repositórios locais quanto com o repositório do servidor central.

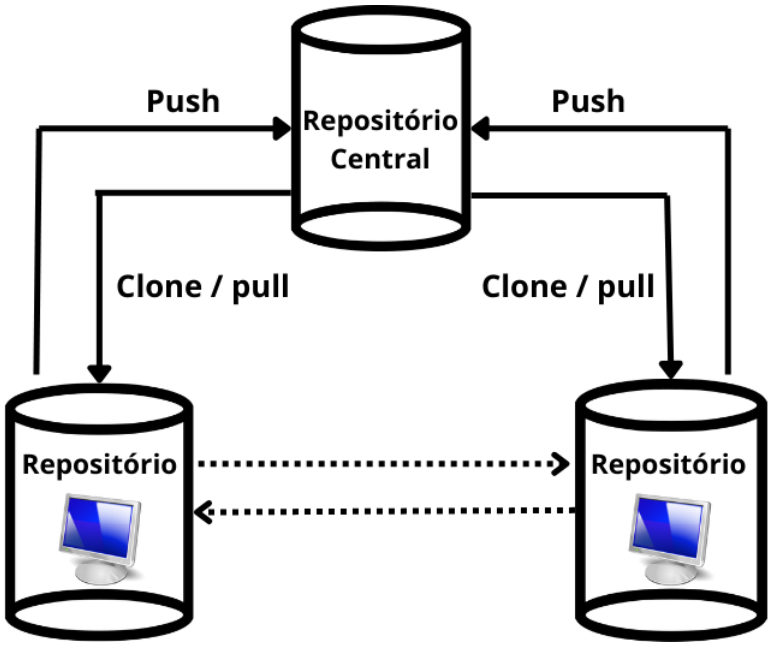

Figura 2.2: Sistema de controle de versão distribuído, adaptado de Ghiotto et al. (2018)

2.2 Hospedagem de repositórios

Na década de 2000, houve a popularização das plataformas de hospedagem de repositórios de software. Essas plataformas hospedam diversos repositórios que armazenam artefatos produzidos durante o desenvolvimento de software, permitindo o trabalho colaborativo entre os desenvolvedores (HASSAN, 2008).

Uma das maiores e mais populares plataformas de hospedagem de repositórios da atualidade é o *GitHub*, possuindo mais de 100 milhões de usuários e mais de 420 milhões de repositórios em março de 2024[1]. O *GitHub* hospeda os repositórios do SCV *Git* e disponibiliza metadados para tais repositórios, auxiliando na caracterização dos repositórios.

Alguns tipos de metadados que podem ser encontrados no *GitHub* são as *estrelas*, *commits*, *forks* e *watchers*. As *estrelas* permitem aos usuários demonstrarem interesse pelos repositórios, permitindo a medição da popularidade de um dado repositório (WEBER; LUO, 2015). Os *commits* armazenam as contribuições realizadas sobre um dado repositório (ALALI; KAGDI; MALETIC, 2008). Os *forks* permitem a bifurcação de repositórios para a conta de outro usuário. Criar uma bifurcação é produzir uma cópia pessoal do projeto, permitindo que as alterações de um *fork* não gere efeito colateral no repositório original e que contribuições possam ser incorporadas no projeto original, propiciando possíveis atualizações e modificações ao repositório original(JIANG et al., 2017). Os *watchers* foram planejados para serem usados por aqueles que desejassem receber notificações sobre novas atividades realizadas em um dado repositório (KALLIAMVAKOU et al., 2016).

Esses metadados estão presentes nos projetos e podem ser conferidos acessando a página de um repositório na plataforma do *GitHub*. Como um exemplo, obteve-se o repositório com mais quantidade de estrelas em abril de 2024, o *freeCodeCamp*[2]. Na Figura 2.3, é perceptível que certos metadados são visíveis, no qual mais de 387.000 pessoas deram estrelas (metadado de estrelas) e o repositório sofreu ramificação mais de 35.300 vezes (metadado de *forks*).

[1] ⟨https://github.com/about⟩
[2] ⟨https://github.com/freeCodeCamp/freeCodeCamp⟩

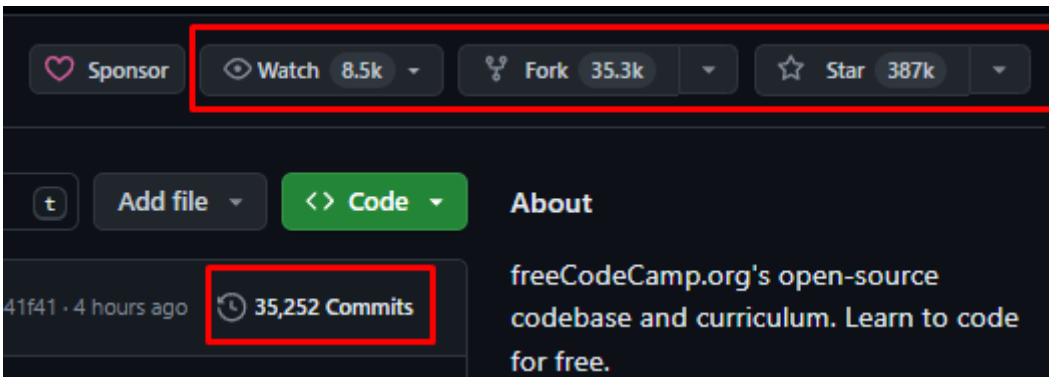

Figura 2.3: Metadados encontrados no repositório *freeCodeCamp*

Outras ferramentas de hospedagem podem ser encontradas na literatura. O *GitLab*, por exemplo, é uma plataforma de código aberto que oferece um fácil gerenciamento de repositórios *Git*. Ele oferece o gerenciamento de acesso a projetos, uma *wiki* para documentação, e possibilidade de definição de metas e *issues* para gerenciamento de tarefas (LEHTOVIRTA, 2017).

Por fim, os sistemas de hospedagem de repositórios possibilitam que o desenvolvimento de software esteja em constante evolução. À medida que mais softwares migram para o modelo de código aberto (OSS), que se baseia na colaboração da comunidade, o crescimento contínuo dessa colaboração impulsiona avanços em diversas áreas, como governo e educação (ALAMER; ALYAHYA, 2017).

2.3 Mineração de Repositórios de Software

A Mineração de Repositórios de Software (MSR) analisa a evolução do software para obtenção de informações importantes sobre sistemas e projetos. Ela é realizada por meio de técnicas de mineração dos dados históricos dos desenvolvedores nos repositórios de software. O campo da MSR tem crescido graças ao advento dos projetos de código aberto e ao crescimento de plataformas como o *GitHub*, facilitando o acesso a repositórios de grandes projetos (HASSAN, 2008).

A partir da MSR, é possível extrair informações com o intuito de compreender quais práticas e metodologias de desenvolvimento de softwares são mais eficientes (HASSAN, 2008). Visto que há uma grande fonte de dados do estado atual e de todo o histórico

de alterações dos artefatos, é possível desenvolver diversas pesquisas e ferramentas para auxiliar no desenvolvimento de software.

No estudo publicado por D'Ambros et al. (2008), são mencionados vários assuntos pertinentes ao campo de análise de repositórios de software, incluindo a análise do esforço dos desenvolvedores e das redes sociais. Nesse contexto, examina-se o esforço da equipe de programadores na manutenção e evolução do software, bem como sua comunicação interna. Outro tópico analisado no estudo é o impacto e propagação da mudança em um software, em que é avaliado o impacto da adição de um novo recurso ou a alteração de um recurso existente, como adição ou alteração de linhas no código-fonte.

Segundo Hassan (2008), alguns tipos de repositórios de software que podem ser objetos de estudo são os repositórios de controle de código-fonte como o *Subversion*, que registram o histórico de desenvolvimento de um projeto e rastreiam todas as mudanças no código-fonte juntamente com metadados como o nome do desenvolvedor que fez a mudança e o horário em que a mudança foi feita. Outro modelo são os repositórios de *bugs*, que rastreiam o histórico de resolução de relatórios de *bugs* ou solicitações de recursos relatados por usuários e desenvolvedores de grandes projetos de software. *Bugzilla* e *Jira* são exemplos de repositórios de *bugs*.

Segundo Kagdi, Collard e Maletic (2007), a mineração de repositórios pode ser vista por meio de uma taxonomia dividida em 4 categorias que representam uma grande parcela das investigações realizadas pela mineração de repositórios. Tais categorias são:

- tipos de repositórios que serão minerados (o quê): se refere a quais repositórios e quais características presentes em tais repositórios serão utilizadas no momento da mineração de repositórios. Algumas das características que podem ser mineradas são os artefatos de software (*e.g.*, código-fonte, documentação) e os metadados presentes nos repositórios, como as mensagens presentes nos *commits*;

- o propósito (porque): se refere ao objetivo da mineração de repositórios, como identificação de *bugs*, padrões de uso ou características do processo de desenvolvimento;

- a metodologia adotada (como): refere-se às técnicas utilizadas na Mineração de Repositórios para responder a questionamentos de pesquisa. Algumas dessas aborda-

gens incluem a obtenção de métricas de complexidade de um software, computando informações extraídas de diferentes versões de um sistema de software, e a investigação dos motivos que levam um sistema de software a evoluir de uma versão para outra, utilizando códigos-fonte e documentos do repositório na análise; e

- o método de avaliação (qualidade): se refere as métricas de avaliação dos resultados obtidos, como a métrica de precisão, que visa descobrir quanto da informação obtida é relevante para os resultados desejados.

Nesse contexto, abordagens como a *Github Search* podem ser utilizadas para apoiar a característica de categoria de tipos de repositórios, pois uma abordagem como essa permite a semi-automatização do processo de seleção de repositórios de software pelo usuário, dando-lhe a oportunidade de decidir quais características e metadados ele busca nos repositórios. Segundo Kagdi, Collard e Maletic (2007), os metadados presentes nos repositórios constituem uma fonte valiosa de informações nos contextos de mudanças de software, podendo ser analisados independentemente ou com outros dados, como o código-fonte.

Existem diversos dados que podem ser explorados a partir de repositórios abertos, como número de alterações realizadas e quem realizou as alterações. Esses dados podem ser filtrados para a obtenção de repositórios que tenham características semelhantes (CASALNUOVO et al., 2017). Através dessas características, é possível identificar repositórios com dados similares e, por meio de ferramentas como a *GitHub API*, é possível que pesquisadores achem repositórios cujas características sejam relevantes para suas pesquisas e, assim, extraiam informações através da mineração de repositórios. Dessa forma, a seleção de repositórios de software pode ajudar a encontrar repositórios concernentes às ideias dos usuários.

2.4 *Github API*

O *GitHub* fornece acesso à sua base de dados através da *GitHub API*, utilizada por ferramentas na literatura para obter repositórios do *GitHub*, como o *GitHub Search* e a abordagem promovida aqui. Essa API tem as formas de *REST (API v3)* e *GraphQl (API v4)*.

A API v3 é uma API REST na qual um *Endpoint* é definido por uma URL e uma lista de parâmetros. Por exemplo, na *GitHub API v3*, o seguinte *Endpoint* pode ser acessado.

$$GET/search/repositories?q = estrelas :> 100$$

Uma requisição neste *Endpoint* retorna todos os atributos dos repositórios do *GitHub* com mais de 100 estrelas, fazendo com que o usuário não possa decidir quais informações dos repositórios serão retornadas. Nesse aspecto, a *API v4* apresenta vantagem, pois uma chamada do *GraphQL* pode ser equivalente a várias chamadas *REST*, permitindo que o usuário possa realizar uma requisição mais específica, usando como parâmetro de busca apenas os dados dos repositórios que ele deseja que sejam retornados (BRITO; VALENTE, 2020).

A *API v4* opera de maneira similar à busca avançada do *GitHub*, mas retorna os dados no formato *JSON*. Ela ainda enfrenta, no entanto, limitações semelhantes às da busca avançada, como o limite de retorno de até 1.000 repositórios. Além disso, não é possível realizar buscas por certos metadados dos repositórios, como a quantidade de *commits*, número de *watchers* ou por múltiplas linguagens simultaneamente.

Para retratar o problema de máximo de 1000 repositórios, suponha um cenário no qual o usuário deseja buscar todos os repositórios com mais de 1000 *estrelas*. Serão encontrados mais de 40.000 repositórios com tal característica, mas a *API* apenas retornará 1.000 destes 40.000 repositórios. Isso ocorre devido à limitação de no máximo 1000 repositórios por requisição da *GitHub API* (DABIC; AGHAJANI; BAVOTA, 2021).

Já para a limitação de ausência de certos metadados na busca, pode-se pensar em um cenário no qual o usuário deseja todos os repositórios que contenham mais de 4.000 *commits* e que tenham código-fonte escritos nas linguagens *Java* e *JavaScript*. Com tais campos, a *GitHub API* não atenderia a tal solicitação. Isso ocorre porque não é possível utilizar como parâmetro de busca a quantidade de *commits* e nem simultaneamente 2 ou mais linguagens presentes nos artefatos de software dos repositórios (GITHUB, 2022).

2.5 Seleção de repositórios de software

A seleção de repositórios é uma das primeiras etapas para os estudos na área de mineração de repositórios de software. Ela consiste em selecionar repositórios de software através dos dados, com intuito de analisar e obter respostas às questões de pesquisa de interesse (DABIC; AGHAJANI; BAVOTA, 2021). Tal passo é crucial para garantir que os projetos selecionados resultem em dados úteis para a mineração de repositórios. Para realizar tal seleção e obter os repositórios, desenvolvedores podem utilizar ferramentas como a *GitHub API* e o *GitHub Search*. Porém, conforme descrito na Seção 2.4, as ferramentas tradicionais apresentam limitações, como na *GitHub API*, em que há um limite na quantidade de repositórios retornados por uma busca e a restrição quanto a que atributos podem ser selecionados no momento de seleção dos repositórios.

Segundo Munaiah et al. (2017), existe uma dificuldade em separar repositórios de software que trarão dados relevantes de repositórios que não trarão um valor final para a pesquisa, pois a proporção de ruído, como repositórios vazios, pode distorcer o estudo sobre uma amostra de repositórios e pode levar os pesquisadores a chegar a conclusões irreais e potencialmente imprecisas. Tais ruídos podem ser percebidos no estudo de Kalliamvakou et al. (2016), em que se obteve um total de 434 repositórios do *GitHub* e percebeu-se que apenas 275 repositórios armazenavam o desenvolvimento de um software, enquanto os outros 159 repositórios armazenavam experimentos, trabalhos acadêmicos ou estavam vazios.

O estudo de Munaiah et al. (2017), cita como uma boa métrica a quantidade de *estrelas* presentes em um repositório, que pode ser utilizada para identificar repositórios com dados potencialmente relevantes para uma pesquisa. A justificativa seria que

repositórios com uma abundância de *estrelas* seriam repositórios populares e conteriam softwares pelos quais as pessoas teriam interesse. O estudo avaliou 1.857.423 repositórios do *GitHub*, e chegou a conclusão de que há uma alta probabilidade de um repositório armazenar um sistema de software caso ele possua pelo menos 1.123 *estrelas*.

Através dessas informações, é perceptível que há uma importância da seleção de repositórios de software e das abordagens que apoiam essa área, pois desenvolvimentos nessa área facilitam a obtenção de repositórios com dados vantajosos.

2.6 Considerações finais

Neste capítulo, apresenta-se o conceito de sistemas de controle de versões, ferramentas essenciais para o gerenciamento dos repositórios necessários para a seleção de repositórios de software. Posteriormente, o que são as plataformas de hospedagem de repositórios, alguns exemplos e quais informações dos repositórios essas plataformas hospedam. Conceituou-se a área de mineração de repositórios, quais benefícios essa área promove e como ferramentas de seleção de repositórios de software podem auxiliar tal área. Depois, demonstrou-se quais modelos de *API* o *GitHub* oferece e quais limitações é possível encontrar nestas *API* ao tentar realizar buscas que facilitem a seleção de repositórios. Por fim, apresentou-se o campo de Seleção de repositórios de software e as dificuldades enfrentadas ao buscar por repositórios de software que trarão valor a uma pesquisa.

3 O estado da arte em Seleção de Repositórios de Software

Esse capítulo é destinado à comparação dos trabalhos relacionados obtidos na revisão sistemática da literatura, considerando as abordagens sobre os principais assuntos a serem explorados neste trabalho. Um resumo mais detalhado de cada um destes trabalhos pode ser encontrado no Apêndice A.

O artigo de Reza, Badreddin e Rahad (2020) apresenta a ferramenta *ModelMine* utilizada para minerar repositórios de código aberto no *GitHub* e *GitLab*, dividida em seis etapas: indexação, paginação, redução de consulta, proteção de requisições, visualização de dados e ranqueamento. A ferramenta foi comparada ao *PyDriller* em termos de desempenho e usabilidade, mas tem a limitação de não conseguir superar o limite de 1000 repositórios da *GitHub API*.

O artigo de Romano et al. (2021) apresenta a ferramenta *G-Repo*, criada para auxiliar na criação de conjuntos de repositórios para pesquisas em mineração de repositórios. Ela utiliza a *GitHub API* para buscar repositórios, superando a limitação de 1000 resultados por pesquisa. A ferramenta também detecta idiomas de repositórios, com uma validação mostrando 98% de concordância entre a ferramenta e avaliadores humanos ao identificar repositórios em inglês. Contudo, a ferramenta enfrenta a limitação de não filtrar repositórios que não contenham software.

O artigo de Lefeuvre et al. (2023) trata da criação de conjuntos personalizados de dados de software usando a plataforma *Software Heritage*, que armazena milhões de projetos de código aberto. A proposta é desenvolver um processo de extração genérico e reprodutível de dados, como código-fonte e bug reports, utilizando uma "impressão digital" composta por critérios e um carimbo de data e hora para garantir a reprodutibilidade. A principal limitação é o alto consumo de recursos para manter os dados atualizados, devido às restrições de *APIs* como a do *GitHub*, que limitam as requisições.

O artigo de Do et al. (2020) aborda a mineração de metadados de repositórios,

como *commits* e *forks*, em plataformas como *GitHub* e outras hospedagens *Git*. Utilizando o *Software Heritage* para armazenar essas informações, os autores desenvolvem métodos para detectar *forks* e duplicações de repositórios, verificando se compartilham o mesmo diretório de origem e conjunto de revisões. Eles investigam a correlação entre padrões de *forks*, saúde e riscos do software e indicadores de sucesso, concluindo que, embora haja correlação, análises mais profundas são necessárias para resultados concretos.

O Artigo de Sokol, Aniche e Gerosa (2013) apresenta o *MetricMiner*, uma aplicação web projetada para ajudar pesquisadores da área mineração de repositórios, facilitando o armazenamento e a análise de dados. A ferramenta oferece funcionalidades de cálculo de métricas, extração de dados e acesso a repositórios compartilhados por usuários, economizando tempo e recursos. Em comparação com outras ferramentas como *Sonar* e *Boa*, o *MetricMiner* se destaca pela variedade de funcionalidades, como suporte a repositórios *Git* e *SVN*, mas tem limitações, como a falta de gráficos para visualização de dados.

A abordagem *GitHub Search* proposta por Dabic, Aghajani e Bavota (2021), ajuda desenvolvedores a selecionar repositórios para pesquisas em mineração de software. A ferramenta permite a busca de repositórios no *GitHub* com base em 25 características, como quantidade de estrelas e data de criação. Com mais de 1.200.000 repositórios armazenados e um minerador que mantém os dados atualizados, a ferramenta oferece repositórios constantemente renovados. Porém, uma limitação é o uso de um *web crawler* para acessar informações de colaboradores dos repositórios, o que a torna dependente do layout atual do *GitHub*, podendo afetar a precisão caso o layout seja alterado.

O trabalho de Gousios (2013) descreve o *GHTorrent*, uma ferramenta que coleta e armazena dados de repositórios do *GitHub* para pesquisas, superando as limitações da *GitHub API* com o uso de *caching* e armazenamento em banco de dados *MongoDB*. A ferramenta também permite acesso paralelo por múltiplos usuários. No entanto, enfrenta limitações como dados desatualizados e mudanças nos formatos de dados do GitHub. Apesar dessas limitações, o *GHTorrent* facilita a replicação de estudos, oferecendo um conjunto de dados homogêneos, sendo considerado valioso para pesquisas.

O artigo de Cosentino, Luis e Cabot (2016) analisa 93 pesquisas sobre mineração de repositórios no *GitHub*, focando em métodos, conjuntos de dados e limitações. O

trabalho identificou problemas como baixa replicabilidade, amostragem fraca e falta de diversidade metodológica. A maioria das pesquisas (75%) usa métodos empíricos com metadados do *GitHub*, e as ferramentas mais comuns para obter os dados são a *GitHub API* (39,8%) e *GHTorrent* (34,4%). As limitações incluem dados desatualizados e dificuldades de generalização. O artigo recomenda melhorar a amostragem, compartilhar repositórios e atualizar os dados para aumentar a qualidade e replicabilidade das pesquisas.

O artigo de (MOMBACH; VALENTE, 2018) compara três ferramentas para obter dados de repositórios: *GitHub REST API*, *GitHub Archive* e *GHTorrent*. A *GitHub API* fornece dados atualizados em tempo real, enquanto o *GitHub Archive* atualiza a cada hora e o *GHTorrent* mensalmente. Para consultas sobre *commits*, os resultados são semelhantes nas três ferramentas, mas para linguagens de programação, a *GitHub API* é mais custosa, o *GitHub Archive* não oferece suporte e o *GHTorrent* exige múltiplas requisições. O estudo conclui que a escolha da ferramenta depende das necessidades da pesquisa.

A tabela 3 apresenta os trabalhos relacionados identificados e está organizada em seis colunas. A coluna "Nome" destaca o nome de cada um dos trabalhos. A coluna "MSR" destaca os artigos que abordam a área de Mineração de Repositórios. A coluna "Análises Comparativas" revela os trabalhos que realizam comparações entre abordagens ou entre outros trabalhos. A coluna de "Repositórios de plataformas além do *GitHub*" revela os trabalhos que utilizam não apenas repositórios do *GitHub*, mas também de outros sistemas de hospedagem (*e.g.*, *GitLab*). Por fim, a coluna "Limitações da *GitHub API*" destaca os trabalhos que citam ou que sofrem das limitações da *API*.

Tabela 3.1: Tabela comparativa entre os trabalhos

Nome	**MSR**	**Análises comparativas**	**Repositórios de plataformas além do *GitHub***	**Limitações da *GitHub API***
Sampling projects in GitHub for MSR studies	X			X
G-Repo: a Tool to Support MSR Studies on GitHub	X	X		X
Modelmine: A Tool to facilitate mining models from open source repositories	X	X	X	X
The GHTorrent dataset and Tool suite	X			X
GitHub REST API vs GHTorrent vs GitHub Archive: A Comparative Study		X		X
Findings from GitHub: Methods, Datasets and Limitations	X	X		X
Fingerprinting and Building Large Reproducible Datasets	X			X
MetricMiner: Supporting researchers in mining software repositories	X	X	X	
Mining and Creating a Software Repositories Dataset	X		X	X

Na Tabela 3, é possível perceber que apenas o trabalho de Mombach e Valente (2018) não aborda explicitamente o tema de Mineração de Repositórios, mesmo trazendo referências a artigos publicados em conferências de MSR. Os 4 primeiros trabalhos da tabela, assim como os trabalhos de Sokol, Aniche e Gerosa (2013) e Lefeuvre et al. (2023), trazem ferramentas que apoiam a área de MSR. A pesquisa de Cosentino, Luis e Cabot (2016) analisa 93 trabalhos vinculados área de MSR. Já a pesquisa de Do et al. (2020) trouxe um estudo de uma mineração realizada sob milhões de repositórios obtidos através da ferramenta *Software Heritage*.

Na Tabela 3 ainda é possível observar que os trabalhos de Romano et al. (2021), Reza, Badreddin e Rahad (2020), Mombach e Valente (2018), Cosentino, Luis e Cabot (2016) e Sokol, Aniche e Gerosa (2013) realizam análises comparativas: o primeiro faz uma comparação entre sete avaliadores e a ferramenta. O segundo faz uma comparação com a ferramenta *PyDriller*. O terceiro aborda uma análise comparativa entre as ferramentas *GHTorrent*, *GitHub API* e *GitHub Archive*. O quarto artigo realiza comparações entre 93 pesquisas com o tema de Mineração de repositórios de software. Por fim, o último artigo compara o *MetricMiner* com outras ferramentas como *Sonar*, *Boa*, *Eclipse Metrics*, etc.

Com relação aos sistemas de hospedagem de repositórios, todos os sistemas envolvidos nos artigos realizam a obtenção dos repositórios do *GitHub*. Porém, a ferramenta

de Reza, Badreddin e Rahad (2020) também permite a busca de repositórios no sistema *GitLab*. Já a abordagem de Sokol, Aniche e Gerosa (2013) possui suporte tanto para repositórios *Git* quanto *SVN*, ou seja, permite repositórios de sistemas como o *GitLab* além do próprio *GitHub*. Por fim, a abordagem de Do et al. (2020) também possui foco em repositórios advindos do *GitLab* ou *BitBucket*.

Por fim, apenas o trabalho de Sokol, Aniche e Gerosa (2013) não comenta sobre as limitações da *GitHub API*. Isso pode ocorrer devido ao fato do *MetricMiner* apenas utilizar repositórios adicionados pelos usuários.

4 Abordagem

Este capítulo apresenta uma abordagem que visa auxiliar pesquisadores da área de mineração de repositórios a buscarem por repositórios alinhados com seus ideais de pesquisa. Ela permite que o usuário realize as buscas dos repositórios por meio de suas características, como estrelas e linguagem de programação, e filtre os repositórios por meio de características e critérios de seleção definidos pelo usuário.

A abordagem busca também solucionar certas limitações encontradas na literatura, como: o limite de 1000 resultados por pesquisa, a restrição de metadados no processo de seleção de repositórios, a falta de rastreabilidade dos critérios utilizados pelo usuário para chegar ao conjunto de projetos selecionados para o corpo do experimento e os metadados de repositórios desatualizados obtidos por ferramentas da literatura. Para contornar tais problemas, a abordagem fragmenta as consultas que retornam mais de 1.000 repositórios através de suas características, armazena as decisões tomadas pelo usuário durante a seleção dos repositórios resultantes e sempre realiza pesquisas sobre as versões mais recentes dos repositórios.

A Figura 4.1 apresenta uma visão geral da abordagem, que possui quatro etapas: (1) a definição de parâmetros para a seleção de repositórios, (2) fragmentação da busca, (3) filtragem dos repositórios e criação de critérios, (3.A) Resultado da busca, (3.B) criação de critério, (3.C) realimentação da busca e (4) persistência dos resultados. Nas seções seguintes, serão descritas as etapas pensadas no desenvolvimento da abordagem e demonstradas pela Figura 4.1.

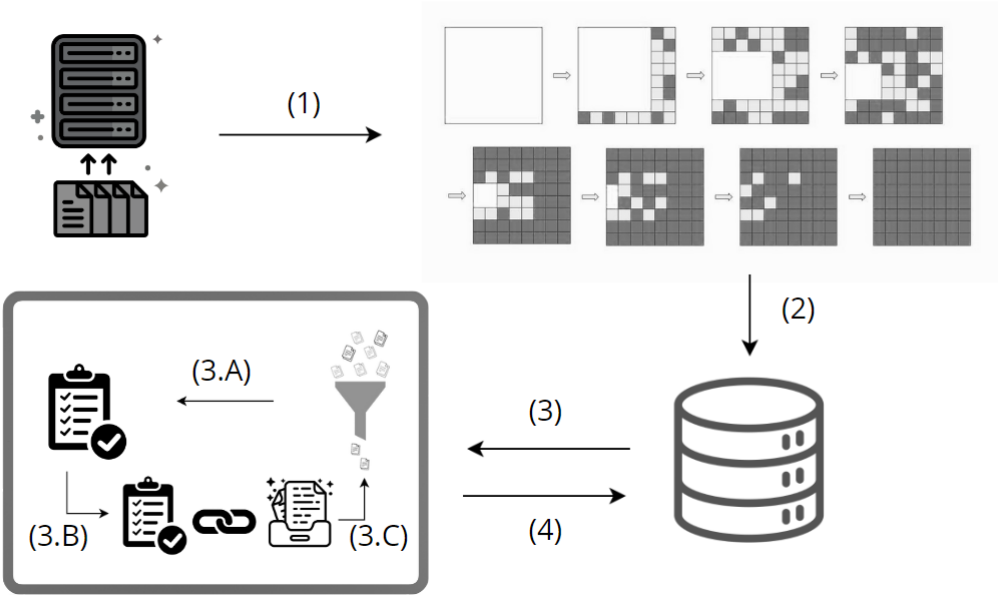

Figura 4.1: Arquitetura da abordagem

4.1 Definição de parâmetros para a seleção de repositórios

O intuito da abordagem é selecionar repositórios de software que estão disponíveis digitalmente; para isso, é necessário utilizar plataformas que armazenem repositórios. Alguns exemplos dessas plataformas são o *GitHub*, o *GitLab* e o *BitBucket*. Outra solução consiste em utilizar bancos de dados populados com os repositórios presentes nessas plataformas, como o *GHTorrent*.

Após definido de onde virão os repositórios, é necessário definir as características dos repositórios que serão obtidos na pesquisa. Tais características podem ser os próprios metadados dos repositórios, como a quantidade de estrelas dos repositórios dos *GitHub*, que indicam a popularidade de um repositório, e as linguagens de programação presentes no repositório, que indicam quais tecnologias foram adotadas na implementação do software.

4.2 Fragmentação da busca

Com a base de repositórios e as características definidas, o usuário parte para a realização da pesquisa. Para obter todos os repositórios que possuem essas características, a abordagem adota a fragmentação da busca. Esta fragmentação é necessária para contornar a limitação de 1000 repositórios por pesquisa, encontrada em certas abordagens que utilizam a *GitHub API*, como a de Reza, Badreddin e Rahad (2020), e superada por outras como a de Romano et al. (2021).

O método de fragmentação adotado divide a pesquisa com base nos metadados dos repositórios. Esses metadados podem ser, por exemplo, o número de estrelas, a data de criação e o tamanho dos repositórios. Essencialmente, a fragmentação é realizada ao máximo para uma dada característica. Se essa fragmentação atingir o menor intervalo possível e ainda não retornar mais que 1000 repositórios, outra característica é fragmentada juntamente com a anterior. Esse processo é repetido sucessivamente, garantindo que, em algum momento, no máximo 1000 repositórios serão retornados em uma busca.

Para um exemplo hipotético, suponha-se uma pesquisa por repositórios com mais de 1000 estrelas, tal fragmentação relacionará as três características dos repositórios citadas anteriormente: o número de estrelas, a data de criação e o tamanho dos repositórios. A primeira fase da fragmentação utiliza o número de estrelas dos repositórios, busca-se pelo limite inferior passado como parâmetro, que neste caso é 1.001 estrelas. Em seguida, testa-se a busca entre intervalos fechados de estrelas, como de 1.001 a 500.001, verificando se o retorno é de até 1.000 repositórios, tais intervalos são pré-definidos e fixos na implementação da abordagem. Caso o conjunto não satisfaça as condições, o intervalo de busca é reduzido para 1.001 a 200.001 estrelas, e assim por diante. Caso necessário, a segunda fase foca na data de criação dos repositórios. O processo se inicia a partir de uma certa data, começando com intervalos anuais, por exemplo, buscaria por repositórios criados entre 01/01/2006 e 01/01/2007. Se ainda não for o suficiente, a fragmentação ocorrerá por intervalos mensais, e depois por intervalos diários. Se ainda for preciso, a terceira etapa segmenta a busca com base no tamanho do repositório. A fragmentação começa com um valor mínimo de 1 KB para ignorar repositórios vazios, como de 1 KB a 100.000 KB, reduzindo esse intervalo sucessivamente. O processo continua até alcançar o

menor intervalo possível de 1 KB de diferença. A Figura 4.2 demonstra uma árvore de decisão de como ocorre uma possível fragmentação de uma pesquisa. As fragmentações vão ocorrendo a medida que são necessárias e se ramificam para um próximo critério, caso o critério anterior tenha sido fragmentado ao máximo (*e.g.*, data de criação de 08/04/2024 até 09/04/2024).

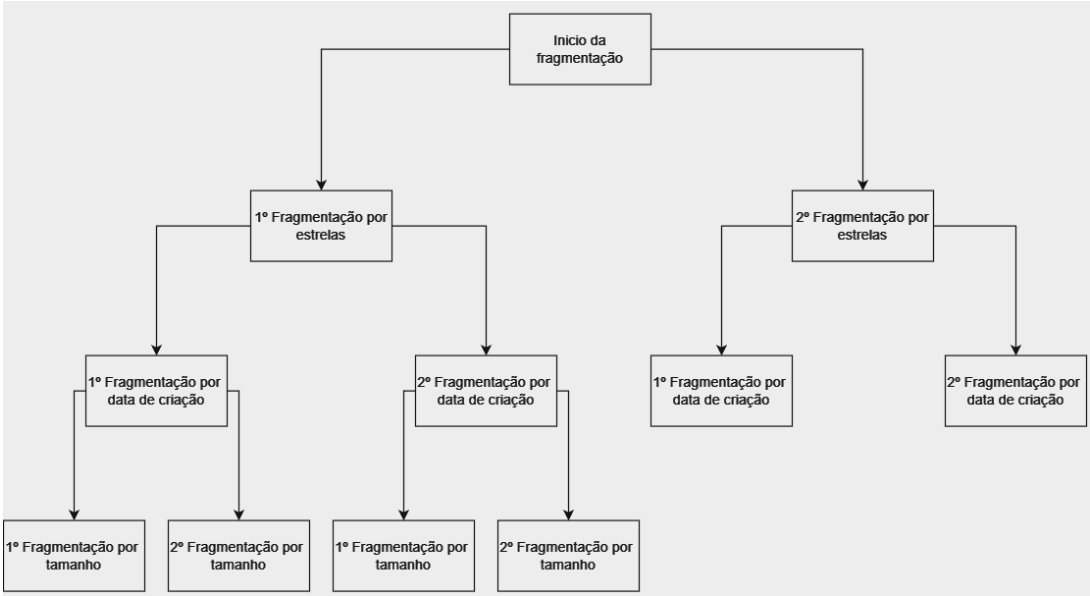

Figura 4.2: Árvore de decisão que demonstra como ocorre a fragmentação.

Caso, mesmo após a fragmentação utilizando os três critérios (estrelas, data de criação e tamanho), ainda houver mais de 1.000 repositórios na busca, serão retornados apenas os primeiros 1.000 repositórios que a *GitHub API* fornecer, ignorando o excedente restante e, para o usuário tomar ciência de tal limitação, uma notificação é enviada avisando-o que a busca seguirá com os 1000 repositórios encontrados. Essa fragmentação ocorre como um efeito cascata, em que o algoritmo se fragmenta ao máximo mediante um critério. Caso isso não seja suficiente, passa para os critérios seguintes. Se encontrar resultados viáveis, eles são retornados.

Uma busca real que pode ser utilizada é a procura por repositórios com mais de 1.000 estrelas que utilizem a linguagem *TypeScript*. Ao realizar essa pesquisa no *GitHub*, foram encontrados 3.663 repositórios, um valor que ultrapassa o limite de 1.000 repositórios. Deste modo, a abordagem passa para a primeira etapa, quebrando o conjunto em partes menores considerando o número de estrelas. Neste exemplo, o algoritmo irá

fragmentar a busca em 5 conjuntos. O primeiro, que retorna 487 repositórios com quantidade de estrelas variando de 1.001 a 1.201, o segundo, que retorna 824 repositórios com quantidade de estrelas variando de 1.202 a 1.702, o terceiro, que retorna 788 repositórios com o número de estrelas variando de 1.703 a 2.703, o quarto, que retorna 984 repositórios com o número de estrelas variando de 2.704 a 7.704, e, por fim, uma última busca que retorna 580 repositórios com a quantidade de estrelas restantes. Somando as cinco buscas, os 3.663 repositórios iniciais foram obtidos.

A fragmentação se torna essencial quando um grupo contém mais de 1000 repositórios, momento em que o algoritmo de fragmentação é empregado para reduzir os grupos a um tamanho máximo de 1000 repositórios cada. À medida que os grupos são divididos em subgrupos menores, este processo é referido como algoritmo de fragmentação.

4.3 Criação de critérios e filtragem dos repositórios

A abordagem dá ao usuário a possibilidade de criar critérios que permitem que ocorra o armazenamento e o rastreamento do histórico das decisões do usuário ao longo da seleção de repositórios, facilitando a identificação dos repositórios que estiverem ou não alinhados com seus ideais de pesquisa e permitindo que haja a reprodutibilidade dos experimentos.

Estabelecer critérios para os repositórios simplifica a compreensão das motivações por trás das seleções feitas pelos pesquisadores. Considere, por exemplo, um pesquisador que deseja identificar repositórios que utilizem um tipo específico de *framework* de desenvolvimento. Ele pode criar um critério que explicite qual *framework* é esse (*React*, por exemplo) e aplicar tal critério nos repositórios que possuem um software desenvolvido com *React* com o intuito de caracterizá-los. Com esse critério definido, ao analisar diversos repositórios, ele conseguirá classificar cada um dos repositórios segundo o critério estabelecido, determinando quais são pertinentes à sua pesquisa.

Tais critérios podem ser definidos de duas maneiras, a primeira é como critérios de inclusão, que registrarão os repositórios alinhados à pesquisa do usuário. A segunda maneira é como critérios de exclusão, estes registrarão quais repositórios não fazem parte dos ideais de pesquisas do usuário. Com a utilização de ambos os critérios combinados, o pesquisador pode facilmente agrupar os repositórios entre os que entraram e aqueles que

não entraram para o experimento.

É possível definir uma operação de conjuntos, para generalizar qualquer filtragem por critérios na abordagem. Seja um filtro do usuário, no qual todos os critérios utilizados são de inclusão, nomeados de CI_i, com i variando de 1 ao número de critérios de inclusão selecionados. Sendo assim, um filtro que contém apenas critérios de inclusão é dado pela união entre todos os repositórios que atendem tais critérios, como definido pela Equação 4.1.

$$(CI_1 \cup CI_2 \cup CI_3 ... \cup CI_i) \tag{4.1}$$

A operação geral dos critérios de exclusão é semelhante. Seja um filtro do usuário, no qual todos os critérios utilizados são de exclusão, nomeados de CE_j, com j variando de 1 ao número de critérios de exclusão selecionados. Sendo assim, um filtro que contém apenas critérios de exclusão é dado pela união entre todos os repositórios que atendem tais critérios, como definido na Equação 4.2.

$$(CE_1 \cup CE_2 \cup CE_3 ... \cup CE_j) \tag{4.2}$$

O conjunto de critérios de exclusão subtrai do conjunto de critérios de inclusão todos os repositórios que estejam nos dois conjuntos. Caso nenhum critério de inclusão seja selecionado no filtro pelo usuário, será utilizado o conjunto de todos os repositórios obtidos na busca original, que será definido aqui como conjunto Universo (U). Dessa forma, a operação pode ser definida como na Equação 4.3.

$$U \setminus (CE_1 \cup CE_2 \cup CE_3 ... \cup CE_j) \tag{4.3}$$

Caso tanto os critérios de exclusão, quanto os de inclusão sejam adicionados no filtro, será possível substituir o conjunto universo pela operação de união entre os critérios de inclusão definida anteriormente. Tendo a operação geral definida por 4.4

$$(CI_1 \cup CI_2 \cup CI_3 ... \cup CI_n) \setminus (CE_1 \cup CE_2 \cup CE_3 ... \cup CE_j) \tag{4.4}$$

Em um exemplo real, um pesquisador deseja repositórios com mais de 220.000 *estrelas* e, dentre esses repositórios, definir quais armazenam um software e quais não armazenam. Essa busca retornou um total de 10 repositórios, que podem ser vistos na Tabela 4.1.

Tabela 4.1: Repositórios com mais de 220.000 estrelas

Repositórios
freeCodeCamp
free-programming-books
awesome
public-apis
coding-interview-university
build-your-own-x
developer-roadmap
996.ICU
system-design-primer
react

Após a obtenção dos repositórios, o usuário criou 2 critérios, o primeiro nomeado "Armazena um software", que foi atribuído aos repositórios presentes na Tabela 4.2, e outro nomeado "Não armazena um software", que foi atribuído aos repositórios presentes na Tabela 4.3. Com isso, ele obteve um total de 5 repositórios que continham o primeiro critério e os outros 5 que continham o segundo. Na Figura 4.3, é possível visualizar o conjunto de repositórios.

Tabela 4.2: Repositórios que armazenam um software

Repositórios
freeCodeCamp
public-apis
developer-roadmap
system-design-primer
react

Tabela 4.3: Repositórios que não armazenam um software

Repositórios
free-programming-books
awesome
coding-interview-university
build-your-own-x
996.ICU

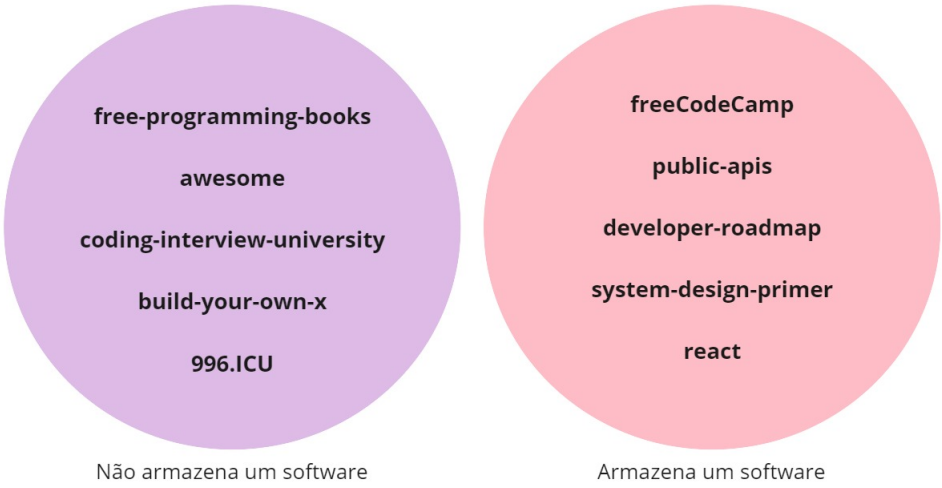

Figura 4.3: Conjuntos de repositórios atrelados aos 2 critérios de definidos nesta etapa.

Após um tempo, foi decidido que os repositórios que tivessem um patrocinador deveriam ser excluídos deste filtro da pesquisa, portanto, foi criado o critério nomeado "Possui patrocinador". Tal critério foi atribuído aos repositórios da Tabela 4.4. Na Figura 4.4, é possível visualizar o conjunto de repositórios.

Tabela 4.4: Repositórios que possuem patrocinador

Repositórios
freeCodeCamp
free-programming-books
awesome
coding-interview-university
developer-roadmap

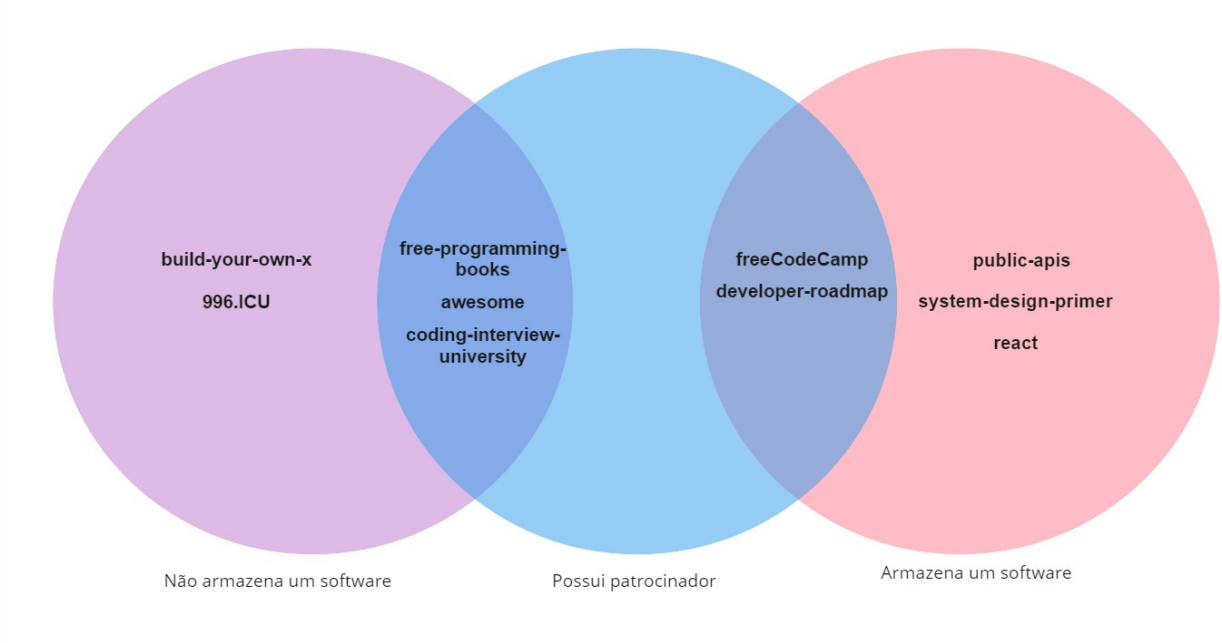

Figura 4.4: Conjunto de repositórios atrelados aos 3 critérios definidos nesta etapa.

Com os critérios definidos e vinculados, o usuário partiu para a realização da filtragem e definiu os 3 critérios criados da seguinte forma, os critérios de "Armazena um software" e "Não armazena um software" foram definidos como critérios de inclusão, o critério "Possui patrocinador" foi definido como critério de exclusão. A união da Equação 4.5 explicita a relação entre os 2 critérios de inclusão definidos.

$$Software \cup NaoSoftware. \tag{4.5}$$

Porém, como explicitado pela Figura 4.4, com a utilização do critério de patrocinador como um critério de exclusão, ocorre de 5 repositórios serem removidos do conjunto resultante anterior, obtendo então apenas os seguintes repositórios presentes na tabela 4.5.

Tabela 4.5: Repositórios que não possuem patrocinador

Repositórios
build-your-own-x
996.ICU
system-design-primer
react
public-apis

A operação de conjuntos 4.6 define o resultado da filtragem.

$$(Software \cup NaoSoftware) \setminus (Patrocinador) \tag{4.6}$$

Após a obtenção dos repositórios através da pesquisa e definido os critérios, o usuário pode realizar uma filtragem que permita uma seleção mais específica dos repositórios obtidos, utilizando tanto os critérios vinculados aos repositórios, quanto os metadados presentes nos mesmos. Com isso, o usuário tem a chance futura de buscar por repositórios de forma mais específica dentre suas pesquisas já realizadas.

Por fim, com toda a seleção do repositório feita, a abordagem visa armazenar todo o histórico da pesquisa, como os parâmetros iniciais de busca, os repositórios resultantes, os critérios criados e vinculados nos repositórios e as filtragens feitas sobre os repositórios. Isso garante que, no futuro, o pesquisador tenha acesso aos repositórios já obtidos, para averiguar a motivação por trás dos experimentos e os resultados obtidos, e também permitindo-o reciclar os repositórios para a realização de uma nova seleção de repositórios. Isso resulta no enriquecimento da pesquisa atual e de possíveis pesquisas futuras, ao armazenar todo o histórico de decisão do pesquisador, e possibilitar a reprodutibilidade dos experimentos.

4.4 Considerações Finais

O capítulo inicia apresentando as motivações por trás do trabalho, quais problemas da literatura ele visa suprir, como a limitação de 1000 repositórios por pesquisa e, por fim, estabelece uma arquitetura geral da abordagem apresentada. Inicialmente, é estabelecido de onde podem vir os repositórios a serem buscados e quais características podem ser utilizadas para filtrar os repositórios retornados. Depois é demonstrado o modelo de fragmentação de buscas adotado para suprir o problema de apenas 1000 repositórios por busca. Com a pesquisa iniciada já estabelecida é definido o conceito de critérios e como eles podem auxiliar a tomada de decisão nas pesquisas. Também expõe como a abordagem pode auxiliar os pesquisadores através do armazenamento do histórico das pesquisas e das tomadas de decisões já feitas. Como contribuição gerais, o software supre as limitações de ferramentas que, como a *GitHub API*, trazem apenas 1000 repositórios por pesquisa. A abordagem também traz novos campos de busca, como os campos de múltiplas linguagens, de *commits* e de *watchers*.

5 Implementação da abordagem

Este capítulo é destinado à apresentação da implementação da abordagem. A Seção 5.1 traz as tecnologias empregadas para a implementação do software. A Seção 5.2 demonstra o funcionamento geral da abordagem, mostrando um exemplo de pesquisa para demonstrar passo a passo como as funcionalidades foram desenvolvidas e servindo como uma documentação para a execução do software.

5.1 Tecnologias utilizadas

A abordagem faz o uso de várias tecnologias para englobar todas as funcionalidades. A linguagem principal empregada é o *Python*, que serve como a base para os algoritmos e o funcionamento da ferramenta. Além disso, as linguagens *JavaScript*, *CSS* e *HTML* foram usadas como alicerces para a apresentação visual.

Em relação aos *frameworks*, foram utilizados o *Django* e o *Celery*. O *Django* é responsável pela estruturação do código seguindo a arquitetura MTV (model–template–views), enquanto o *Celery* organiza, sinaliza e agenda tarefas em filas de execução assíncronas, garantindo um funcionamento mais rápido e organizado da ferramenta.

O banco de dados utilizado por padrão no *framework Django* foi o *SQLite*. Além disso, o *Redis* foi empregado como um banco de dados de chave-valor na memória, atuando como cache e corretor de mensagens. Essa configuração agiliza a comunicação entre as tarefas ao desacoplar as tarefas do algoritmo relacionado à visualização no servidor Web do *Django*.

A ferramenta também integra a *GitHub API V4*, que utiliza a linguagem de consulta *GraphQL* para buscar repositórios no *GitHub* por meio de *queries*. Por fim, para facilitar a execução do código pelo usuário, foi utilizado o *Docker*. Ele permite a criação de um contêiner que empacota todos os arquivos necessários para a execução do software, simplificando assim o uso da ferramenta.

5.2 Funcionamento da abordagem

A Figura 5.1 mostra a tela inicial da ferramenta, a qual permite ao usuário definir a sua busca mediante certos metadados dos repositórios do *GitHub*. O campo *Research Theme* está relacionado ao título da pesquisa e permite definir o objetivo da busca, além de diferenciá-la de outras possíveis pesquisas realizadas pelo usuário. O campo *User* é destinado a identificar qual usuário realizou a pesquisa.

O campo *Stars* é associado ao metadado de estrelas do *GitHub* e permite ao usuário utilizar operadores matemáticos, como '>', '<', '>=' e '<=', seguidos de um número inteiro, para pesquisar repositórios com base na quantidade de estrelas. Além disso, é possível usar o operador '..' entre dois números inteiros para definir um intervalo de estrelas (*e.g.*, 1000..5000). De maneira semelhante, o campo *Forks* está relacionado ao metadado do *GitHub* de quantidade de *forks* e utiliza a mesma notação que o campo *Stars*. O campo *Watchers* está relacionado a quantidade de *Watchers* do *GitHub*, seguindo a mesma notação. Já o campo *Commits* refere-se a quantidade de *commits* do *GitHub*, utilizando também notação similar ao campo *Stars*.

O campo *Languages* permite pesquisar por múltiplas linguagens nos repositórios desejados, utilizando a sintaxe do operador lógico 'AND' para combinar as linguagens (*e.g.*, Java AND JavaScript AND Python). Já o campo *Main Language* diverge do anterior, ao permitir a busca por uma única linguagem, considerada a principal do repositório. Assim, são retornados apenas os repositórios que têm essa linguagem em maior porcentagem em relação às outras presentes. A Figura 5.1 mostra a tela inicial de pesquisa realizando uma busca por repositórios que possuem a linguagem *Java* como principal e mais de 1000 *estrelas* e demonstrando os campos citados anteriormente.

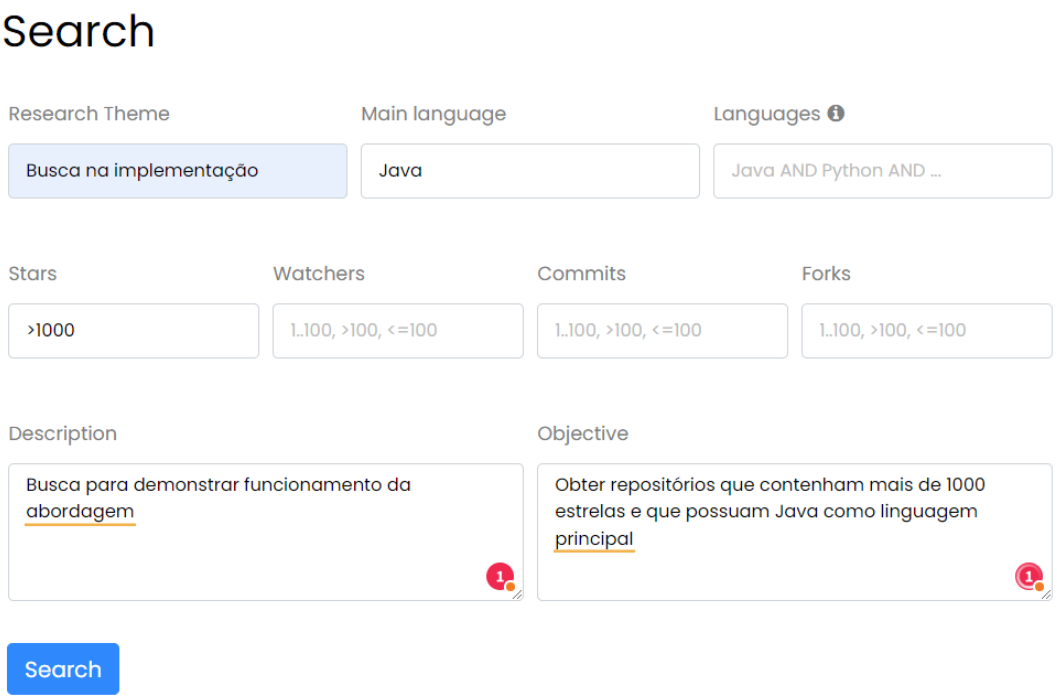

Figura 5.1: Tela de pesquisa com valores preenchidos.

Divergindo dos outros campos presentes na busca, os campos *Commits*, *Watchers* e *Languages* não são passados para a *GitHub API* no momento de buscar os repositórios, por não serem campos disponíveis na *API*. Essa restrição pode ser considerada então uma limitação da *API*. Para contornar isso, foi necessário utilizar a *API* sem esses 3 campos e realizar uma filtragem posterior sobre esses metadados no software, garantindo que os resultados mostrados estejam no escopo definido pelo usuário. Por fim, para evitar repositórios vazios que não trarão nenhum valor para a pesquisa, foi definido nas *queries* de busca um valor padrão de tamanho de repositórios maior que 0 KB.

Caso o usuário tente iniciar a pesquisa e os valores preenchidos nos campos não estejam condizentes com a sintaxe definida, uma mensagem de aviso será disparada alertando haver algum problema em um dos campos da pesquisa, sendo necessário realizar a correção.

Com a iniciação da pesquisa, a aplicação leva o usuário para a tela de buscas ativas, que mostra todas as buscas do usuário que estão em execução, como mostra a Figura 5.2. A listagem de pesquisas mostra todas as pesquisas realizadas pelo usuário com os campos de título, o estado de andamento de cada pesquisa, o usuário realizador,

a data de início e um link para a página de resultado de cada uma delas. Com relação ao estado de andamento, tal campo é definido por meio de 3 situações possíveis. O primeiro, representado pela sigla 'OG', que significa *Ongoing*, demonstra que a pesquisa está em estado de andamento. O segundo estado representado pela sigla 'ST', que significa *Stopped*, demonstrando que a pesquisa está em estado de espera. Por fim, o terceiro estado, representado pela sigla 'CO', que significa *Completed*, mostra que a pesquisa foi finalizada.

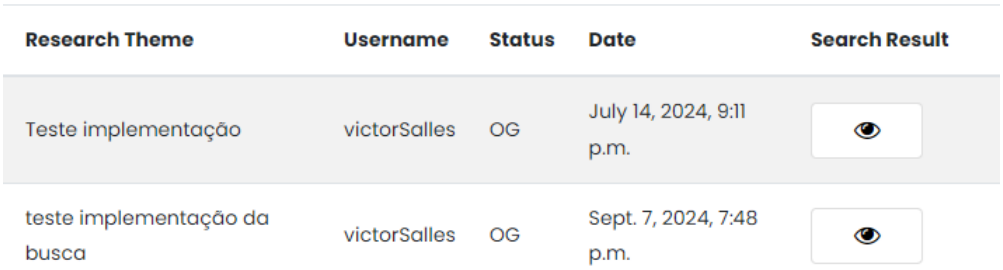

Figura 5.2: Tela de buscas ativas.

Ao clicar no ícone do olho de uma dada pesquisa, o usuário será levado a tela de repositórios retornados, como mostra a Figura 5.3. Tal busca retornou um total de 3443 repositórios com linguagem principal *Java* e com mais de 1000 *estrelas*. Estes repositórios ficaram todos paginados para uma melhor visualização e, por padrão, foi definido que cada página trouxesse um total de 24 repositórios. Em cada um dos repositórios, é possível visualizar metadados como quantidade de estrelas, porcentagem de cada linguagem de programação e descrição.

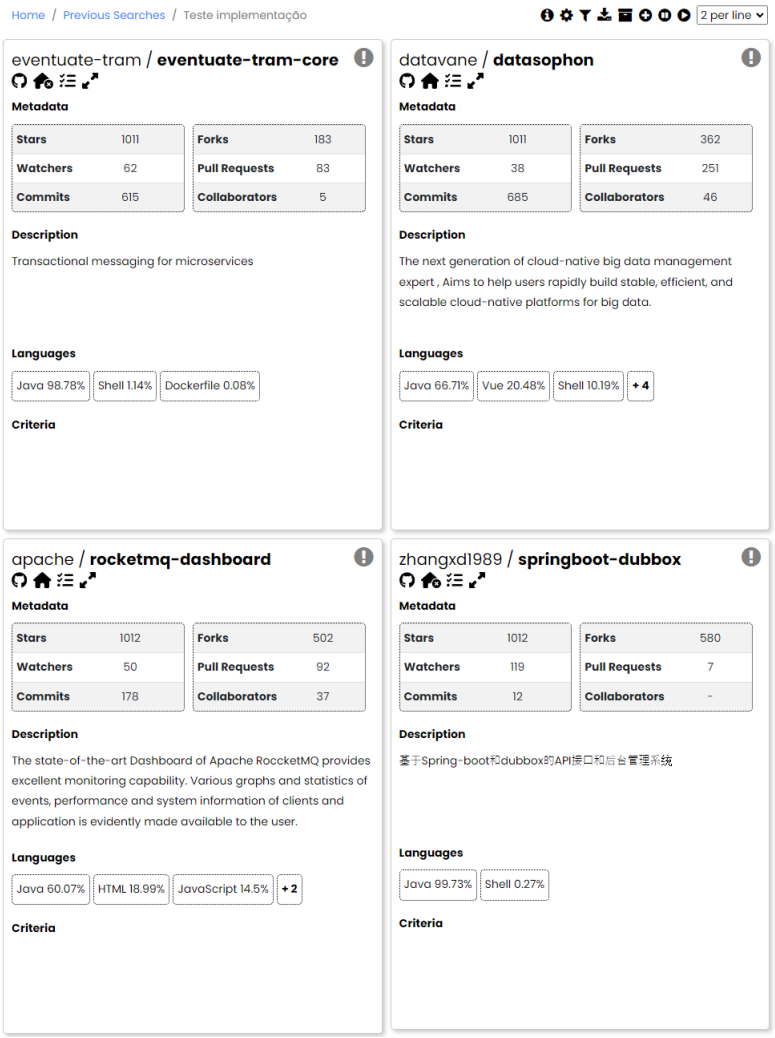

Figura 5.3: Tela de Resultados.

Os ícones superiores à direita na Figura 5.3 permitem algumas funcionalidades. O primeiro ícone leva o usuário para a tela de detalhes da pesquisa (Figura 5.4), que apresenta os valores preenchidos no momento da pesquisa, sendo possível relembrar os critérios. O segundo ícone leva para uma tela que permite ao usuário selecionar os campos que estarão disponíveis na tela de filtragem da pesquisa (Figura 5.5). O terceiro ícone gera uma tela que dá a possibilidade do usuário preencher campos que serão utilizados para realizar a filtragem dos resultados (Figura 5.6). O quarto ícone permite a seleção dos metadados dos repositórios para realizar o *download* dos resultados no formato de arquivos PDF ou XLSX, trazendo também a possibilidade de realizar a formatação dos campos de data (Figura 5.7). O quinto ícone realiza o arquivamento da pesquisa. Para desarquivar, é necessário ir para a tela de buscar arquivadas (Figura 5.8) e desarquivá-las.

O sexto ícone exibe uma tela de criação de um critério de seleção, no qual o usuário pode preencher um título e uma descrição, que definirão um critério (Figura 5.9). O penúltimo ícone realiza a pausa da pesquisa para ser retomada posteriormente, o *status* é alterado para "Stopped"/"ST" (Figura 5.10). O último ícone permite a continuação da pesquisa do ponto em que foi pausado, retornando para o *status* de "OnGoing"/ "OG" (Figura 5.2).

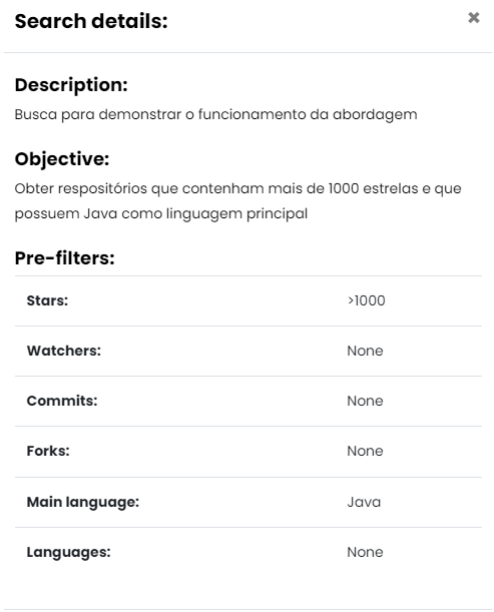

Figura 5.4: Tela de destalhes de pesquisa.

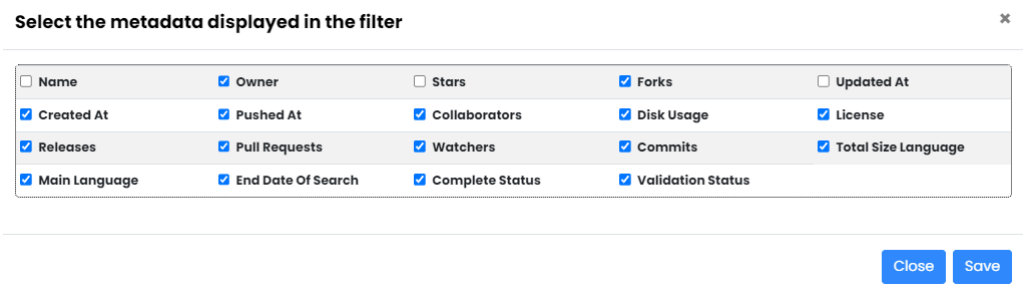

Figura 5.5: Tela de seleção de metadados para o filtro.

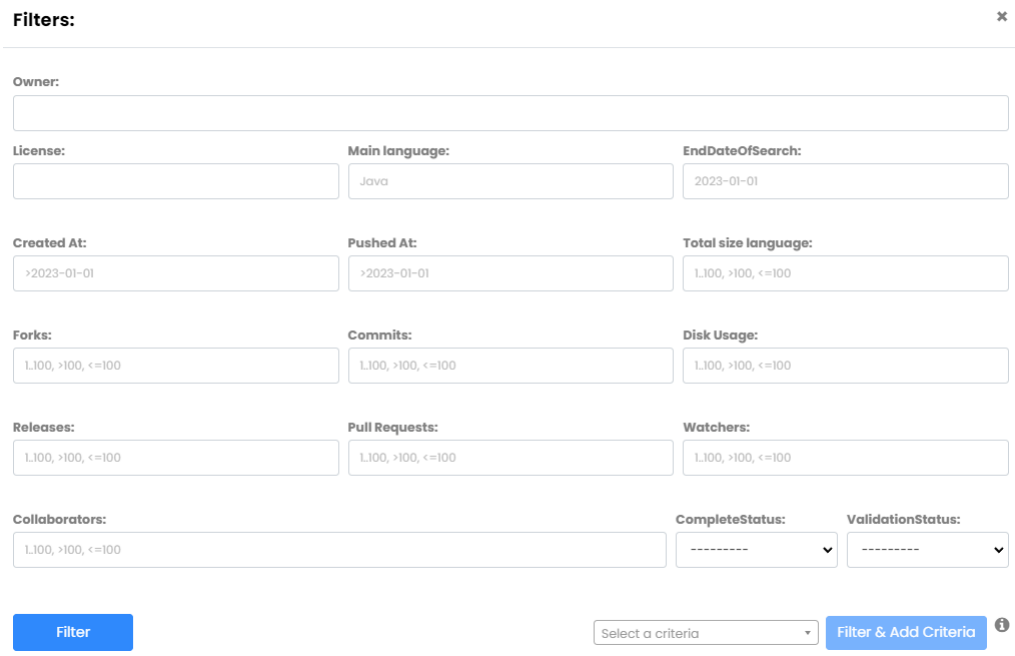

Figura 5.6: Tela de filtragem dos repositórios.

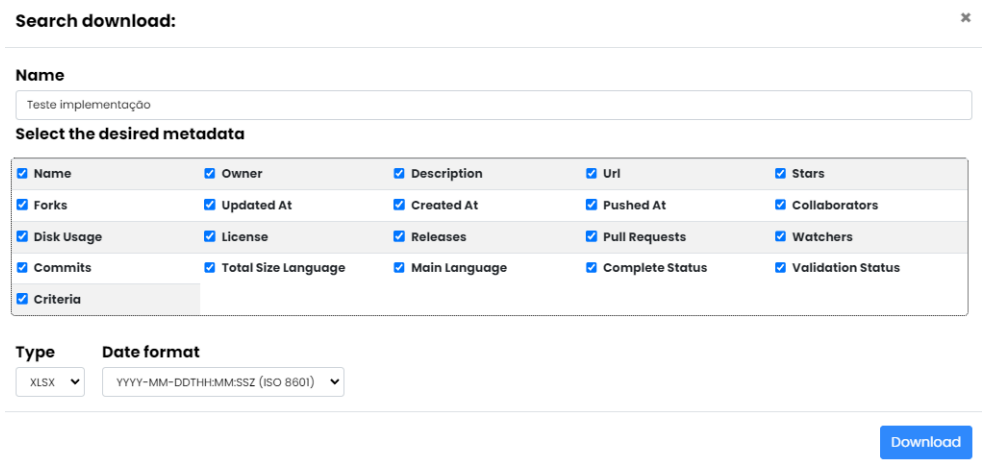

Figura 5.7: Tela de download.

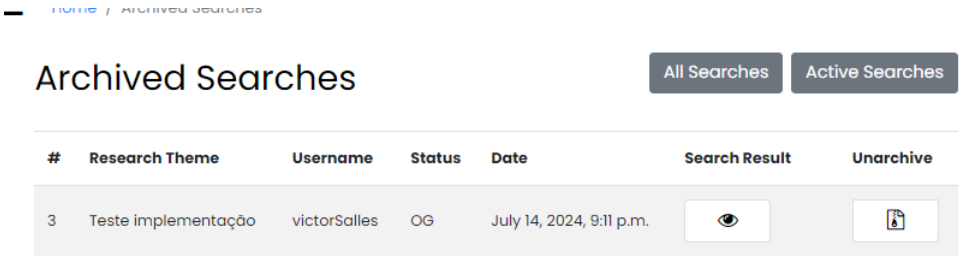

Figura 5.8: Tela de pesquisas arquivadas.

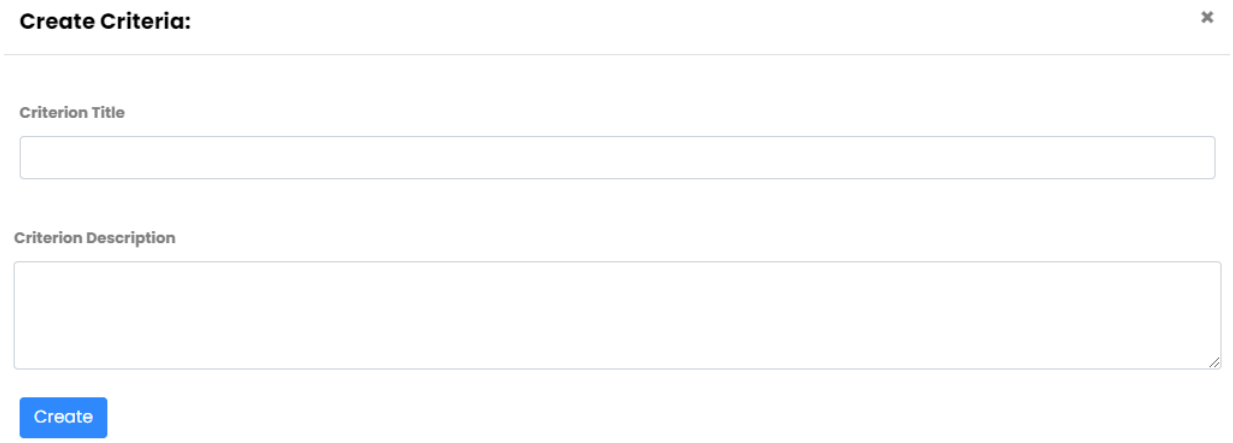

Figura 5.9: Tela de criação de critérios de seleção.

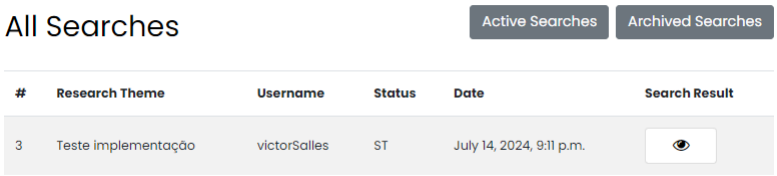

Figura 5.10: Tela que demonstra a pesquisa pausada.

Após todos os repositórios terem sido obtidos, foi definido um critério nomeado de "Repositório que armazena um software", que será aplicado aos repositórios que armazenarem um projeto de software qualquer (Figura 5.11).

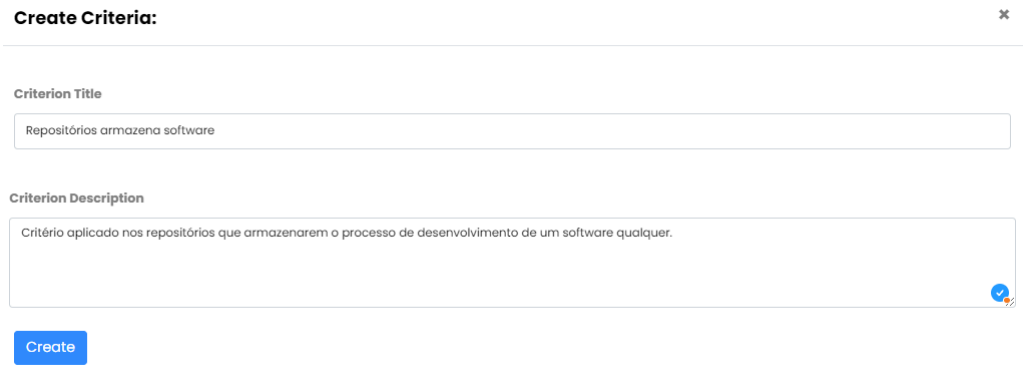

Figura 5.11: Tela de criação do critério.

Clicando no 3º ícone presente internamente em cada um dos repositórios, é possível aplicar os critérios criados no repositório selecionado, como mostra a Figura 5.12. Em seguida, foi criado um segundo critério, para definir os repositórios que contenham uma *Wiki*, sendo adicionado ao repositório "eventuate-tram-core". Após aplicado ambos os critérios, eles podem ser vistos vinculados a cada um dos repositórios, como mostra a Figura 5.13 .

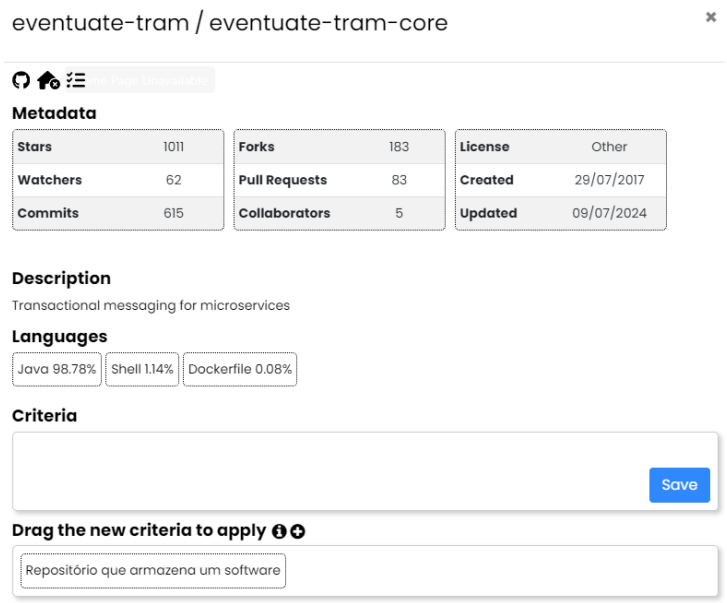

Figura 5.12: Tela de Aplicação do critério.

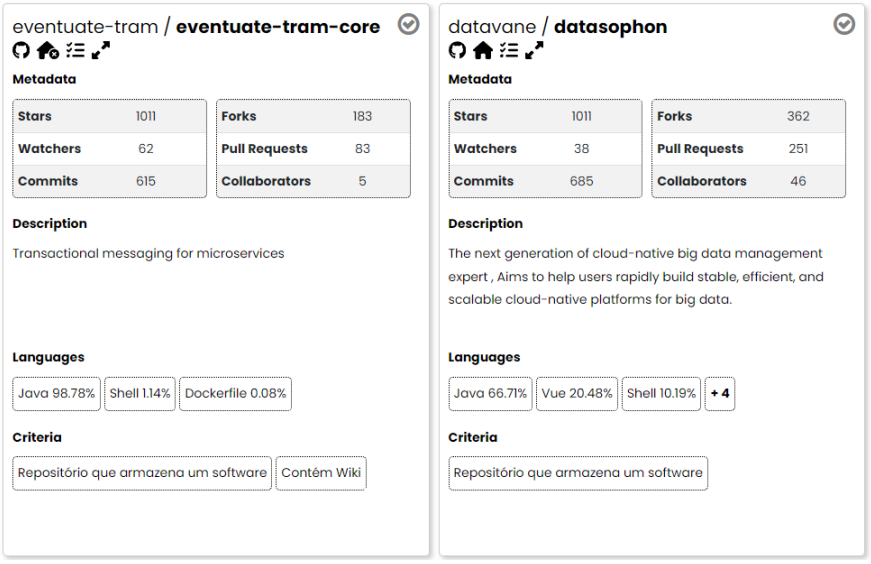

Figura 5.13: Critérios visíveis na tela de resultados da pesquisa.

Com os critérios aplicados, o usuário pode utilizá-los na filtragem. É possível selecionar quais critérios serão de inclusão e quais serão de exclusão (Figura 5.14). Neste exemplo, serão retornados os repositórios que armazenarem um software, mas que não contiverem uma wiki, que, nesse caso, será apenas 1 único repositório, pois somente ele possui o critério de "Repositório que armazena um software", mas não possui o critério de "Contém Wiki" (Figura 5.15).

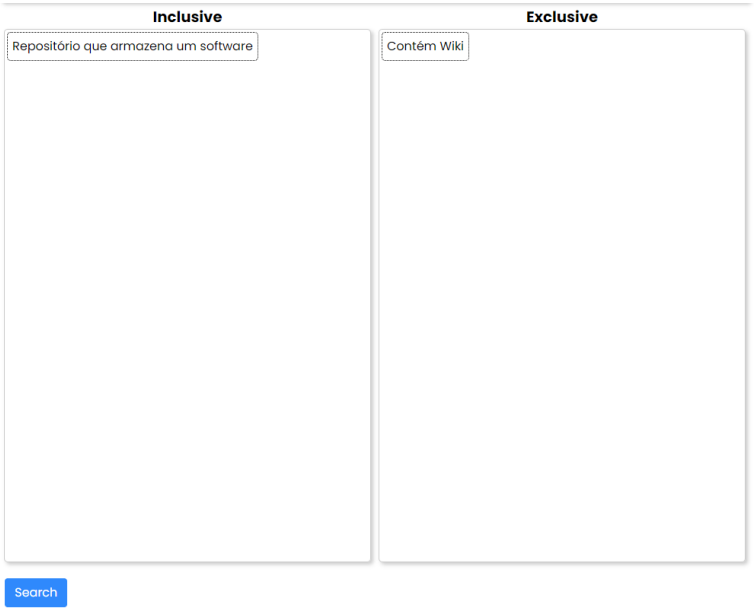

Figura 5.14: Tela de filtragem por critérios (1º exemplo).

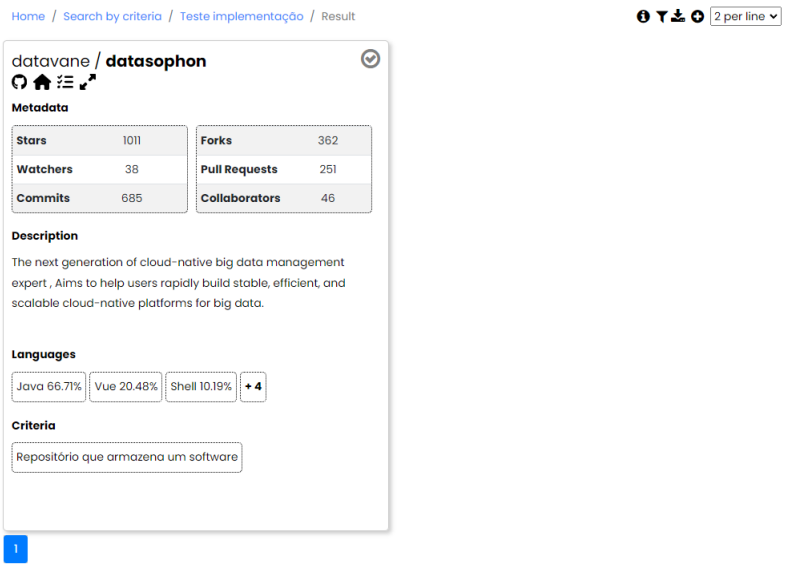

Figura 5.15: Repositório retornado da filtragem por critérios (1º exemplo).

Caso o usuário queira que seja retornado todos os repositórios que contenham pelo menos um dos critérios, ele pode adicioná-los como critérios de inclusão, como na Figura 5.16, retornando assim os 2 repositórios (Figura 5.17).

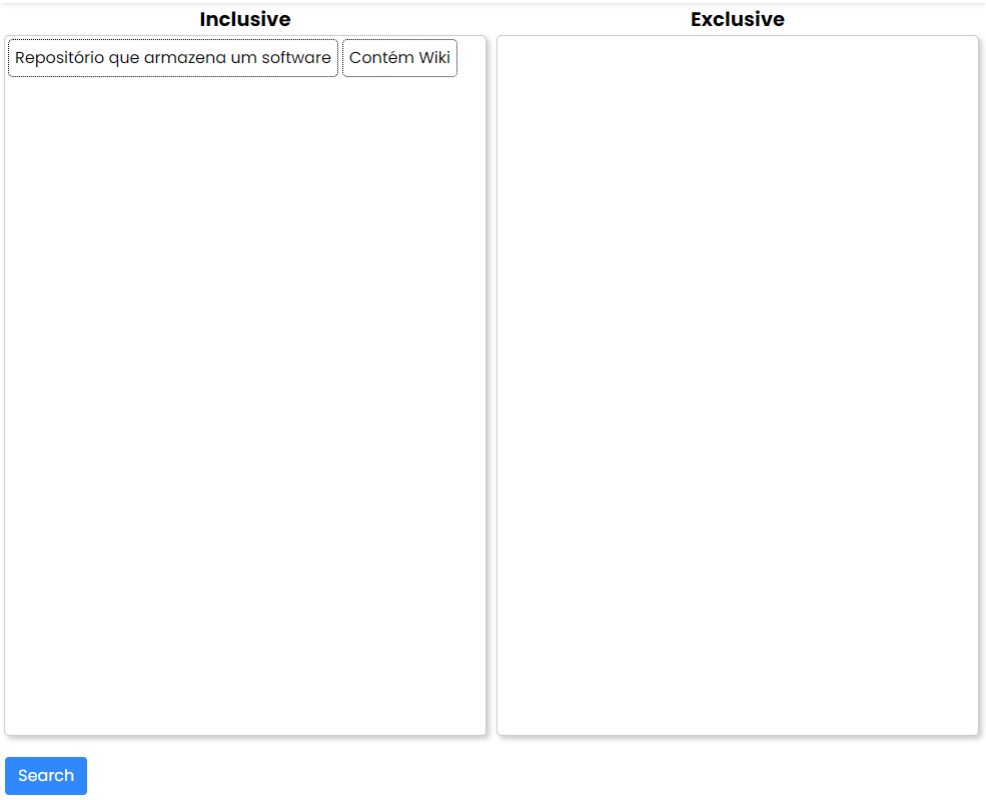

Figura 5.16: Tela de filtragem por critérios (2º exemplo).

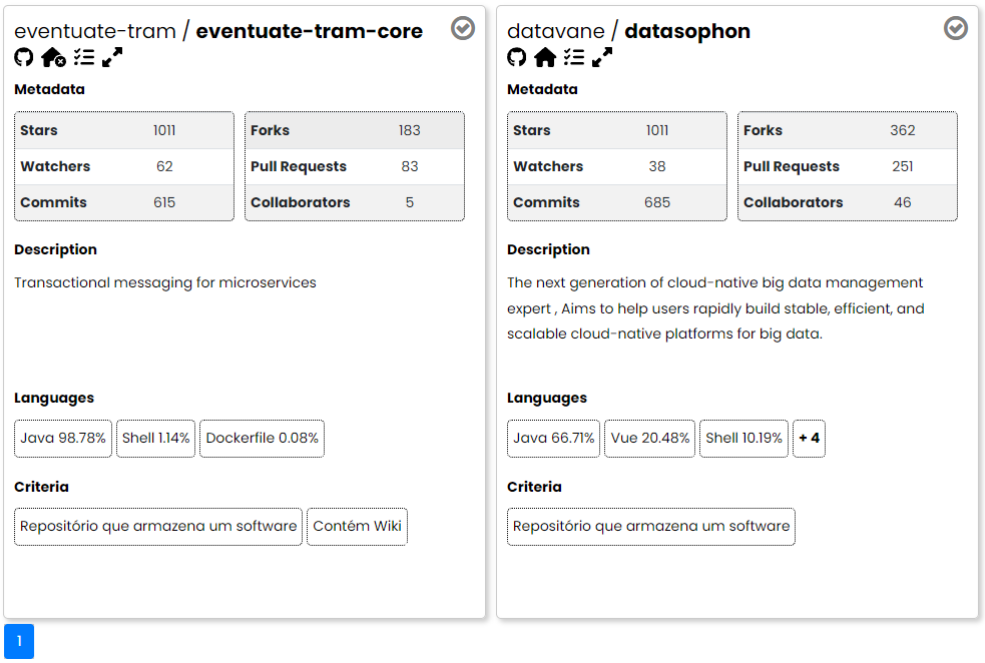

Figura 5.17: Repositórios retornados da filtragem por critérios (2º exemplo).

Por fim, a filtragem por critérios pode ser utilizada para buscar por todos os repositórios que não contiverem nenhum dos critérios aplicados, ao utilizar ambos os critérios como de exclusão, como na Figura 5.18. Assim, todos os repositórios que não possuírem nenhuma das tags vinculadas serão retornados (Figura 5.19).

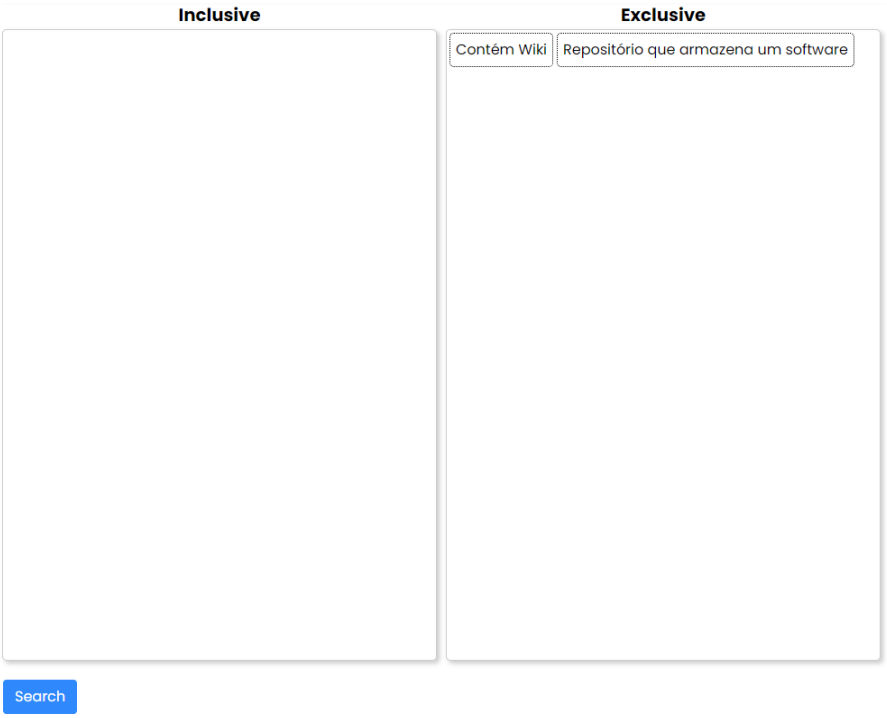

Figura 5.18: Tela de filtragem por critérios (3º exemplo).

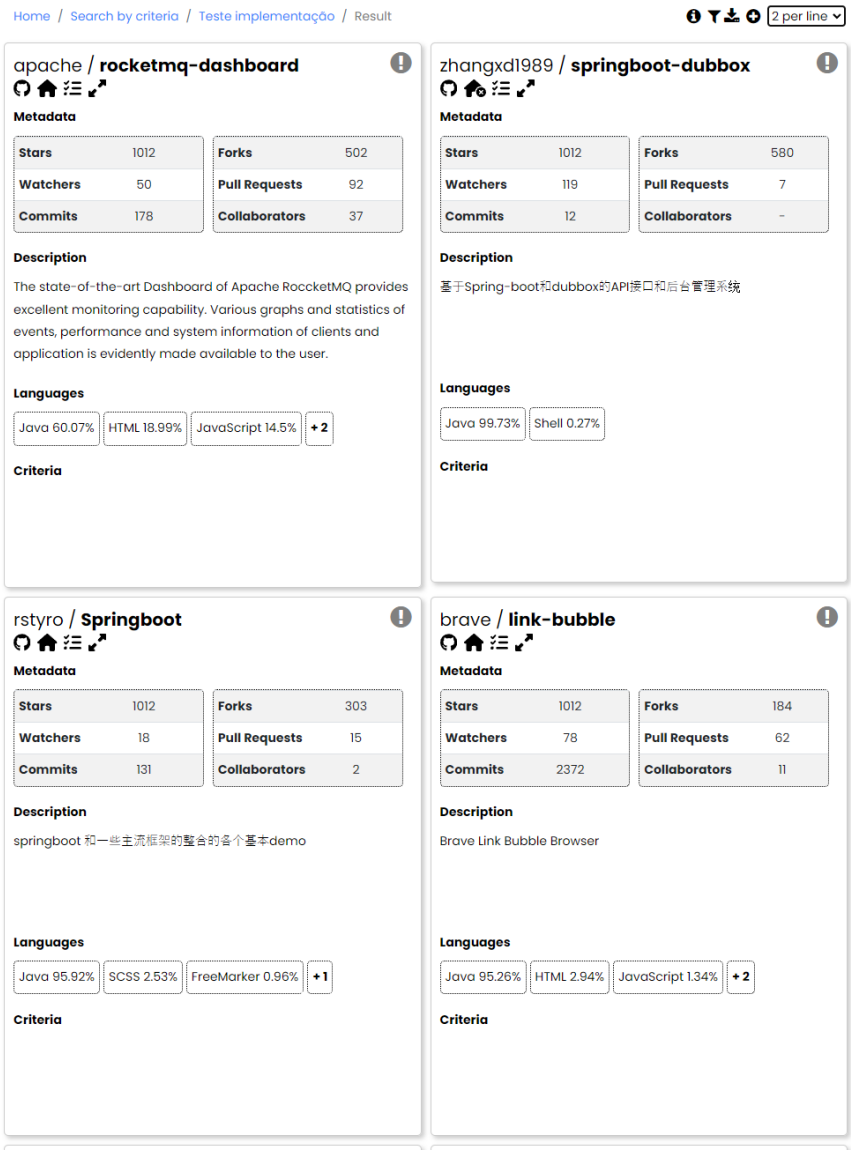

Figura 5.19: Repositórios retornados da filtragem por critérios (3º exemplo).

5.3 Limitações da implementação

Esta seção descreve as limitações ainda perceptíveis na implementação da abordagem, que podem ser tratadas em trabalhos futuros. Elas são as seguintes:

- A ferramenta atual realiza a busca pelos repositórios presentes apenas na plataforma de hospedagem *GitHub*, essas buscas podem ser estendidas para os repositórios presentes em sistemas como o *GitLab* e o *Bitbucket*. Isso limita a área de atuação dos usuários que estejam buscando por uma variedade maior de repositórios.

- Uma deficiência é o pequeno escopo de pesquisa que, por mais que conte com parâmetros que não existem na busca avançada do *GitHub*, ainda é muito limitado quanto à variedade de parâmetros que podem ser encontrados na busca avançada e na API do *GitHub*.

- Para garantir que as informações dos repositórios de uma pesquisa estejam sempre atualizadas, a ferramenta realiza as pesquisas em tempo real, fazendo com que, para cada nova pesquisa, seja necessário o consumo da *GitHub API*. Sendo assim, certas pesquisas podem levar horas, dias ou até semanas para serem realizadas, demandando que o usuário espere um tempo elevado para a obtenção de seus repositórios.

- O tempo de execução da ferramenta pode ser um problema caso o número de núcleos da CPU seja inferior a 4, pois a ferramenta particiona o trabalho de consumo das tarefas das filas em relação ao número de *cores*, fazendo assim com o que o trabalho ocorra de maneira assíncrona e paralela.

- A ferramenta foi implementada considerando a versão atual do *GitHub* e de sua API. Caso haja alteração nos metadados ou no funcionamento da API V4 do *GitHub*, a ferramenta terá que passar por um processo de adaptação para acomodar as novas configurações.

- Como a implementação da fragmentação de busca dos repositórios funciona sobre as características de quantidade de estrelas, data da criação e tamanho, pode ocorrer que a fragmentação sobre esses 3 critérios não seja o suficiente para retornar

resultados menores que 1000 repositórios, sendo então necessário a adição de novos critérios de fragmentação mediante uma refatoração do código.

5.4 Considerações Finais

O capítulo se inicia apresentando quais tecnologias foram utilizadas para implementar a abordagem, como as linguagens de programação, *frameworks* e banco de dados. Logo após, foi definido um exemplo de utilização, que passou pelas principais funcionalidades da abordagem, entre elas: a realização da pesquisa e visualização de seus resultados, a criação de critérios, a vinculação destes critérios com os repositórios, e a filtragem dos repositórios mediante a definição dos critérios entre critérios de inclusão ou de exclusão. Por fim, são descritas as limitações encontradas na implementação da ferramenta, como o pequeno escopo de campos de busca e o longo tempo de demora para se realizar uma pesquisa com muitos repositórios.

6 Validação da abordagem

Como experimento avaliativo para este trabalho foi refeito um experimento realizado em 2020 no trabalho de Romano et al. (2021), tal experimento obteve 1500 repositórios do *GitHub*. Após replicá-lo, comparou-se o conjunto de 1500 repositórios obtidos no experimento deste trabalho com os 1500 repositórios do experimento comparado, com o objetivo de verificar quais mudanças ocorreram entre os repositórios e levantar uma discussão a respeito da reprodutibilidade de experimentos.

Este capítulo detalha como foi conduzida a avaliação da abordagem e quais foram os resultados obtidos. A Seção 6.1 descreve como foi selecionada a pesquisa replicada no experimento e como ela foi conduzida pelos autores. Na Seção 6.2 está descrito como o experimento base foi reproduzido por este trabalho, e como os repositórios resultantes de ambos os experimentos foram comparados e, por fim, o recurso computacional empregado para a realização do experimento. Na Seção 6.3 são definidas as questões que guiaram a pesquisa. Na Seção 6.4 são respondidas às questões de pesquisa. A Seção 6.5 promove uma discussão quanto as implicações que os resultados podem ter para pesquisas que envolvam seleção de repositórios de software. Na Seção 6.6 são elencados aspectos que podem ameaçar a validade do experimento proposto. Por fim, a Seção 6.7 traz um resumo dos principais pontos discutidos no capítulo.

6.1 Experimento base

Com o intuito de avaliar a abordagem implementada, buscou-se na literatura por um experimento que busque por repositórios do *GitHub* através dos metadados (*e.g., stars*, *commits*) e fornecesse os repositórios obtidos. A amostra de artigos utilizada foi o conjunto de artigos obtidos através da revisão sistemática, detalhada no Apêndice A, pois tais artigos explicitam diversas ferramentas que facilitam a seleção de repositórios de software. Dentre tais artigos, apenas o trabalho de Romano et al. (2021) atendia aos critérios necessários citados no início desta seção. Nesse trabalho encontrado foi realizado um ex-

perimento para validar a funcionalidade de detecção da linguagem falada dos repositórios da ferramenta *G-repo* e, para obter tais repositórios, buscou por repositórios do *GitHub* que possuíssem *Java* como linguagem de programação principal e mais de 1.000 *estrelas*. Tal pesquisa foi realizada em 07 de agosto de 2020.

Os autores não indicam quantos repositórios foram obtidos após essa pesquisa na ferramenta, mas para o experimento final, eles realizaram um corte nesses repositórios resultantes para obter apenas os 1500 repositórios com mais estrelas, estes compreendem repositórios de diversas áreas, tal qual repositórios que armazenam frameworks como o *Spring*[2], repositórios de softwares como o *NextCloud*[3] e repositórios didáticos como o *Algorithms*[4]. A planilha resultante com esses 1500 repositórios pode ser encontrada neste link[5].

Na planilha de 1.500 repositórios disponibilizada pela pesquisa de Romano et al. (2021), foi encontrado 2 repositórios que se repetem, um deles aparece 3 vezes e outro 2 vezes, mas o motivo da repetição é desconhecida (*e.g.*, *RxJava-Android-Samples*[6], *android-tips-tricks*[7]). Após a identificação deste problema, é possível inferir que há um total de 1.497 repositórios únicos na planilha. Mas para facilitar a comparação entre os 2 experimentos, foi assumido que essa planilha possuia 1.500 repositórios.

6.2 Design do Experimento

O intuito deste capítulo é realizar a mesma pesquisa na ferramenta aqui proposta, validando o funcionamento da abordagem e trazendo uma análise comparativa entre os resultados, buscando também responder às questões de pesquisa levantadas em 6.3.

A pesquisa realizada por este trabalho em 08 de maio de 2024, buscou por repositórios que possuíssem *Java* como linguagem de programação principal e mais de 1000 *estrelas*, os mesmos critérios utilizados no experimento base. Após feito a busca, foram obtidos 3.443 repositórios do *GitHub*, estes foram ordenados pela quantidade de *estrelas* e

[2]⟨https://github.com/spring-projects/spring-framework⟩
[3]⟨https://github.com/nextcloud/android⟩
[4]⟨https://github.com/pedrovgs/Algorithms⟩
[5]⟨https://github.com/MatHeartGaming/G-Repo/tree/master/manual_classification⟩
[6]⟨https://github.com/kaushikgopal/RxJava-Android-Samples⟩
[7]⟨https://github.com/nisrulz/android-tips-tricks⟩

os primeiros 1.500 resultados foram selecionados, tendo todos eles mais de 2.237 *estrelas*[8].

O experimento foi realizado em uma máquina com a seguinte configuração de hardware: a CPU foi um AMD Ryzen 5 5600, a GPU se trata de uma NVIDIA RTX 4060, com 32Gb de memória RAM e Windows 11 de sistema operacional.

6.3 Questões de pesquisa

Para avaliar a abordagem, a pesquisa original foi nomeada de experimento base, e a pesquisa replicada por este trabalho nomeada de experimento derivado. Com isso, algumas questões de pesquisa foram levantadas, com intuito de serem respondidas através da análise dos resultados obtidos na busca. As questões e as motivações por trás delas são apresentadas a seguir:

QP1 — O conjunto de repositórios do experimento base contém os mesmos repositórios do experimento derivado?

Esta questão visa descobrir se os mesmos repositórios foram obtidos em ambos os experimentos. A resposta "sim" mostraria que ambos os experimentos retornaram os mesmos repositórios e que houve uma reprodutibilidade, a resposta "não" mostraria que ambos os experimentos retornaram diferentes repositórios e que não houve reprodutibilidade.

QP2 — Quais foram os repositórios que não estavam no experimento base que entraram no experimento derivado? Quais os motivos?

Está questão visa identificar quais repositórios foram adicionados aos resultados do experimento derivado e compreender os motivos que causaram essa mudança. Com isso, é possível analisar quais alterações tais repositórios sofreram ao longo do tempo (*e.g.,* aumento da quantidade de *estrelas*).

QP3 — Quais foram os repositórios que estavam no experimento base que não se encontram no experimento derivado? Quais os motivos?

Está questão objetiva compreender quais repositórios que antes se encontravam no experimento base e agora não mais estão no experimento derivado, assim como buscar os motivos que fizeram com que tal mudança ocorresse. Com isso, pode analisar-se quais

[8]⟨https://github.com/Guima01/Validacao-da-Abordagem/blob/main/Resultados_Abordagem.ods⟩

alterações tais repositórios sofreram ao longo do tempo para saírem dos resultados da busca (*e.g.,* Java deixou de ser a linguagem de programação principal).

QP4 — Houve repositórios que sofreram modificações em sua URL do experimento base para o experimento derivado? Quais os motivos?

Está questão visa identificar quais repositórios se mantiveram entre as pesquisas, mas sofreram alterações que dificultaram o reconhecimento entre os resultados, como alteração no nome dos repositórios ou no nome do usuário. Com isso, é possível distinguir quais repositórios se mantiveram nos resultados, mas que se apresentaram distintamente entre os resultados dos experimentos.

6.4 Resultados

Através da comparação entre os 1.500 repositórios do experimento base com os 1.500 do experimento derivado, foram obtidos 3 grupos distintos de resultados.

O primeiro agrupa os repositórios que sofreram alterações em sua URL, mas que se mantiveram entre os resultados dos experimentos. Para encontrá-los foi necessário acessar a URL de todos os repositórios do experimento base e verificar se houve redirecionamento para uma nova URL. Cada URL modificada foi buscada nos repositórios do experimento derivado e caso ela tenha sido encontrada, foi considerada um repositório que sofreu alteração em sua URL.

O segundo agrupa os repositórios não encontrados no experimento derivado e que estão no experimento base. Para encontrá-los foi necessário buscar todos os repositórios que não sofreram redirecionamento na URL e buscá-los nos resultados do experimento derivado, os que não foram encontrados entraram neste grupo. Junto a esses repositórios já agrupados, foram adicionados também todos os repositórios que sofreram modificações na URL, mas que também não foram encontrados nos resultados do experimento derivado.

Por fim, o terceiro agrupa os repositórios encontrados no experimento derivado e que não estão no experimento base. Para obtê-los foi necessário buscar a URL de todos os repositórios do experimento derivado nos resultados do experimento base, selecionando todos os que não foram encontrados e excluindo os repositórios que tiveram as URL modificadas. Com estes grupos definidos, as questões levantadas podem ser sanadas.

QP1 — O conjunto de repositórios do experimento base contém os mesmos repositórios do experimento derivado?

O conjunto de repositórios obtidos não foi o mesmo, o que indica que não houve uma reprodutibilidade do experimento, sendo reprodutibilidade definida neste contexto como a capacidade de novos pesquisadores realizarem o mesmo experimento com a mesma configuração experimental e obter os mesmos resultados Tutko, Henley e Mockus (2022). Houve um total de 349 repositórios que saíram dos resultados ao longo destes 4 anos, sendo tal saída dada pelos mais diversos motivos, e por consequência, novos 349 repositórios entraram nos resultados. O Gráfico 6.1 mostra a relação entre os repositórios inseridos e mantidos de um experimento para o outro. A barra "Total" representa o total de repositórios resultantes, a barra "Mantidos" mostra a quantidade de repositórios que estão tanto no experimento base quanto no experimento derivado, a barra "Inseridos" mostra quantidade de novos repositórios obtidos apenas no experimento derivado.

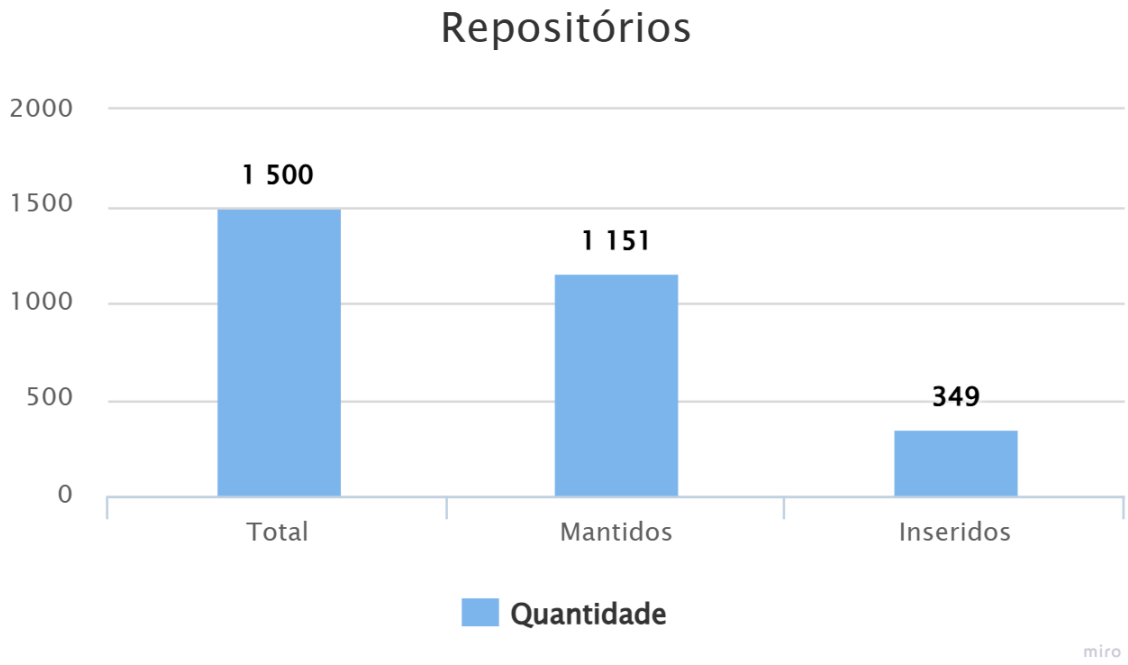

Figura 6.1: Relação entre repositórios alterados e que se mantiveram no experimento derivado

Através da análise da planilha, é perceptível que um total de 23,27% do total de repositórios resultantes do experimento derivado são novos repositórios inseridos, enquanto os 76,73% restantes eram repositórios também do experimento base.

QP2 — Quais foram os repositórios que não estavam no experimento base que entraram no experimento derivado? Quais os motivos?

Houve 349 repositórios que obtiveram ao longo destes 4 anos as características necessárias para entrarem nos resultados [9]. Para entender o motivo pelo qual tais repositórios entraram neste escopo, foram selecionados aleatoriamente uma amostra de 10 repositórios e aplicado neles a biblioteca *github-linguist* sobre o *commit* mais perto da data da realização do experimento base, biblioteca utilizada pelo próprio *GitHub* para calcular a porcentagem de linguagens de um repositório. Para obter a quantidade de *estrelas* na data de tal *commit*, foi utilizado a ferramenta *daily-stars-explorer*[10]. Tais ferramentas são necessárias, pois o *GitHub* não disponibiliza as linguagens de programação nem a quantidade de *estrelas* de um repositório em um determinado *Commit*.

Foi selecionado apenas essa pequena amostra de 10 repositórios por demandar um grande intervalo de tempo para obter as informações através do uso dessas ferramentas. Para obter a porcentagem de linguagem de cada repositório, por exemplo, foi necessário realizar o *Clone* de cada um dos 10 repositórios, alterar a *branch* para o *commit* mais próximo da data do experimento base, e adicionar o repositório ao *GitHub*, fazendo com que ele calculasse as linguagens. Por outro lado, o uso da ferramenta *daily-stars-explorer* é um trabalho mais fácil, por só ser necessário adicionar a URL do repositório na ferramenta, e ela busca todo o histórico de *estrelas* do repositório.

A Tabela 6.1 explicita os repositórios aleatórios obtidos. A coluna "Repositório" indica o nome do repositório e a coluna "URL" indica o link do repositório no *GitHub*.

[9] ⟨https://github.com/Guima01/Validacao-da-Abordagem/blob/main/Repositorios_Inseridos.ods⟩
[10] ⟨https://emanuelef.github.io/daily-stars-explorer/⟩

Tabela 6.1: repositórios e suas respectivas URLs

Repositório	URL
megabasterd	⟨https://github.com/tonikelope/megabasterd⟩
JavaSourceCodeLearning	⟨https://github.com/coderbruis/JavaSourceCodeLearning⟩
supertokens-core	⟨https://github.com/supertokens/supertokens-core⟩
iceberg	⟨https://github.com/apache/iceberg⟩
ShadowLayout	⟨https://github.com/lihangleo2/ShadowLayout⟩
OnJava8-Examples	⟨https://github.com/BruceEckel/OnJava8-Examples⟩
Lbry-android	⟨https://github.com/lbryio/lbry-android⟩
ZxingLite	⟨https://github.com/jenly1314/ZXingLite⟩
novel-plus	⟨https://github.com/201206030/novel-plus⟩
pf4j	⟨https://github.com/pf4j/pf4j⟩

A Tabela 6.2 explicita os resultados obtidos na análise dos repositórios no momento da realização do experimento base. A coluna "Repositório" indica o nome do repositório. A coluna "Porcentagem da linguagem *Java*" indica a porcentagem de código *Java* encontrada na data de execução do experimento base. A coluna "Número de estrelas" indica a quantidade de estrelas encontrada na data de execução do experimento base.

Tabela 6.2: Metadados dos repositórios selecionados na data de execução do experimento base

Repositório	Porcentagem da linguagem *Java*	Número de estrelas
megabasterd	100%	837
JavaSourceCodeLearning	100%	87
supertokens-core	99.2%	52
iceberg	89.6%	576
ShadowLayout	100%	880
OnJava8-Examples	99.2%	1126
Lbry-android	99.9%	419
ZxingLite	51.6%	1357
novel-plus	96.1%	281
pf4j	99.0%	1117

É perceptível que, dentre os 10 repositórios, todos eles já possuíam a linguagem *Java* como a principal e isso se manteve até a pesquisa mais recente, dando a entender que o motivo pelo qual esses repositórios entraram nos resultados foi porque alcançaram o ranking de 1.500 repositórios mais populares, dado que 7 deles nem possuíam mais de 1.000 estrelas na data do 1º experimento e os outros 3 já possuíam, porém, não se encontravam entre os 1.500 repositórios com mais estrelas.

A Tabela 6.3 explicita os resultados obtidos na análise dos repositórios no momento da realização do experimento derivado. A coluna "Repositório" indica o nome do repositório. A coluna "Porcentagem da linguagem *Java*" indica a porcentagem de código *Java* encontrada na data de execução do experimento derivado. A coluna "Número de estrelas" indica a quantidade de estrelas encontrada na data de execução do experimento derivado.

Tabela 6.3: Metadados dos repositórios selecionados na data de execução do experimento derivado

Repositório	Porcentagem da linguagem *Java*	Número de estrelas
megabasterd	100%	4267
JavaSourceCodeLearning	100%	3363
supertokens-core	98.5%	11957
iceberg	96.9%	5552
ShadowLayout	99.4%	3323
OnJava8-Examples	98.4%	3006
Lbry-android	99.9%	2449
ZxingLite	92.8%	2924
novel-plus	97.7%	3531
pf4j	99.5%	2244

Ao comparar a tabela 6.3 com a tabela 6.2 é perceptível que houve um crescimento na quantidade de *estrelas* dos repositórios, o repositório *supertokens-core* passou do repositório com menos *estrelas* para o com maior quantidade, tendo um crescimento de quase 230 vezes, diferente do *ZxingLite* que teve aumento de *estrelas* de pouco mais de 2 vezes. Outro ponto perceptível, é que 9 dos 10 repositórios tiveram pouca variação na porcentagem da linguagem *Java*, tendo uma margem de variação menor que 10%, apenas o repositório *ZxingLite* teve uma mudança brusca, passando de 51.6% no experimento base para 92.8% no experimento derivado.

QP3 — Quais foram os repositórios que estavam no experimento base e que não se encontram no experimento derivado? Quais os motivos?

Houve um total de 349 repositórios que saíram dos resultados do experimento base[11], 5 deles são repositórios no qual o usuário não existe mais ou não pode ser acessado (*e.g., cropper*[12], *jjdxm_ijkplayer*[13], *jianshi*[14], *SpringIndicator*[15], *ZXBlog*[16]). Também há 5 repositórios que podem ter sido excluídos ou não estão mais acessíveis, mas seus usuários continuam acessíveis (*e.g., blynk-server*[17], *incubator-heronn*[18], *JamsMusicPlayer*[19], *liugh-parent*[20], *nd4j*[21]).

Além dos repositórios supracitados, e como já comentado em 6.1, houveram 2 repositórios repetidos nos resultados do experimento base. E por escolhas de implementação da abordagem desenvolvida, já comentadas em 5.2, houve um único repositório que não foi obtido por não possuir mais que 0 KB de tamanho, sendo considerado, assim, irrelevante para a validação.

Os restantes dos repositórios que não se encontram no experimento derivado, saíram devido à perda de pelo menos um dos critérios de busca, ou seja, não possuem mais *Java* como linguagem principal ou não estão entre os 1500 repositórios com mais *estrelas*. O diagrama 6.2 representa a quantidade de repositórios removidos devido a cada uma das características.

[11] ⟨https://github.com/Guima01/Validacao-da-Abordagem/blob/main/Repositorios_Removidos.ods⟩
[12] ⟨https://github.com/edmodo/cropper⟩
[13] ⟨https://github.com/lingcimi/jjdxm_ijkplayer⟩
[14] ⟨https://github.com/wingjay/jianshi⟩
[15] ⟨https://github.com/chenupt/SpringIndicator⟩
[16] ⟨https://github.com/ZXZxin/ZXBlog⟩
[17] ⟨https://github.com/blynkkk/blynk-server⟩
[18] ⟨https://github.com/apache/incubator-heronn⟩
[19] ⟨https://github.com/psaravan/JamsMusicPlayer⟩
[20] ⟨https://github.com/qq53182347/liugh-parent⟩
[21] ⟨https://github.com/deeplearning4j/nd4j⟩

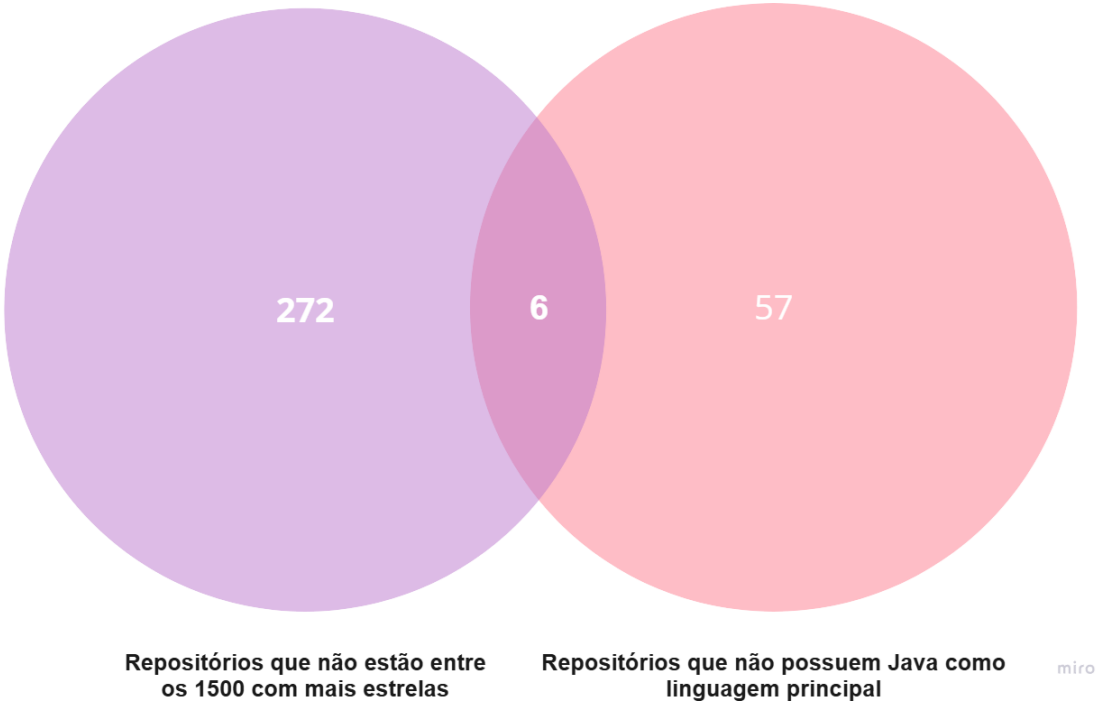

Figura 6.2: Quantidade de repositórios removidos que não seguiam pelo menos uma característica da pesquisa

Foram removidos 57 repositórios que não possuem mais *Java* como linguagem principal. Destes, 29 passaram a ter *Kotlin* como principal, sendo que em 2 repositórios, o *Kotlin* substituiu completamente qualquer traço de *Java*, constituindo 100% do código. Além disso, os repositórios restantes tiveram as seguintes alterações de linguagem principal: 1 para *C++*, 1 para *CSS*, 1 para *Dart*, 1 para *Jupyter Notebook*, 1 para *MDX*, 1 para *Python*, 1 para *Vue*, 1 para *Ballerina*, 2 repositórios para *C*, 2 para *Go*, 2 para *HTML*, 3 para *JavaScript*, 3 para *TypeScript* e 8 repositórios sem linguagem alguma. Vale ressaltar que o *GitHub* classifica linguagens não apenas como linguagem de programação, por isso, resultados como *CSS* e *Jupyter Notebook* aparecem.

Dentre esses repositórios sem linguagem, 5 possuíam arquivos *Java* no *commit* mais próximo à realização do experimento base, mas no experimento derivado não possuem nenhum (*e.g.*, *PLDroidPlayer*[22]). Um repositório possui arquivos *Java* na *Branch Master*, porém esta não mais é a *Branch* principal do repositório. Dois repositórios possuem arquivos *Java*, mas não constam como linguagem do repositório no *GitHub*.

A quantidade de *estrelas* é o principal responsável pela saída de repositórios.

[22] ⟨https://github.com/pili-engineering/PLDroidPlayer⟩

Um total de 272 repositórios não estão entre os 1500 mais populares. Dentre esses 272 repositórios, o repositório *Drools*[23] possui quantidade de *estrelas* menor do que 1000 atualmente, porém, ao utilizar o serviço *daily-stars-explorer*, percebeu-se que ele sempre esteve abaixo de 1000 *estrelas* e que se trata de um *fork* do repositório *incubator-kie-drools*[24]. Uma hipótese é que o repositório *Drools* tenha se transformado no repositório *incubator-kie-drools* e o original se tornou um *fork* desta alteração.

Há também os repositórios que saíram por não terem mais *Java* como linguagem principal e por não estarem entre os repositórios com mais *estrelas*. Seis repositórios foram encontrados, 2 com a linguagem *Kotlin*, 1 com *HTML*, e 3 sem linguagem alguma. Desses 3 sem linguagem, 2 estão com menos que 1000 *estrelas* na busca, com um deles contendo apenas 3 *estrelas* (*e.g.*, *Jersey*[25]) e outro contendo 1 *estrela* (*e.g.*, *android-bootstrap*[26]). O repositório Jersey, segundo a própria descrição, não é mais o repositório ativo que armazena o *framework* REST de mesmo nome, podendo ser esse o motivo de o repositório ter perdido suas *estrelas*. Já o repositório *android-bootstrap* não possui nenhuma informação da qual é possível identificar o motivo para a perda de suas *estrelas*.

QP4 — Houve repositórios que sofreram modificações em sua URL do experimento base para o experimento derivado? Quais os motivos?

Houve um total 83 repositórios tiveram modificações em seu nome ou no nome de usuário ao longo do tempo[27], fazendo com que fossem encontrados na nova pesquisa com uma URL distinta. A Tabela 6.4 apresenta 3 exemplos de repositórios modificados entre os experimentos. A primeira coluna identifica a URL do repositório presente nos resultados do experimento base e a segunda coluna identifica a URL do repositório presente nos resultados do experimento derivado.

Tabela 6.4: Exemplos de repositórios modificados entre os experimentos

URL no experimento base	URL no experimento derivado
github.com/jhalterman/failsafe	github.com/failsafe-lib/failsafe
github.com/YunaiV/onemall	github.com/YunaiV/yudao-cloud
github.com/graknlabs/grakn	github.com/vaticle/typedb

[23] ⟨https://github.com/kiegroup/drools⟩
[24] ⟨https://github.com/apache/incubator-kie-drools⟩
[25] ⟨https://github.com/jersey/jersey⟩
[26] ⟨https://github.com/AndroidBootstrap/android-bootstrap⟩
[27] ⟨https://github.com/Guima01/Validacao-da-Abordagem/blob/main/Repositorios_Editados.ods⟩

A tabela 6.4 traz 3 exemplos distintos de modificações, a primeira linha identifica o repositório *failsafe* que teve apenas o nome do usuário dono do repositório alterado de *jhalterman* para *failsafe-lib*. A segunda linha traz o repositório *yudao-cloud* que no experimento base estava com o nome de *onemall*. Por fim, a última linha identifica o repositório *typedb* do usuário *vaticle*, esse repositório no experimento base estava com o nome de *grakn* enquanto seu usuário estava nomeado como *graknlabs*.

6.5 Discussão

Com os resultados obtidos a partir da validação e tendo respondido às questões de pesquisa, é possível discutir as implicações que tais resultados podem ter para os pesquisadores.

O experimento de Romano et al. (2021) obteve um total de 1500 repositórios para seu experimento. Se houvesse uma tentativa de replicar essa mesma pesquisa, os usuários encontrariam 349 novos repositórios em relação ao experimento base, não encontrariam 349 repositórios que estava presente no experimento base e encontrariam 88 repositórios que sofreram alterações de URL entre os dois experimentos. Isso poderia acarretar um possível desgaste para identificar que mudanças ocorreram dentre os repositórios de ambos os experimentos. Portanto, manter o registro dos parâmetros de pesquisa, dos repositórios obtidos e de seus metadados, é fundamental para garantir a reprodutibilidade de uma pesquisa de repositórios de software.

Através do registro das informações da pesquisa base é possível tentar reproduzi-lo em uma nova pesquisa. Pois através das informações dos parâmetros iniciais, pesquisadores podem conduzir uma nova pesquisa com tais parâmetros, e através dos registros dos repositórios obtidos, os pesquisadores podem verificar se os mesmos repositórios foram obtidos entre as pesquisas, e quais mudanças ocorreram entre seus metadados, assim como ocorreu com os experimentos comparados nesse capítulo.

6.6 Ameaças à Validade

Nesta seção, serão discutidos os pontos que podem limitar os resultados da validação realizada neste capítulo. Dentre os pontos de ameaça a validade levantados, é possível categorizá-los em 2 tipos, ameaças externas e ameaças de conclusão.

A utilização de apenas um estudo para comparar os resultados do experimento pode ser considerada uma ameaça à validade externa, pois ela compromete a generalização dos resultados do experimento (TRAVASSOS; GUROV; AMARAL, 2002). Ao buscar por trabalhos que incluíssem uma seleção de repositórios do *GitHub*, apenas uma pesquisa que atendia este critério foi encontrada. Por conta disso, a validação ficou restrita aos parâmetros utilizados por essa pesquisa, especificamente repositórios da linguagem *Java* com mais de 1000 estrelas, sem, então, explorar outros possíveis campos oferecidos pela *GitHub API*, como a quantidade de *forks* e data de criação, ou até mesmo campos implementados exclusivamente na abordagem deste trabalho, como a quantidade de *commits*.

A pequena amostra de 10 repositório para definir uma resposta de como os repositórios removidos se comportaram, e a presença de repositórios repetidos na planilha resultante do experimento base são consideradas ameaças à validade de conclusão. Esse tipo de ameaça envolve a precisão com que as conclusões são tiradas a partir dos dados Travassos, Gurov e Amaral (2002).

6.7 Considerações Finais

O capítulo descreve a metodologia e os resultados de um experimento que utiliza a *GitHub API v4* para obter repositórios do *GitHub*, comparando com resultados de um experimento anterior realizada em 2020 com a *GitHub API v3*. O estudo se concentra em repositórios de *Java* com mais de 1000 estrelas, analisando mudanças nos repositórios ao longo de quatro anos. As principais questões abordadas pela pesquisa são as entradas e saídas de repositórios nos resultados e os motivos dessas mudanças.

A comparação mostrou que 83 repositórios tiveram alterações de nome ou usuário, 349 novos repositórios entraram nos resultados do experimento derivado por alcançarem

mais de 1000 *estrelas*, terem *Java* como linguagem principal e por ultrapassarem outros 349 repositórios do experimento base. Esses outros 349 repositórios substituídos, não entraram no experimento derivado por não estarem mais entre os 1500 repositórios com mais *estrelas* ou por não terem mais linguagem *Java* como principal ou por outros motivos distintos, como a exclusão do repositório e a inacessibilidade do perfil dos usuários.

A discussão destaca a importância de manter registros detalhados dos metadados dos repositórios para garantir a reprodutibilidade das pesquisas. O capítulo também aborda as limitações da validação, como a dependência de uma única pesquisa encontrada para replicar o experimento.

Em suma, o estudo promove a funcionalidade da abordagem, verifica como um conjunto de repositórios pode mudar ao longo do tempo e como é importante manter o registro dos repositórios utilizados em pesquisas para garantir a reprodutibilidade dos resultados.

7 Conclusões

Este trabalho apresentou uma abordagem para apoiar estudos na área de Mineração de Repositórios de Software (MSR), focando na seleção de repositórios.

A implementação da ferramenta permite a definição de parâmetros alinhados com os objetivos de pesquisa do usuário, a coleta de dados de plataformas de hospedagem e a superação de limitações das ferramentas tradicionais. Com o intuito de permitir o acesso a repositórios relevantes e permitir a rastreabilidade das decisões tomadas durante o processo de seleção.

Os resultados iniciais demonstraram que a abordagem proposta consegue superar as restrições impostas por certas ferramentas como a *GitHub API*, como o limite de 1000 repositórios por busca, e oferece novas funcionalidades como novos campos de busca e a criação de critérios para caracterizar e filtrar os repositórios. Além disso, a capacidade de registrar o histórico de decisões do usuário contribui significativamente para a reprodutibilidade das pesquisas.

A validação da abordagem, realizada por meio da replicação de um experimento de um trabalho da literatura que buscava por repositórios do *GitHub* com certas características, revelou mudanças significativas nos repositórios ao longo do tempo, destacando a importância de manter registros detalhados dos repositórios.

Para trabalhos futuros, vislumbra-se a expansão da ferramenta para incluir repositórios de outras plataformas, como *GitLab* e *Bitbucket*, além da adição de novos parâmetros de busca que possam enriquecer ainda mais a experiência do usuário. A integração da ferramenta em uma plataforma Web acessível ao público também é uma perspectiva promissora, permitindo que mais pesquisadores se beneficiem das funcionalidades desenvolvidas.

Em suma, este trabalho contribui para a área de MSR e estabelece um caminho para futuras investigações que busquem aprimorar a seleção de repositórios de software, promovendo um avanço significativo na qualidade e na reprodutibilidade das pesquisas.

A Apêndice - Literatura sobre seleção de repositórios para mineração de software

Visando realizar uma revisão bibliográfica dos estudos de seleção de repositórios relacionados a MSR, foi feita uma revisão da literatura baseada na abordagem híbrida proposta por Mourão et al. (2020). A qual realiza buscas por trabalhos relacionados em bibliotecas digitais e, através dos trabalhos resultantes, realiza uma iteração da técnica de *Backward Snowballing* (busca por trabalhos na lista de referências), e uma iteração de *Forward Snowballing* (Busca nos trabalhos que o citam), com o intuito de refinar os resultados.

Para atingir o objetivo proposto, conduziu-se uma pesquisa na biblioteca digital *Scopus*. Essa base foi escolhida baseando-se nos estudos de Mourão et al. (2020) que demonstra, por meio de experimentos práticos, que a utilização da *Scopus* oferece uma maior representatividade na obtenção de resultados. Resumidamente, o uso exclusivo da *Scopus* resulta em aproximadamente 75% dos resultados esperados. Em contraste, a utilização somente da *Compendex* identifica cerca de 65% dos resultados, enquanto o uso isolado da *IEEEXlore* retorna apenas 42% dos resultados.

Para efetuar o uso da plataforma *Scopus*, desenvolveu-se uma *string* de busca, aderindo à sintaxe de busca da plataforma *Scopus*. Essa *string* foi elaborada para buscar artigos relativos às áreas de "Seleção de repositórios de software" e de "Mineração de repositórios de software". Para tal, incorpora palavras-chave relacionadas ao assunto, com o intuito de limitar ao máximo a obtenção de resultados irrelevantes. Conforme o estudo de Barreto, Murta e Rocha (2012), o primeiro passo na construção de uma *string* de busca envolve a definição dos parâmetros de população, intervenção, comparação e resultado.

- População (Population): Pesquisas relacionadas a seleção, mineração e armazenamento de repositórios e projetos de software.
- Intervenção (Intervention): Abordagens vinculadas a área de MSR.

- Comparação (Comparison): Não há, pois não pretende comparar abordagens, e sim, caracterizá-las.

- Saída (Output): Técnicas, Abordagens, Métodos, Metodologias ou Ferramentas de seleção de repositórios de software voltado para a área de MSR.

O próximo passo, determinou as expressões para a população, a intervenção e a saída. Com relação ao termos da população, foi utilizado os que trouxessem resultados relacionados a seleção de repositórios, como "selection repositories" ou "search repositories". Os asteriscos nos termos (*) permitem definir variações, por exemplo, "select*" pode buscar por termos como "selection" ou "selected". Por fim, para calibrar a *string*, foram definidos sinônimos e termos utilizados na literatura. A expressão de intervenção está relacionada a área de mineração de repositórios de software, então utilizou o termo em inglês e sua sigla. Com relação à expressão de saída, foi definido os termos em inglês de técnicas, metodologias, ferramentas, e seus sinônimos.

- População: ("select* repositor*" OR "search* repositor*" OR "min* repositor*" OR ("mining" AND "data" AND "software repositor*") OR "sampl* project*" OR "sampl* repositor*" OR "creat* of dataset*"); e

- Intervenção: ("MSR" OR "Mining Software Repositories").

- Saída (Output): ("approach"OR "method"OR "methodology"OR "procedure"OR "mechanism"OR "findings"OR "research"OR "study"OR "technique"OR "knowledge"OR "tool"OR "support"OR "investigat*").

A partir da união das 3 expressões com o operador AND, foi obtido a expressão final A.1 utilizada na plataforma *Scopus*.

TITLE-ABS ((“select* repositor*”OR “search* repositor*”OR “min* repositor*”OR (“mining”AND “data”AND “software repositor*”) OR “sampl* project*”OR “sampl* repositor*”OR “creat* of dataset*”) AND (“MSR”OR “Mining Software Repositories”) AND (“approach”OR “method”OR “methodology”OR “procedure”OR “mechanism”OR “findings”OR “research”OR “study”OR “technique”OR “knowledge”OR “tool”OR “support”OR “investigat*”)) AND (LIMIT-TO (SUBJAREA,“COMP”)) AND (LIMIT-TO (LANGUAGE,“English”) OR LIMIT-TO (LANGUAGE,“Portuguese”))

(A.1)

Os resultados foram filtrados para considerar a busca apenas no título e no abstract dos artigos, considerando apenas os idiomas português e inglês, e se estão na área da computação. Para buscar as palavras-chave apenas no título ou no abstract dos artigos, foi utilizada a cláusula “TITLE-ABS” envolvendo a *string* de busca. Para filtrar artigos na área da computação foi utilizada a cláusula “LIMIT-TO (SUBJAREA, “COMP”)”. Para limitar os resultados ao inglês e ao português, foi adicionado as cláusulas “LIMIT-TO (LANGUAGE, “English”) OR LIMIT-TO (LANGUAGE, “Portuguese”)”.

Após a realização da busca na plataforma Scopus, completou-se a primeira fase dos procedimentos de seleção. A pesquisa inicial gerou um total de 169 artigos em 27 de abril de 2024. Tendo esses resultados em mãos, o próximo passo foi a seleção de publicações pertinentes. Desta forma, os seguintes critérios de exclusão foram adotados:

- CE1 – Publicações que não se relacionavam com o tema desejado, como, publicações que não abordavam a etapa de busca/obtenção de repositórios de software; e
- CE2 - Publicações não disponíveis para download de forma completa nas bibliotecas digitais nem mediante nenhum outro meio sem custos para o pesquisador; e
- CE3 – Publicações que descrevem e/ou apresentam *keynote speeches*, tutoriais, cursos, workshops e similares.

Visto que os critérios de busca aplicados não asseguram automaticamente a relevância de todas as publicações encontradas para o contexto da pesquisa, foi realizado um processo de seleção baseado em duas etapas, com intuito de refinar ainda mais a seleção dos artigos. A 1ª etapa visa a leitura do título e resumos (abstracts) de cada um dos trabalhos. Caso o artigo não esteja alinhado com o objetivo da pesquisa, ele é eliminado de acordo com um dos critérios de exclusão. Nos casos em que a análise do resumo não for o suficiente, uma 2ª etapa é necessária, visando a leitura do artigo inteiro. Se após a leitura dos artigos, eles não estiverem alinhados com a pesquisa, também serão excluídos de acordo com os critérios de exclusão.

A Tabela A.1, demonstra o estado final dos artigos obtidos, após a realização da leitura e do processo de exclusão dos mesmos. Ela está organizada em 3 colunas, a coluna de "Artigos" demonstra o título do artigo, a coluna de "1ª etapa" revela se houve a exclusão do artigo baseado na 1ª etapa do processo de seleção, e a coluna de "2ª etapa" revela se houve a exclusão do artigo baseado na 2ª etapa do processo de seleção. Para ambas as colunas de etapas, caso o artigo tenha sido aceito, será assinalado um OK na coluna, caso não, será assinalado por qual critério se deu sua exclusão.

Um exemplo de leitura pode ser visto na seguinte linha, a coluna de "Artigos" traz o nome do artigo "Modelmine: A tool to facilitate mining models from open source repositories". A coluna "1ª etapa" possui um OK, pois o artigo não foi excluído na etapa de leitura do título e do resumo. A coluna "2ª etapa" também possui um OK, sendo assim, o artigo foi um dos selecionados dos resultantes finais no processo de obtenção dos artigos.

Artigos	1ª etapa	2ª etapa
Modelmine: A tool to facilitate mining models from open source repositories	OK	OK

Tabela A.1: Artigos excluídos

Artigos	1ª etapa	2ª etapa
Characterizing Commits in Open-Source Software	C2	
Mining Software Repositories for the Characterization of Continuous Integration and Delivery	C2	
Mining software repositories to identify library experts	C2	
Software mining studies: Goals, approaches, artifacts, and replicability	C2	
Generating duplicate bug datasets	C2	
Model-based mining of source code repositories	C2	
Boa: An Enabling Language and Infrastructure for Ultra-Large-Scale MSR Studies	C2	
Extracting enhanced artificial intelligence model metadata from software repositories	C2	
SysRepoAnalysis: A tool to analyze and identify critical areas of source code repositories	C2	
Towards mining norms in open source software repositories	C2	
Argo: Projects' Time-Series Data Fetching and Visualizing Tool for GitHub	C2	
Latent Dirichlet Allocation: Extracting Topics from Software Engineering Data	C2	

Artigos	1ª etapa	2ª etapa
Mining rational team concert repositories: A case study on a software project	C2	
Emerging topics in mining software repositories: Machine learning in software repositories and datasets	C2	
Poster: Understanding newcomers success in open source community	C2	
Exploring social contagion in open-source communities by mining software repositories	C2	
Imprecisions diagnostic in source code deltas	C2	
How do you feel today? buggy!	C2	
Mining software repositories with iSPARQL and a software evolution ontology	C2	
Mining software repository to identify crosscutting concerns using combined techniques	C2	
Boa: Analyzing ultra-large-scale code corpus	C2	
How to learn enough data mining to be dangerous in 60 minutes	C2	
SamikshaViz: A panoramic view to measure contribution and performance of software maintenance professionals by mining bug archives	C2	
Codebook: Discovering and exploiting relationships in software repositories	C2	
A systematic process for Mining Software Repositories: Results from a systematic literature review	C2	
22nd International Conference on Neural Information Processing, ICONIP 2015	C3	
Proceedings - 2019 IEEE/ACM 16th International Conference on Mining Software Repositories, MSR 2019	C3	

Artigos	1ª etapa	2ª etapa
Proceedings - 2023 IEEE/ACM 20th International Conference on Mining Software Repositories, MSR 2023	C3	
CBSOFT 2021 - Brazilian Conference on Software; Proceedings - 35th Brazilian Symposium on Software Engineering, SBES 2021	C3	
Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009	C3	
MSR'11: Proceedings of the 8th Working Conference on Mining Software Repositories, Co-located with ICSE 2011	C3	
2013 10th Working Conference on Mining Software Repositories, MSR 2013 - Proceedings	C3	
2012 9th IEEE Working Conference on Mining Software Repositories, MSR 2012 - Proceedings	C3	
30th International Conference on Software Engineering, ICSE 2008 Co-located Workshops - Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR 2008	C3	
MSR 2004: International workshop on mining software repositories	C3	
Proceedings - ICSE 2007 Workshops: Fourth International Workshop on Mining Software Repositories, MSR 2007	C3	
MSR 2005 international workshop on mining software repositories	C3	
30th International Conference on Software Engineering, ICSE 2008 - 2008 International Working Conference on Mining Software Repositories, MSR'08	C3	
Proceedings - 13th Working Conference on Mining Software Repositories, MSR 2016	C3	
ACM International Conference Proceeding Series	C3	

Artigos	1ª etapa	2ª etapa
Preface of the 1st International Workshop on Mining Software Repositories for Software Architecture (MSR4SA'21)	C3	
Synthesis of benchmarks for the C programming language by mining software repositories	C1	
Operationalizing Threats to MSR Studies by Simulation-Based Testing	C1	
Estimating development effort in free/open source software projects by mining software repositories: A case study of OpenStack	C1	
GreenMiner: A hardware based mining software repositories software energy consumption framework	C1	
Abnormal Working Hours: Effect of Rapid Releases and Implications to Work Content	C1	
Pitfalls and guidelines for using time-based Git data	C1	
On the Impact of Refactoring on the Relationship between Quality Attributes and Design Metrics	C1	
Cross-status communication and project outcomes in OSS development: A language style matching perspective	C1	
HBSNIFF: A static analysis tool for Java Hibernate object-relational mapping code smell detection	C1	
On a Factorial Knowledge Architecture for Data Science-powered Software Engineering	C1	
Combining text mining and visualization techniques to study teams' behavioral processes	C1	
Analyzing distributions of emails and commits from OSS contributors through mining software repositories: An exploratory study	C1	
A Declarative Foundation for Comprehensive History Querying	C1	

Artigos	1ª etapa	2ª etapa
Merged Dataset Creation Method Between Thermal Infrared and Microwave Radiometers Onboard Satellites	C1	
Software Development during COVID-19 Pandemic: An Analysis of Stack Overflow and GitHub	C1	
Escaping the time pit: Pitfalls and guidelines for using time-based git data	C1	
On the use of textual feature extraction techniques to support the automated detection of refactoring documentation	C1	
Mining Software Defects: Should We Consider Affected Releases?	C1	
MSR4ML: Reconstructing Artifact Traceability in Machine Learning Repositories	C1	
Sentiment analysis in jira software repositories	C1	
Exploring a framework for identity and attribute linking across heterogeneous data systems	C1	
On the evolution and impact of architectural smells—an industrial case study	C1	
Exploring a framework for identity and attribute linking across heterogeneous data systems	C1	
Refining code ownership with synchronous changes	C1	
Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories Proceedings	C1	
App store mining and analysis: MSR for app stores	C1	
Dealing with noise in defect prediction	C1	
Detecting asynchrony and dephase change patterns by mining software repositories	C1	
Automatically labelled software topic model	C1	

Artigos	1ª etapa	2ª etapa
The effect of IMPORT change in software change history	C1	
Using alloy to support feature-based DSL construction for mining software repositories	C1	
Filtering noise in mixed-purpose fixing commits to improve defect prediction and localization	C1	
An experience report on scaling tools for mining software repositories using mapreduce	C1	
The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information	C1	
DAHLIA: A visual analyzer of database schema evolution	C1	
Fostering synergies: How semantic web technology could influence software repositories	C1	
Domain analysis for mining software repositories: Towards feature-based DSL construction	C1	
Trustrace: Mining software repositories to improve the accuracy of requirement traceability links	C1	
Using the GPGPU for scaling up mining software repositories	C1	
Assessing software quality of agile student projects by data-mining software repositories	C1	
Exploring identical users on GitHub and stack overflow	C1	
A Survey on How Test Flakiness Affects Developers and What Support They Need To Address It	C1	
A decision support platform for guiding a bug triager for resolver recommendation using textual and non-textual features	C1	
Fostering real-time software analysis by leveraging heterogeneous and autonomous software repositories	C1	
Studying the effectiveness of Application Performance Management (APM) tools for detecting performance regressions for web applications: An experience report	C1	

Artigos	1ª etapa	2ª etapa
Quick remedy commits and their impact on mining software repositories	C1	
On the naturalness of hardware descriptions	C1	
CloneMap: A Clone-Aware Code Inspection Tool in Evolving Software	C1	
Organizational Influencers in Open-Source Software Projects	C1	
Exploring regularity in source code: Software science and Zipf's law	C1	
Visualizing software metrics with service-oriented mining software repository for reviewing personal process	C1	
Filling the gaps of development logs and bug issue data	C1	
A historical dataset of software engineering conferences	C1	
Ethics in the mining of software repositories	C1	
Lessons learned from applying social network analysis on an industrial Free/Libre/Open Source Software ecosystem	C1	
Speeding up the data extraction of machine learning approaches: A distributed framework	C1	
How much do code repositories include peripheral modifications?	C1	
Mining software repositories for adaptive change commits using machine learning techniques	C1	
Research on mining software repositories to facilitate refactoring	C1	
Mining unstructured software repositories	C1	
An empirical study on reducing omission errors in practice	C1	
Mining software repositories with CVSgrab	C1	
Nirikshan: Process mining software repositories to identify inefficiencies, imperfections, and enhance existing process capabilities	C1	

Artigos	1ª etapa	2ª etapa
Mining software repositories for comprehensible software fault prediction models	C1	
Mining software repositories for model-driven development	C1	
Mining challenge 2012: The Android platform	C1	
A code inspection tool by mining recurring changes in evolving software	C1	
Mining software repository for cleaning bugs using data mining technique	C1	
CDS: A Cross-Version Software Defect Prediction Model with Data Selection	C1	
PHANTOM: Curating GitHub for engineered software projects using time-series clustering	C1	
Educational software engineering: Where software engineering, education, and gaming meet	C1	
A method to identify and correct problematic software activity data: Exploiting capacity constraints and data redundancies	C1	
Mining Software Repositories for Social Norms	C1	
Ethical Mining: A Case Study on MSR Mining Challenges	C1	
Standing on shoulders or feet? The usage of the MSR data papers	C1	
EMF patterns of usage on github	C1	
Data Quality Matters: A Case Study on Data Label Correctness for Security Bug Report Prediction	C1	
Mining knowledge on technical debt propagation	C1	
Exploring the applicability of low-shot learning in mining software repositories	C1	
Software process simulation based on mining software repositories	C1	

Artigos	1ª etapa	2ª etapa
University-industry collaboration and Open Source Software (OSS) dataset in Mining Software Repositories (MSR) research	C1	
Mining software repositories for defect categorization	C1	
Mining software repositories to acquire software risk knowledge	C1	
A systematic mapping study on mining software repositories	C1	
GrimoireLab: A toolset for software development analytics	C1	
Data-driven search-based software engineering	C1	
A reflection on the impact of model mining from GitHub	C1	
Mining modern repositories with Elasticsearch	C1	
Software Intelligence: The future of mining software engineering data	C1	
A survey on mining software repositories	C1	
Mining software repositories for accurate authorship	C1	
A survey on the use of topic models when mining software repositories	C1	
Creating and analyzing source code repository models: A model-based approach to mining software repositories	C1	
SamikshaUmbra: Contribution and performance assessment of software maintenance professionals by mining software repositories	C1	
Online sharing and integration of results from mining software repositories	OK	C1
QORAL: An external domain-specific language for mining software repositories	OK	C1
Data warehousing in an industrial software development environment	OK	C1

Artigos	1ª etapa	2ª etapa
University-industry collaboration and Open Source Software (OSS) dataset in Mining Software Repositories (MSR) research	OK	C1
Mining software repositories for defect categorization	OK	C1
The MSR cookbook: Mining a decade of research	OK	C1
Process mining software repositories from student projects in an undergraduate software engineering course	OK	C1
Semantic web enabled software analysis	OK	C1
Tools in mining software repositories	OK	C1
Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories	OK	C1
Mining software repositories to support OSS developers: A recommender systems approach	OK	C1
Data mining tools and techniques for mining software repositories: A systematic review	OK	C1
AIMMX: Artificial Intelligence Model Metadata Extractor	OK	C1
A platform for software engineering research	OK	C1
Candoia: A platform for building and sharing mining software repositories tools as apps	OK	C1
Mining source code repositories at massive scale using language modeling	OK	C1
PANDORA: Continuous Mining Software Repository and Dataset Generation	OK	C1
MSRBot: Using bots to answer questions from software repositories	OK	C1
Mining software repositories to acquire software risk knowledge	OK	C1

Artigos	1ª etapa	2ª etapa
The road ahead for mining software repositories	OK	C1
Using Pig as a data preparation language for large-scale mining software repositories studies: An experience report	OK	C1
Boa meets python: A boa dataset of data science software in python language	OK	C1
Boa: Ultra-large-scale software repository and source-code mining	OK	C1
Visual querying and analysis of large software repositories	OK	C1
A Linked Data platform for mining software repositories	OK	C1
Mining software repositories to assist developers and support managers	OK	C1
PyDriller: Python framework for mining software repositories	OK	C1

Tendo definido quais artigos foram removidos segundo os critérios de exclusão, foram selecionados 6 artigos da plataforma *Scopus*. Através destes artigos foi realizado uma busca mediante as técnicas de *snowballing*. Primeiramente realizou-se uma rodada de *Backward Snowballing*, que trouxe os artigos de Cosentino, Luis e Cabot (2016), e Gousios (2013), ambos encontrados nas referências de Lefeuvre et al. (2023). Depois, foi realizado uma rodada da técnica de *Forward Snowballing*, em que foi encontrado o artigo de Mombach e Valente (2018), entre os que citavam o trabalho de Cosentino, Luis e Cabot (2016). Estes artigos podem ser encontrados na Tabela A.2, e seus resumos logo em seguida.

Tabela A.2: Artigos selecionados

Artigos	1ª etapa	2ª etapa
Modelmine: A tool to facilitate mining models from open source repositories	OK	OK
G-Repo: A Tool to Support MSR Studies on GitHub	OK	OK
Fingerprinting and Building Large Reproducible Datasets	OK	OK
Mining and Creating a Software Repositories Dataset	OK	OK
MetricMiner: Supporting researchers in mining software repositories	OK	OK
Sampling Projects in GitHub for MSR Studies	OK	OK
Findings from github: methods, datasets and limitations	OK	OK
The ghtorent dataset and tool suite	OK	OK
GitHub REST API vs GHTorrent vs GitHub Archive: A comparative study	OK	OK

Modelmine: A tool to facilitate mining models from open source repositories
Resumo
O *ModelMine* é uma ferramenta focada em mineração de artefatos e designs baseados em modelos de repositórios de código aberto hospedados no *GitHub* ou no *GitLab*. O trabalho divide a arquitetura da ferramenta em 6 etapas: etapa de indexação processa extensões para gerar índices buscáveis; etapa de paginação limita o número de resultados por página; etapa de redução da consulta faz com que haja um número máximo de resultados por requisição; etapa de consulta utiliza um protocolo para proteger-se de usuários não autorizados e bloquear muitas requisições simultâneas; etapa de representação dos dados trata os resultados obtidos para visualização; e a etapa de ranqueamento dos resultados realiza uma classificação dos resultados. Para validar a abordagem os autores realizam uma comparação entre o *ModelMine* e a ferramenta *PyDriller*. Para comparar os desempenhos, foram utilizadas métricas de tempo de execução, memória máxima e complexidade ciclomática. Para a avaliação de usabilidade foram utilizadas as categorias de interface do usuário, curva de aprendizado, visualização das informações e relatórios de erros. Como limitações, a ferramenta não supera o máximo de 1000 repositórios da *GitHub API*.

G-Repo: A Tool to Support MSR Studies on GitHub
Resumo
O artigo apresenta a ferramenta *G-Repo*, desenvolvida para apoiar os pesquisadores, facilitando a criação de conjuntos de repositórios para estudos na área de mineração de repositórios. O trabalho descreve quais são os primeiros passos para realizar uma pesquisa na área de MSR, citando alguns serviços de hospedagem utilizados como o *GitHub* e sua *API* para obter acesso aos dados dos repositórios. Os autores fazem uma descrição dos módulos característicos da ferramenta, como o módulo de busca, o qual utiliza da *GitHub API* para obter os repositórios, realizando múltiplas quebras na pesquisa, para superar a limitação da *API* de 1000 repositórios por busca. A ferramenta possui a funcionalidade de detecção de idiomas de repositórios. Para validar essa funcionalidade, foram utilizados sete avaliadores humanos em comparação com a ferramenta, para identificar o idioma de 1500 repositórios, no qual houve uma concordância de 98% entre os avaliadores e a ferramenta quanto ao número de repositórios em inglês. Como limitação, a ferramenta não supera um problema do *GitHub* citado pelo trabalho, a possibilidade de obter repositórios que não armazenam um software nas buscas, pois segundo o trabalho, tais repositórios não trazem retorno para uma pesquisa da área de MSR.

Fingerprinting and Building Large Reproducible Datasets
Resumo
O artigo aborda a criação de conjuntos de dados personalizados de projetos e repositório de software no contexto de mineração de repositórios. O foco está na utilização do *Software Heritage*, plataforma que armazena e disponibiliza milhões de projeto de código-aberto e seus históricos. Através do *Software Heritage* os autores visam desenvolver um processo de extração genérico e reprodutível dos dados armazenados, que no geral são elementos de código-fonte e dados que caracterizam propriedades externas como *bug reports*. É proposta a definição de uma "impressão digital" única, composta por critérios e um carimbo de data e hora. Os critérios são utilizados para extrair o conjunto de dados, facilitando a reprodução futura do processo de extração. O carimbo garante que o estado dos dados fonte seja registrado no momento da extração. Há uma limitação de alto uso de recursos para criar e manter atualizado os softwares disponíveis através do *Crawler* do *Software Heritage*, pois ele utiliza de APIs como a do *GitHub*, que possuem limites de requisições e limite máximo de resultados retornados.

Mining and Creating a Software Repositories Dataset
Resumo
O artigo foca na mineração de metadados de repositórios como quantidade de *commits* e de *forks*, tais repositórios estão presentes no *GitHub* e outras plataformas de hospedagem de repositórios *Git*. Os autores utilizam o serviço do *Software Heritage* para armazenar essas informações, com intuito de que possam ser usados em estudos futuros. A abordagem introduz métodos para detectar e extrair *forks* e duplicações dos repositórios de software, no qual verificam se dois repositórios possuem o mesmo diretório de origem e o mesmo conjunto de revisões. Por fim, realizam uma pesquisa para averiguar a correlação entre Padrões de *forks*, saúde e riscos de um software, e indicadores de sucesso. Chegam a conclusão de que é possível a correlação entre tais métricas, mas que é necessário análises mais profundas para obter resultados concretos.

MetricMiner: Supporting researchers in mining software repositories
Resumo
Os autores apresentam o MetricMiner, uma aplicação Web criada para auxiliar pesquisadores em várias etapas da mineração de repositórios de software, como facilitar o armazenamento dos repositórios de software de uma pesquisa, para que o usuário não gaste uma alta quantidade de memória necessária para armazená-los. O software também permite uma fácil análise desses repositórios, oferecendo funcionalidades de cálculo de métricas e extração de dados, também oferece repositórios previamente inseridos por outro usuário, proporcionando economia de tempo e recursos no momento de realizar a seleção e obtenção das informações dos repositórios. Para validar a ferramenta, os autores realizam comparações com outras ferramentas como *Sonar*, *Boa* e *Eclipse Metrics*. No geral, o *MetricMiner* se sai melhor por ter uma maior variedade de funcionalidades, como a possibilidade de armazenar repositórios *Git* e *SVN*, mas sofre de limitações por não ter outras funcionalidades como gráficos para visualizações de dados, e não armazenar repositórios CVS.

Sampling Projects in GitHub for MSR Studies
Resumo
Os autores propuseram a abordagem *GitHub Search* para auxiliar desenvolvedores no momento de realizar a seleção de repositórios para as pesquisas da área de mineração de repositórios de software. Tal abordagem se trata de uma ferramenta que realiza a busca de repositórios do *GitHub* entre 25 características presentes neles, como quantidade de estrelas e data da criação. A ferramenta possui mais de 1.200.000 de repositórios do *GitHub* armazenados em seu banco de dados, e por meio de um minerador de repositório, sempre mantém os repositórios atualizados, verificando se receberam novas modificações, oferecendo assim, milhões de repositórios atualizados para os usuários. Uma possível limitação da ferramenta aparece no uso de um *web crawler* para acessar o HTML da página de um repositório, com intuito de obter certos dados como as informações dos colaboradores, fazendo com que a ferramenta fique limitada ao layout atual dos repositórios do site do *GitHub*.

The GHTorent dataset and tool suite dataset and tool suite
Resumo
Os autores comentam sobre a ferramenta *GHTorrent*, que coleta e armazena dados de repositórios do *GitHub* para pesquisa. O *GHTorrent* supera as limitações de requisições da *GitHub API*, ao ser projetado para usar excessivamente *caching*, armazenando os resultados no banco de dados MongoDB, prevenindo assim, solicitações duplicadas, e também projetado para ser distribuído, permitindo o acesso paralelo de diversos usuários. Algumas limitações que a ferramenta enfrenta e que podem ser prejudiciais de um ponto de vista da MSR são os dados desatualizados, *timestamps* ausentes, *commits* com nomes diferentes para um mesmo usuário, e mudanças nos formatos de dados do *GitHub*. A ferramenta pode facilitar a replicação de estudos existentes devido ao seu conjunto de dados homogeneizados dentre os milhares de projetos armazenados. O artigo conclui que a ferramenta é valiosa para a comunidade de pesquisa e destaca a contínua busca dos desenvolvedores por melhorias na utilidade dos dados. Pontos positivos incluem a detalhada estruturação do *GHTorrent*, o armazenamento de dados e as oportunidades de pesquisa futuras destacadas no artigo.

Findings from github: methods, datasets and limitations
Resumo
O artigo apresenta uma análise de 93 pesquisas sobre mineração de repositórios no *GitHub*, focando em três aspectos principais: métodos empíricos aplicados, conjuntos de dados utilizados e limitações encontradas. A análise revelou preocupações com os processos de coleta de dados, baixa replicabilidade, técnicas de amostragem fracas e a pouca variedade de metodologias. Os resultados das análises mostraram que 75% das pesquisas utilizam método empíricos, observando metadados do *GitHub* diretamente, enquanto o restante usa estudos e entrevistas. Com relação aos conjuntos de dados, as ferramentas mais usadas são a *GitHub API* (39,8%) e *GHTorrent* (34,4%). Limitações encontradas estão relacionadas as inferências baseadas nos métodos empíricos, processo de coleta de dados, generalização dos resultados e uso de serviços terceirizados. A pesquisa sugere a necessidade de avaliar quão novo são os dados coletados e destaca que a maioria das pesquisas utiliza menos de 100 repositórios, dificultando a generalização e replicabilidade dos resultados. Como conclusões, o trabalho identifica problemas como baixa replicabilidade e técnicas de amostragem fracas, e recomenda que pesquisadores compartilhem os repositórios utilizados, empreguem técnicas de amostragem eficientes e diversifiquem os tipos de estudos para aumentar a qualidade dos resultados.

GitHub REST API vs GHTorrent vs GitHub Archive: A comparative study
Resumo
Os autores comparam três ferramentas para obtenção de informações de repositórios de software: *GitHub REST API*, *GitHub Archive* e *GHTorrent*. Após uma breve definição de cada ferramenta, o artigo apresenta consultas específicas que podem ser realizadas através delas, como obtenção dos 1000 projetos com mais estrelas, número de *commits* por contribuidor e linguagens usadas nos 1000 projetos com mais quantidades de estrelas. A avaliação qualitativa realizada, focou na atualização dos dados, no qual a *GitHub REST API* oferece dados sempre atualizados, o *GitHub Archive* atualiza os dados por hora e o *GHTorrent* mensalmente. Os resultados mostraram que a *GitHub API* é mais precisa para obter repositórios com mais estrelas, enquanto para o número de *commits* por contribuidor, os resultados podem ser similares nas três ferramentas. Para consultar linguagens de programação, a *GitHub API* é custosa, o *GitHub Archive* não oferece suporte e o *GHTorrent* exige realizar mais de uma requisição. O estudo conclui que é importante entender as vantagens e limitações de cada ferramenta para escolher a fonte de dados adequada para pesquisas. Pontos positivos incluem a eficiente exemplificação do funcionamento das ferramentas e a demonstração de consultas em cada uma delas.

Bibliografia

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

ALAMER, G.; ALYAHYA, S. Open source software hosting platforms: A collaborative perspective's review. *J. Softw.*, v. 12, n. 4, p. 274–291, 2017.

ALMARZOUQ, M.; ALZAIDAN, A.; ALDALLAL, J. Mining github for research and education: challenges and opportunities. *International Journal of Web Information Systems*, Emerald Publishing Limited, v. 16, n. 4, p. 451–473, 2020.

BARRETO, A. S.; MURTA, L. G. P.; ROCHA, A. R. Uma abordagem para definição de processos baseada em reutilização visando à alta maturidade em processos. In: SBC. *Anais do XI Simpósio Brasileiro de Qualidade de Software*. [S.l.], 2012. p. 429–443.

BERCZUK, S. P.; APPLETON, B. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201741172.

BRITO, G.; VALENTE, M. T. Rest vs graphql: A controlled experiment. In: IEEE. *2020 IEEE international conference on software architecture (ICSA)*. [S.l.], 2020. p. 81–91.

CASALNUOVO, C.; SUCHAK, Y.; RAY, B.; RUBIO-GONZÁLEZ, C. Gitcproc: A tool for processing and classifying github commits. In: *Proceedings of the 26th ACM SIGSOFT international symposium on software testing and analysis*. [S.l.: s.n.], 2017. p. 396–399.

CHACON, S.; STRAUB, B. Pro git. *Recuperado de: http://labs. kernelconcepts. de/-downloads/books/Pro% 20Git*, 2010.

COLLINS-SUSSMAN, B.; FITZPATRICK, B.; PILATO, C. M. Subversion: The definitive guide. *Draft: Revision*, v. 5113, p. 26, 2003.

COSENTINO, V.; LUIS, J.; CABOT, J. Findings from github: methods, datasets and limitations. In: *Proceedings of the 13th International Conference on Mining Software Repositories*. [S.l.: s.n.], 2016. p. 137–141.

DABIC, O.; AGHAJANI, E.; BAVOTA, G. Sampling projects in github for msr studies. *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, p. 560–564, 2021.

D'AMBROS, M.; GALL, H. C.; LANZA, M.; PINZGER, M. Analysing software repositories to understand software evolution. *Software evolution*, Springer, v. 8, n. 30, p. 196, 2008.

DO, T.-B.; NGUYEN, H.-N. H.; MAI, B.-L. L.; NGUYEN, V. Mining and creating a software repositories dataset. In: IEEE. *2020 7th NAFOSTED Conference on Information and Computer Science (NICS)*. [S.l.], 2020. p. 78–83.

GHIOTTO, G.; MURTA, L.; BARROS, M.; HOEK, A. V. D. On the nature of merge conflicts: a study of 2,731 open source java projects hosted by github. *IEEE Transactions on Software Engineering*, IEEE, v. 46, n. 8, p. 892–915, 2018.

GITHUB. *GitHub*. 2022. Disponível em: ⟨https://github.com/⟩.

GOUSIOS, G. The ghtorent dataset and tool suite. In: IEEE. *2013 10th Working Conference on Mining Software Repositories (MSR)*. [S.l.], 2013. p. 233–236.

HASSAN, A. E. The road ahead for mining software repositories. In: *2008 IEEE International Conference on Software Maintenance*. Los Alamitos, CA, USA: IEEE Computer Society, 2008. p. 48–57. Disponível em: ⟨https://doi.ieeecomputersociety.org/10.1109/FOSM.2008.4659248⟩.

JANÁK, J. *Issue tracking systems*. Tese (Doutorado) — Masarykova univerzita, Fakulta informatiky, 2009.

JIANG, J.; LO, D.; HE, J.; XIA, X.; KOCHHAR, P. S.; ZHANG, L. Why and how developers fork what from whom in github. *Empirical Software Engineering*, Springer, v. 22, p. 547–578, 2017.

KAGDI, H.; COLLARD, M. L.; MALETIC, J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice*, Wiley Online Library, v. 19, n. 2, p. 77–131, 2007.

KALLIAMVAKOU, E.; GOUSIOS, G.; BLINCOE, K.; SINGER, L.; GERMAN, D. M.; DAMIAN, D. An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, Springer, v. 21, p. 2035–2071, 2016.

LEFEUVRE, R.; GALASSO, J.; COMBEMALE, B.; SAHRAOUI, H.; ZACCHIROLI, S. Fingerprinting and building large reproducible datasets. In: *Proceedings of the 2023 ACM Conference on Reproducibility and Replicability*. [S.l.: s.n.], 2023. p. 27–36.

LEHTOVIRTA, N. Managing software project with gitlab: How it is done at sparta consulting oy. Jyväskylän ammattikorkeakoulu, 2017.

MOMBACH, T.; VALENTE, M. T. *GitHub REST API vs GHTorrent vs GitHub Archive: A comparative study*. [S.l.]: Tech. Rep, 2018.

MOURÃO, E.; PIMENTEL, J. F.; MURTA, L.; KALINOWSKI, M.; MENDES, E.; WOHLIN, C. On the performance of hybrid search strategies for systematic literature reviews in software engineering. *Information and software technology*, Elsevier, v. 123, p. 106294, 2020.

MOUSTAFA, S.; ELNAINAY, M. Y.; MAKKY, N. E.; ABOUGABAL, M. S. Software bug prediction using weighted majority voting techniques. *Alexandria engineering journal*, Elsevier, v. 57, n. 4, p. 2763–2774, 2018.

MUNAIAH, N.; KROH, S.; CABREY, C.; NAGAPPAN, M. Curating github for engineered software projects. *Empirical Software Engineering*, Springer, v. 22, p. 3219–3253, 2017.

O'SULLIVAN, B. *Mercurial: The Definitive Guide: The Definitive Guide*. [S.l.]: "O'Reilly Media, Inc.", 2009.

OTTE, S. Version control systems. *Computer systems and telematics*, Citeseer, p. 11–13, 2009.

REZA, S. M.; BADREDDIN, O.; RAHAD, K. Modelmine: a tool to facilitate mining models from open source repositories. In: *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. [S.l.: s.n.], 2020. p. 1–5.

ROMANO, S.; CAULO, M.; BUOMPASTORE, M.; GUERRA, L.; MOUNSIF, A.; TELESCA, M.; BALDASSARRE, M. T.; SCANNIELLO, G. G-repo: a tool to support msr studies on github. In: IEEE. *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. [S.l.], 2021. p. 551–555.

SOKOL, F. Z.; ANICHE, M. F.; GEROSA, M. A. Metricminer: Supporting researchers in mining software repositories. In: IEEE. *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. [S.l.], 2013. p. 142–146.

TRAVASSOS, G. H.; GUROV, D.; AMARAL, E. Introdução à engenharia de software experimental. UFRJ Rio de Janeiro, Brazil, 2002.

TUTKO, A.; HENLEY, A. Z.; MOCKUS, A. How are software repositories mined? a systematic literature review of workflows, methodologies, reproducibility, and tools. *arXiv preprint arXiv:2204.08108*, 2022.

WEBER, S.; LUO, J. What makes an open source code popular on git hub? v. 2015, p. 851–855, 01 2015.

ZOLKIFLI, N. N.; NGAH, A.; DERAMAN, A. Version control system: A review. *Procedia Computer Science*, Elsevier, v. 135, p. 408–415, 2018.