RAISE: A Self-Hosted Platform for Mining and Managing Data from GitHub and Jira

Breno Neves
PUC-Rio
Rio de Janeiro, Brazil
brenonevs@aise.inf.puc-rio.br

Daniel Coutinho
PUC-Rio
Rio de Janeiro, Brazil
dcoutinho@inf.puc-rio.br

Eduardo Sardenberg
PUC-Rio
Rio de Janeiro, Brazil
esardenberg@aise.inf.puc-rio.br

Arthur Alesi
PUC-Rio
Rio de Janeiro, Brazil
arthuralesi@aise.inf.puc-rio.br

Marcelo Machado
PUC-Rio
Rio de Janeiro, Brazil
mbmachado@aise.inf.puc-rio.br

Johny Arriel
PUC-Rio
Rio de Janeiro, Brazil
johny@aise.inf.puc-rio.br

Ana Luísa Cavalvante
PUC-Rio
Rio de Janeiro, Brazil
acavalcante@aise.inf.puc-rio.br

Robbie Carvalho
PUC-Rio
Rio de Janeiro, Brazil
robbie@aise.inf.puc-rio.br

Juliana Alves Pereira
PUC-Rio
Rio de Janeiro, Brazil
juliana@inf.puc-rio.br

ABSTRACT

The growing number of software repositories has opened new opportunities for researchers to investigate how software is developed, how teams collaborate, and how quality evolves over time. However, mining useful information out of these repositories often requires custom, ad hoc scripts tailored to specific studies, which are rarely built to be reusable or shareable. This hinders reproducibility, results in redundant data extraction efforts, and wastes computational resources, particularly in academic environments where multiple researchers or teams may need similar datasets. To address these limitations, we developed RAISE, a self-hosted platform for mining and managing data from GitHub and Jira. RAISE offers both an REST API and a web-based interface, allowing streamlined data retrieval, exploration, and export. It is built with widely adopted technologies such as Django, Docker, Celery and React. It is easy to deploy, supports background task execution, and ensures consistent behavior across environments. The platform provides fine-grained filtering, integrates data from both local and remote repositories, and stores results in a structured database for reuse. To evaluate the RAISE's practical value and usability, we performed a user-centered evaluation with six participants, who engaged in a range of realistic repository mining tasks of varying complexity.

KEYWORDS

Mining Software Repositories, Empirical Software Engineering, GitHub, Jira

1 Introduction

The wide availability of software development metadata and artifacts, coupled with significant advances in repository mining techniques, has played a transformative role in empirical software engineering research. By leveraging large-scale datasets, often sourced from open-source projects, researchers have gained valuable insights into real-world software development processes, team dynamics, and project evolution [4, 26]. These data-driven approaches have enabled the community to ground its findings in observable development behavior at scale. However, as the amount of data and its use continue to grow, researchers face significant challenges in ensuring the reproducibility, scalability, and accessibility of mining processes. Ensuring consistent, efficient, and transparent mining workflows is essential to support rigorous and collaborative research in this field [8].

Researchers often design ad hoc mining strategies tailored to specific studies, resulting in single-use, poorly documented scripts reused only within limited contexts, usually by the same researcher [12]. Although sharing final datasets may address some reproducibility concerns, it does not solve the challenge of in-progress data sharing within institutions (*e.g.*, universities or research labs) where multiple researchers run concurrent studies. Divergences in methods, parameters, or preprocessing steps can introduce inconsistencies [8, 22], and researchers frequently find themselves re-mining the same data due to difficulties in sharing intermediate outputs [22]. This redundancy is especially costly: mining operations often consume significant bandwidth and computational power, and when repeated, they lead to substantial waste of time, effort, and infrastructure. A shared mining tool can mitigate these inefficiencies by standardizing data processing practices and promoting reproducibility across research workflows.

We propose RAISE[1], a modular and self-hosted platform for mining and managing data from GitHub and Jira. RAISE is accessible through both an REST API and a web-based interface. It is built with Django[2] and Docker[3], which are popular and well-supported technologies. Rather than easing deployment, this stack replaces fragmented, ad-hoc pipelines with a unified, locally hostable environment that integrates heterogeneous sources and standardizes collection, storage, and querying. This enables cross-system linkage, fosters data reuse, improves reproducibility, and supports modular addition of new sources without modifying existing ones, granting universities and research groups governance and privacy controls.

[1] RAISE stands for Repository Analysis in Software Engineering, while also evoking the notion of elevating the quality and impact of research
[2] https://docs.djangoproject.com/en/5.2/
[3] https://docs.docker.com/

The platform interacts with the official APIs of GitHub[4] and Jira[5], supports asynchronous background tasks using Celery[6], and stores mined data in a structured PostgreSQL[7] database. Unlike existing solutions often limited to a single data source or reliant on complex, ad-hoc integration pipelines, RAISE natively supports the unified management of GitHub and Jira. Moreover, its modular architecture facilitates extension to additional platforms such as GitLab, Bitbucket, or Azure DevOps.

To mine either a GitHub or Jira project, a user simply submits a request specifying the target system (GitHub or Jira) and a set of filters (*e.g.*, date range, issue type, project name). For GitHub, RAISE automatically integrates local and remote data by utilizing tools such as PyDriller [25] and mixing it with remote API data to provide a rich, linked dataset. It collects entities such as commits, issues, pull requests, branches, and repository metadata. For Jira, it extracts structured issue data and supports advanced filtering through JQL[8]. Each mining request is executed asynchronously and tracked via a task queue system, enabling large-scale data collection without blocking user interactions.

We evaluated RAISE through a user-centered experimental study in which researchers performed realistic repository mining tasks, assessing its usability, perceived effectiveness, and practical contribution to mining workflows. Our experiment captures the tool's impact across varying levels of expertise and analytical requirements, with a focus on how well RAISE supports tasks involving navigation, filtering, and interpretation of complex repository data. Results showed that RAISE's consistently enhanced data mining by facilitating navigation, analysis, and cross-referencing of GitHub and Jira data, streamlining the identification of patterns, linking of planning and implementation artifacts, and extraction of insights from complex workflows. Compared to existing tools, RAISE delivered a more robust, developer-focused experience, characterized by: (1) broad data coverage—including commits, issues, pull requests, software metrics, and cross-artifact linking; (2) streamlined automated extraction process—minimizing manual steps and setup time; and (3) analytical readiness of output—producing results that required minimal post-processing.

The contributions of this paper are threefold: (1) We present RAISE, a self-hosted and modular platform designed to mine and integrate data from GitHub and Jira, enabling the management of heterogeneous artifacts through an extensible architecture that supports institutional deployment and customization; (2) We demonstrate how RAISE supports empirical software engineering studies by simplifying data mining tasks and facilitating structured collaboration across teams, avoiding the fragmentation and proliferation of ad hoc scripts that commonly characterize repository mining practices; and (3) We provide evidence, through controlled experiments with researchers, that RAISE enables interpretation and cross-referencing of data in a streamlined and accessible manner, showing that the tool simplifies complex workflows.

[4]https://docs.github.com/en/rest?apiVersion=2022-11-28
[5]https://developer.atlassian.com/cloud/jira/platform/rest/v3/intro/
[6]https://docs.celeryq.dev/en/stable/
[7]https://www.postgresql.org/docs/
[8]JQL (Jira Query Language) is a flexible query language used in Jira to filter and search for issues based on custom criteria like status, assignee, or keywords.

2 Related Work

Software repositories such as GitHub[9] and project management tools such as Jira[10] have become central pillars of modern software engineering research, offering rich data on development practices, team dynamics, and project evolution. GitHub, in particular, plays a pivotal role in Mining Software Repositories (MSR) due to its openness and scale. These platforms provide access to commits, pull requests, issues, and contributor activity—enabling analyses that leverage data mining and natural language processing (NLP) to generate actionable insights [11, 23]. However, existing mining approaches still struggle with integration, scalability, and analytical depth [5, 10].

To assess how state-of-the-art tools address these challenges compared to RAISE, Table 1 presents a concise summary of the key capabilities of the leading tools in the MSR ecosystem. These tools are also discussed individually throughout this section. It is important to note that a direct comparison with our approach is not feasible. Differences in performance can stem from infrastructure-related factors (*e.g.*, network conditions, hardware) and, the existing tools pursue different design goals (*e.g.*, archival dataset providers) and therefore expose different feature sets and trade-offs.

GitHub Data Mining. GitHub is a central platform for collaborative software development and a valuable source for empirical research on software engineering [1, 10, 13]. Yet, extracting actionable insights from its vast data remains challenging [10].

GHTorrent [9] offers large, regularly updated MySQL dumps of GitHub metadata. Despite its scope, it lacks real-time access, omits rich artifacts like code reviews and comments, and is no longer maintained—limiting its use in studies needing up-to-date or fine-grained data. Tools such as *PyDriller* [25], *MetricMiner* [24], and *GitPython* [6] work only with local repositories and do not integrate with the GitHub API, restricting access to key metadata and user interactions critical for socio-technical analysis. *PyDriller* is lightweight and user-friendly, supporting commit iteration and basic software metrics, but limited to the version history. *MetricMiner* extends capabilities via database-backed queries and source code analysis, but has a complex setup and narrow focus. *GitPython* provides low-level access to Git internals, enabling fine-grained inspections, but lacks high-level abstractions for broader mining tasks.

In contrast, RAISE bridges these gaps by integrating local and remote data sources. It combines deep Git history analysis with platform-level GitHub metadata, supporting the extraction and cross-referencing of commits, pull requests, issues, branches, and software metrics. Its modular design streamlines setup and execution, minimizing manual steps and delivering analysis-ready data without extensive preprocessing.

Jira Data Mining. Jira is a widely adopted platform for issue tracking and project planning within agile workflows [2, 20]. Although it offers structured and detailed data, integrating Jira with external analysis tools remains challenging due to limited interoperability [18]. While several studies have proposed approaches for mining Jira data, most solutions involve extensive manual configuration and lack generalizable, automated workflows.

[9]https://github.com/
[10]https://www.atlassian.com/software/jira

Table 1: Comparison between RAISE and related tools for mining data from GitHub and Jira.

Features	GHTorrent[9]	PyDriller[25]	MetricMiner[24]	GrimoireLab/Jira[7]	GitPython[6]	DRMiner[27]	Ramos & Rossi[21]	RAISE (Proposed)
Open-source platform	X	X	X	X	X		X	X
GitHub integration	X	X	X		X			X
Jira integration				X		X	X	X
Web application				X				X
Interface to query data		X	X	X				X
Code metrics mining		X	X		X			X
Workflow/issue tracking analysis				X		X	X	X
Real-time updates				X				X
Collaborative platform	X		X	X				X
Cross-platform analysis								X

GrimoireLab/Jira [7] supports Jira mining and visualization through a web interface, but lacks GitHub integration, limiting cross-platform analysis. *Montgomery et al.* [17] and *Krismayer et al.* [14] leveraged large-scale datasets or structured workflows, yet relied on ad hoc scripts and were limited in handling unstructured or cross-platform data. Process mining approaches [5, 21] have also been explored. While effective in uncovering workflows or integrating event logs into SCRUM pipelines, these approaches required project-specific setups and lack generalizability. Moreover, *Marques et al.* [16] focused on agile practice analysis through Jira logs, but faced scalability issues due to preprocessing requirements. Lastly, *DRMiner* [27] leverages large language models to extract design rationales from Jira issues, but its reliance on annotated data and fine-tuned prompts further highlights the need for automated, broadly applicable tools, like ours, that enable mining and analysis across platforms without extensive overhead.

Comparative Analysis: RAISE - *MetricMiner* - *GrimoireLab*. We conducted a focused comparison of RAISE with two prominent tools from Table 1: *MetricMiner* [24] (for GitHub) and *GrimoireLab* (for Jira), chosen for their broadest feature coverage among the mentioned solutions. We investigate three core dimensions. First, *data coverage*, which refers to the variety and granularity of artifacts that the tool can extract, such as commits, issues, pull requests, branches, code metrics, and comments. Moreover, we analyzed their ability to automatically establish relationships between them (*e.g.*, linking commits to issues or associating PRs with their respective branches). Second, *usability*, which concerns how straightforward and streamlined the user experience is when configuring and completing repository mining tasks. This includes the number of manual steps involved, the clarity of the workflow, and the effort required to reach task completion through the interface. Third, *analytical readiness*, which considers how close the extracted data is to being directly usable. This includes whether the output requires additional transformation (*e.g.*, cleaning or normalization), whether it follows a well-defined schema suitable for querying, and whether the tool provides integrated exploration mechanisms such as dashboards, filters, or visual summaries. These aspects can be assessed through usability scales or by the presence of built-in capabilities for interpretation and insight.

Data Coverage. RAISE has an intrinsic advantage compared to both tools in terms of data coverage. It natively integrates both GitHub and Jira data within a unified infrastructure, which, if needed, allows for easier integration and comparison between different data sources. *MetricMiner* provides detailed commit-level metrics and supports some historical analysis, it is restricted to local Git repositories and lacks access to high-level collaboration artifacts such as pull requests, reviews, and issues. Conversely, *GrimoireLab* supports Jira issue extraction, but does not establish linkages between issues and corresponding implementation elements (*e.g.*, code commits). RAISE addresses these limitations by extracting a wider range of artifacts—commits, issues, pull requests, branches, comments, code metrics—and establishing automatic connections between them, such as linking pull requests to branches or commits to tracked issues in Jira. This creates a more holistic, relational representation of software development workflows.

Usability. Existing repository mining tools such as *MetricMiner* and *GrimoireLab* impose a steep usability barrier due to their reliance on manual configuration and technical setup. *MetricMiner* requires local repository cloning, SQL-based customization, and integration with external databases, while *GrimoireLab* depends on orchestrating multiple components via Elasticsearch and writing non-trivial YAML files for each data source and visualization layer. These procedures not only increase the cognitive load for users unfamiliar with DevOps or scripting environments, but also extend the number of steps needed to initiate even basic mining tasks, especially when integrating heterogeneous sources such as GitHub and Jira. In contrast, RAISE offers a streamlined user experience through a web-based graphical interface that centralizes task execution and eliminates the need for low-level technical configuration. All actions—such as setting up data sources, executing mining jobs, tracking their progress, and previewing results—are accessible via a cohesive interface with intuitive navigation, feedback indicators, and visual affordances. We show in Section 5 that even users with limited technical background or tool proficiency successfully completed the assigned tasks and positively rated the RAISE's usability.

Analytical Readiness. One of the most pronounced differences lies in the format and accessibility of the mined data. *MetricMiner* outputs raw SQL tables, which require post-processing and manual joins. *GrimoireLab* provides data through dashboards but often lacks structural clarity and demands additional cleaning for reuse. In contrast, RAISE presents the data in a normalized and consistent schema, ready for immediate querying or export. Moreover, it integrates features such as preview interfaces, filterable summaries, and graphical dashboards that help users interpret results in situ, significantly reducing the need for auxiliary tooling. The study participants emphasized how the tool's presentation of data in cards, charts, and structured JSON formats allowed them to reason about patterns and trends without requiring downstream transformation.

3 Overview of RAISE

The existing mining tools highlight a key limitation: the lack of solutions capable of integrating data across platforms to provide a cohesive view of software development processes. As modern repositories such as GitHub and Jira produce large and heterogeneous datasets, there is a need for a platform that supports longitudinal analyses of both sources [15].

RAISE addresses these challenges by providing a unified database that consolidates data from GitHub, Jira, and other sources within a modular, open-source, and containerized infrastructure. This centralization eliminates fragmented integrations that often result in inconsistencies, rework and low reproducibility of experiments. The platform standardizes data collection, storage, and querying, ensuring reproducibility and auditability of studies, while also supporting the reuse of data previously mined by different projects without repeating extractions. Moreover, by combining structured and temporal information, RAISE enables scalable and asynchronous data collection while supporting artifact lifecycle tracing across multiple dimensions, from issues to commits and pull requests.

This modular design ensures extensibility, allowing new sources to be added without disrupting existing ones, and local hosting (*e.g.*, within universities) ensures governance, access control, and compliance with privacy requirements. Acting as a common base for research groups, RAISE reduces the dependency on scattered solutions, speeds up collaboration, and reuse. Through a web-based interface, users can initiate mining activities, monitor execution, and navigate dashboards that summarize collected information via graphical and numerical indicators, while refining queries and inspecting underlying structured data. Its web-based interface offers a user-friendly way to configure, execute, and visualize mining tasks, lowering the technical barrier and streamlining common operations.

This combination of automation, visibility, and interactivity minimizes the need for manual configuration and enables an integrated, scalable engagement with large-scale software repositories, supporting longitudinal and cross-platform analyses of development artifacts and practices.

3.1 GitHub Data Mining

The GitHub mining component of RAISE is designed to extract rich, multi-level data from software repositories by combining local repository analysis with remote API queries. The system integrates several technologies to optimize performance, coverage, and resilience to rate limits.

The mining process begins when the user submits a request to the API, either to initiate data collection (mining) or to query previously stored information. This request—issued via a **REST interface** using either `POST` or `GET` depending on whether the action is data collection or querying—is first handled by the **API Dispatcher**, implemented using Django. The dispatcher is responsible for parsing the request, validating parameters, and routing the task appropriately. Once validated, the request is handed over to the **Redis Broker**, which acts as a message queue to decouple request handling from task execution. From there, the job is picked up by Celery workers and processed asynchronously, ensuring that long-running operations do not block the main web application. A

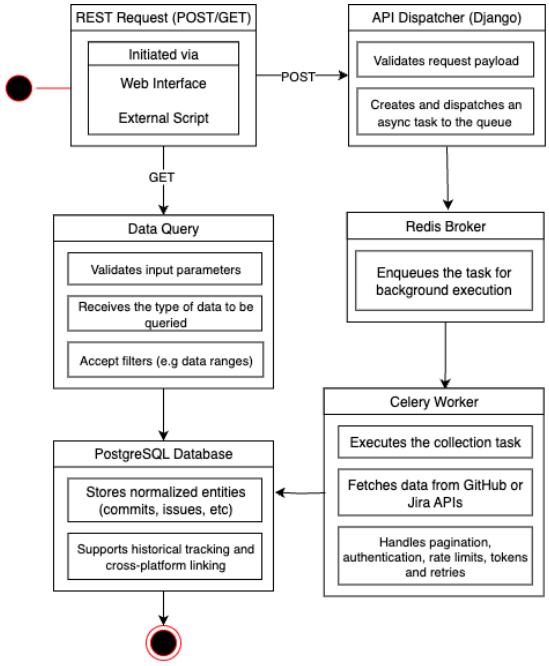

Figure 1: Flowchart illustrating the steps in the operation of the proposed solution.

task ID is returned to the user for monitoring, and the system supports tracking, cancellation, and resumption of tasks when needed. RAISE combines two complementary strategies to collect data:
(i) Local analysis with PyDriller: The repository is cloned locally (if not already available), and PyDriller is used to analyze its Git history. This approach significantly reduces dependency on the GitHub API. For each commit, PyDriller extracts metadata (SHA, message, author, date), the list of modified files, and modifications at the method level. It also supports the computation of code metrics such as cyclomatic complexity and churn, which are useful for identifying hotspots and estimating technical debt.
(ii) Remote data via the GitHub REST API: Certain types of data, such as issues, pull requests, branch metadata, and repository statistics, are only accessible through the official API. RAISE queries these endpoints using paginated requests and handles large datasets incrementally. A rate-limit management system monitors API usage and rotates authentication tokens to avoid request throttling (more information is available on the replication package [19]).

RAISE includes optimizations like dynamic splitting of large time intervals during collection and reuse of previously retrieved complex data (e.g., PR timelines, issue comments), minimizing redundant computations.

It supports collecting GitHub artifacts for analyzing development activity. **Commits** are parsed to extract metadata (authors, timestamps, messages, file changes), with optional method-level analysis. **Branches** are retrieved with associated metadata (name, head SHA, protection), including links to pull requests, enabling the identification of branching strategies.

Issues are mined with rich detail, including identifiers, titles, descriptions, status, priorities, assignees, timestamps, labels, and

full comment threads. Timeline events, such as status changes and user assignments, are also collected to support lifecycle analysis. Similarly, **Pull Requests (PRs)** are enriched with metadata and related discussions, capturing titles, descriptions, states (open, closed, merged), review comments and status, associated branches, linked issues, and events like approvals and merges. This enables a comprehensive view of review workflows and integration cycles.

In addition to these core artifacts, **Repository Metadata**, including stars, forks, license type, contributors, releases, and active branches, is collected to contextualize project characteristics. When necessary, limited web scraping is used only to retrieve metadata missing from the API, *e.g.*, information on which repositories depend on it (used_by), number of contributors, and number of watchers. Finally, **Commit-Issue Linking** allows for automatic cross-referencing of commits and Jira issues when messages include issue keys (e.g., `PROJ-123`), enabling integrated traceability between code changes and planning artifacts across platforms.

The combination of local and remote analysis allows `RAISE` to extract a comprehensive dataset for each repository. The resulting data can support empirical studies of code quality, team collaboration, maintenance practices, and productivity trends across different projects and timeframes.

3.2 Jira Data Mining

The Jira mining module in `RAISE` offers a scalable and customizable solution for extracting structured data from software projects. Using Atlassian's REST API, it retrieves issue-related data, supporting complex filtering, traceability, and workflow analysis.

`RAISE` collects rich **Issue** metadata, including IDs, project keys, types, summaries, descriptions, statuses, story points, linked commits, timestamps, priorities, creators, assignees, resolutions, labels, components, and project-specific custom fields. **Comments** are retrieved with authorship and timestamps, maintaining chronological linkage to their issues.

To contextualize planning, **Project Metadata**—such as team structures, boards, sprints, epics, and versions—is also collected. **Temporal Context** features (e.g., working days, sprint durations, release cycles) enable time-based analysis. **Commit-Issue Linking** is performed by scanning commit messages, branch names, and PR descriptions for Jira keys, establishing traceability between code and planning.

Descriptions in Atlassian Document Format (ADF)[11] are recursively parsed to extract readable content. Parsed fields are stored in dedicated columns, while raw data remains available in an `all_fields` JSON for flexible analysis. This design supports reusable pipelines for large-scale analyses like performance tracking, bug recurrence, and backlog prioritization.

3.3 Data Collection, Storage, and Querying

`RAISE` is designed to collect, store, and expose software development data from GitHub and Jira through a relational data model and a comprehensive set of RESTful API endpoints (see Figure 1 for a data flowchart). These interfaces enable fine-grained queries over the collected data, supporting large-scale empirical analyses and facilitating integration with external tools and workflows.

[11] https://developer.atlassian.com/cloud/jira/platform/apis/document/structure/

Data Collection. Collection is initiated via dedicated API endpoints that support asynchronous operations and configurable filters. All collection tasks are managed in the background to ensure non-blocking performance, and each task returns a unique identifier to allow for asynchronous tracking and management.

Schema and Storage. The collected data is structured into a normalized PostgreSQL schema designed for cross-platform analysis [19]. Commits are stored with identifiers, timestamps, messages, and complexity metrics—including insertions, deletions, and DMM scores[12]. Commits reference modified files, with diff information and modification types, and may include extracted methods annotated with complexity and nesting data. Authors and committers are linked via foreign keys.

Repository-level metadata—such as star count, fork count, topics, and visibility—is maintained in a separate table, while branches and pull requests are stored in dedicated structures with state, merge data, associated commits, and discussion threads. GitHub issues are captured with their full lifecycle attributes, including states, assignees, labels, milestones, and timestamps, with comments and reactions stored as JSON fields.

Jira data is similarly organized in a unified issue table containing keys, summaries, statuses, priorities, project information, and custom fields. Cross-platform integration is achieved by linking issues with related GitHub commits, enabling unified traceability.

The complete schema—including table relationships, field specifications, and entity mappings—is provided in our supplementary material [19], offering transparency and reproducibility for downstream analysis.

Querying Interface. Stored data is accessible via endpoints with support for filtering and pagination. Users can query commits, issues, PRs, branches, and metadata using parameters like text, dates, status, labels, and project fields. Cross-platform and cross-artifact queries are supported, including Jira-specific fields. Endpoints follow standard conventions (e.g., ISO 8601 dates, HTTP status codes, structured parameters).

3.4 The `RAISE` Web Interface

While the API is a powerful interface for more complex data collection and extraction operations, it requires scripting skills, so we identified the need for an interface with a lower barrier of entry, be it for simpler operations, such as collecting data from a single source, or just checking the overall status of the database in an easier and faster way (how much data is collected, if there are any collection tasks running, etc.). Thus, we introduce a web-based front-end application integrated with the API. The interface was developed using modern web technologies, including the React-based Next.js framework, TypeScript for static typing, and Material UI (MUI) for consistent and responsive visual components.

As shown in Figure 2, it allows users to choose between GitHub and Jira as the data source. In the example shown, the GitHub view is selected. The left-hand navigation menu is organized into four main sections: Overview, Collect, Preview, and Jobs.

[12] DMM (Design Metrics for Maintainability) scores are heuristic indicators used to estimate code maintainability based on factors such as method size, nesting depth, and structural complexity.

Figure 2: Screenshot showcasing the dashboard page from our web interface.

The Overview section displays a high-level dashboard summarizing key repository statistics, such as the number of repositories mined, issues, pull requests, commits, and users. Below these statistics, the dashboard includes a general chart that visually presents the evolution of these elements over time, enabling users to track project activity and growth trends. A similar layout is presented for Jira-based repositories, offering insights such as sprint information, issue status transitions, and user assignments. The right-side panel provides date filters, allowing users to specify custom start and end dates for the analysis window.

Through the Collect menu, users can initiate the mining of new repositories from either GitHub or Jira. The user has the option to select the time range for which they want the data to be collected and to choose specific types of data they wish to retrieve, such as issues, comments, pull requests, and commits from the repository.The user can also define a collection interval to collect data within a specific time period. The Jobs menu provides an overview of all data collection tasks, including their current status (Finished, Failure, Cancelled, In Queue, and others), also displaying a more detailed message describing the status of the current process. This allows users to monitor task execution, detect failures, and manage multiple concurrent mining processes and interrupt tasks if necessary. The Preview menu offers a raw data exploration interface in the form of a table where users can browse through the collected data using dynamic filters for searching or sorting, It is possible to view the data in a general overview or select a specific repository for GitHub or a specific project for Jira, as well as exporting it in CSV or JSON formats. In the top-right corner of the interface, users can access the RAISE documentation, which offers detailed guidance on how to use the tool, configure mining tasks, and interpret results.

It supports a wide range of users, from researchers conducting empirical studies to project managers monitoring development activity. To better understand its usability, effectiveness, and areas for improvement, was conducted a qualitative analysis of the tool, involving feedback from six researchers. More details about this experiment with the participants are described in the Table 4.

4 Study Design

This study aims to validate the usability and perceived usefulness of the RAISE tool by conducting an experiment designed to simulate realistic usage scenarios and evaluate how potential users would interact with the platform in practice, considering not only its technical capabilities but also its accessibility and overall user experience. Given that the core motivation of RAISE is to address issues such as fragmented integrations, inconsistencies, rework, and low reproducibility by unifying multiple data sources in a single environment, the evaluation focused on how effectively the tool supports these goals in real usage rather than on direct comparisons with other tools or performance benchmarking, which are not the primary objectives of this work. To achieve this, we adopted a qualitative methodology grounded in structured experiments involving hands-on activities with the tool, allowing us to observe how individuals with different backgrounds and levels of experience navigated its features, interpreted its outputs, and completed mining-related tasks. The study was organized into three main steps: (1) Preparation, (2) Execution, and (3) Data Analysis (see Figure 3). These steps include the design and execution of targeted tasks as well as the application of a set of survey questions, enabling us to capture nuanced insights into the participants' interaction patterns and the tool's effectiveness in supporting typical data mining workflows. We will detail each step in the following sections.

4.1 Preparation

In the preparation phase, we formulated the structure of the experiment based on a prior study [3], which guided the definition of

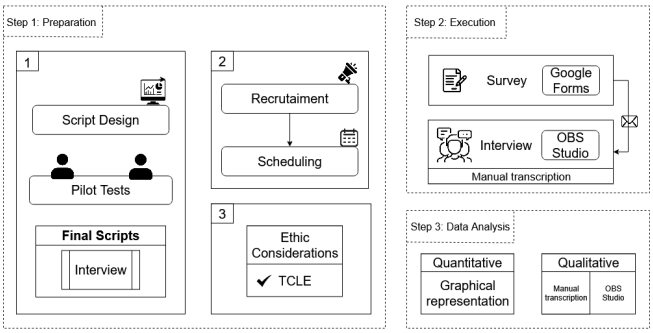

Figure 3: Overview of the three-phase study procedure.

the three categories of tasks used in our protocol: filtering, basic, and assimilation. Filtering tasks were designed to evaluate whether participants had the minimum level of familiarity with the domain and the interface necessary to proceed with the study; these tasks involved basic identification of elements in the interface, such as locating specific fields or interpreting simple data points. Basic tasks required participants to perform isolated operations—such as navigating between screens or retrieving particular information—without external guidance or simultaneous activity, allowing us to assess autonomous interaction with specific features of the platform under conditions that simulate straightforward mining operations. Assimilation tasks, in turn, involved more complex challenges in which participants had to correlate multiple types of information across different screens—such as linking issues to commits or interpreting project-level trends—in order to complete objectives that simulated real-world analytical scenarios. The full list of experimental tasks can be found in Table 2.

Six participants (software engineering researchers with varying education levels) were selected to represent diverse professional and disciplinary backgrounds, allowing us to examine how individuals with different areas of expertise and practical experience engaged with the system. Sessions were scheduled individually. Two additional participants were part of the study but were excluded due to failing most (>50%) of the filtering tasks, implying that they did not have an understanding of the concepts required for participating in the study.

We prepared an informed consent form to ensure that participants were fully aware of the nature and objectives of the study. The document described the purpose of the experiment, the types of data to be collected—specifically audio and video recordings—and how this information would be used to evaluate the proposed tool. Participation was conditioned on reading and signing the form, confirming the participant's understanding and agreement with the research procedures and data usage terms.

4.2 Execution

The execution phase of the study was organized into three distinct stages, which were developed after the execution of two pilot studies: (1) participant onboarding and consent, (2) experimental interaction and interview, and (3) post-experiment survey completion.

1. Introduction and Consent. Each session began with a formal introduction to the study. Participants were welcomed and presented with a printed informed consent form, which they read and signed in person. This document detailed the objectives of the research, ensured anonymity, and clarified that screen and audio

Table 2: Structure of the experimental task script used in the usability study.

Filtering Tasks (FT)	
FT1	How many GitHub pull requests are currently available in the system?
FT2	Which GitHub repositories have data available in the system?
FT3	How many forks does the `facebook/react` repository have?
FT4	How many Jira issues have been collected in the system?
FT5	Which Jira projects have data available in the system?
FT6	How many mining tasks are currently being executed in the system?
Basic Tasks (BT)	
BT1	For the `facebook/react` project, only a limited period of data has been collected. What is that period?
BT2	How many commits were made to the `google/guice` repository during April 2025?
BT3	How many issues were created in the `CSTONE` Jira project during October 2024?
BT4	You are a data scientist and need to export pull request data opened in the `spring-projects/spring-boot` repository in May 2025. How would you perform this task?
Assimilation Tasks (AT)	
AT1	For the `CSTONE` Jira project, who are the users involved? Select one user and identify an issue for which they are responsible.
AT2	You are a researcher studying productivity in the `pandas-dev/pandas` repository. In the first week of June 2025, is there any developer who made a large number of commits? (Participants must gather this data themselves.)
AT3	You are a manager at the Spring organization. Your supervisor requested a report comparing perceived productivity across the `spring-projects/spring-boot` and `spring-projects/spring-framework` repositories in May 2025. What are your findings?
AT4	You supervise a university research group and want to use Jira (`CSTONE`) data to analyze productivity differences between the academic term (Sep–Nov 2024) and the holiday break (Dec 2024–Feb 2025). What differences can be observed?

Table 3: Structure of the survey questionnaire

Evaluation of the `RAISE` Tool		
ID	**Section**	**Question**
Q1	TAM	The information on the tool's Overview screen is clear and understandable.
Q2	TAM	The tool's Overview screen allowed me to easily navigate and explore data.
Q3	TAM	Interacting with the tool's Overview screen required little mental effort.
Q4	TAM	The process of mining new data on the tool's Collect screen is clear.
Q5	TAM	The tool's Preview screen allowed me to easily navigate and export data.
Q6	TAM	Managing mining tasks on the Jobs screen was intuitive.
Q7	TAM	The tool facilitates data mining tasks.
Q8	TAM	I was able to collect, view, and export the mined data without significant difficulty.
Q9	TAM	The tool enhances productivity in repository mining activities.
Q10	TAM	I would use the tool for data mining.
Participant Characterization		
Q11	Charact.	What is your area of expertise?
Q12	Charact.	How many years of professional experience do you have in your field?
Q13	Charact.	How familiar are you with GitHub?
Q14	Charact.	How familiar are you with Jira?
Thank you!		
Q15	Final	What are your thoughts on this study and the tool?

recordings would be used exclusively for research purposes within the scope of the study.

2. Preparation and Task Execution. Following consent, participants were shown a short training video providing an overview of the platform's interface and core functionalities. The video served to ensure a basic level of familiarity with the tool, allowing participants to focus on completing the assigned tasks rather than learning the system during the experiment. After the training, participants began the experimental session, which involved answering a series of practical questions by interacting with the RAISE web interface. Tasks were presented one at a time by the researcher, and participants were instructed to use the *think-aloud* protocol, verbalizing their thought process and decisions during task execution. All sessions were conducted in person and recorded using OBS Studio, capturing both screen activity and audio for subsequent transcription and qualitative analysis.

3. Post-Experiment Survey. After completing all tasks and providing verbal feedback, participants filled out a structured survey designed to collect additional insights. This instrument combined open-ended questions with scaled items based on the Technology Acceptance Model (TAM) to evaluate perceived ease of use, usefulness, and overall impressions of the tool. It also included demographic and background questions to characterize participant profiles. The complete structure of the questionnaire is presented in Table 3.

4.3 Data Analysis

The final step of the study consisted of both quantitative and qualitative analyses, each applied to different types of data collected during the experiment.

Quantitative analysis. After completing the tasks, all participants filled out a structured questionnaire composed of ten TAM-based questions (Q1–Q10), each using a 5-point Likert scale. We aggregated the responses and calculated the average score for each question to identify general trends in participants' perceptions of the platform. These averages were then visualized in a horizontal bar chart (Figure 4), which maps the average score of each item to its corresponding question identifier. This visualization helps summarize the distribution of user feedback across the evaluated dimensions of the system.

Qualitative analysis. In parallel, we conducted a qualitative analysis based on video recordings that were captured during the experiment sessions by using the OBS Studio software. Participants were instructed to verbalize their reasoning and impressions while performing the tasks, following the think-aloud protocol. These sessions were reviewed to extract observations regarding participants' behaviors, hesitation patterns, verbal feedback, and interaction strategies. Additionally, the final open-ended question (Q15) in the questionnaire was used to capture participants' subjective impressions and suggestions for improvement. These insights were used to complement and contextualize the quantitative findings.

4.4 Research Questions

This study was designed to explore how end users—particularly those involved in software engineering and data analytics—interact with and evaluate the RAISE platform in realistic usage scenarios. To guide the structure of the experiments and the subsequent data analysis, we formulated the following two research questions (RQs), each targeting a key dimension of the tool's capabilities, user experience, and comparative advantages over existing solutions:

Table 4: Responses from participants (P1–P6) in the usability study of the Raise tool.

QID	P1	P2	P3	P4	P5	P6
Q1	4	5	3	5	5	2
Q2	4	4	4	5	4	4
Q3	5	4	5	5	1	5
Q4	3	4	3	4	3	3
Q5	5	5	5	4	5	2
Q6	4	4	3	5	2	5
Q7	5	5	5	5	4	4
Q8	5	5	4	5	5	5
Q9	5	5	5	5	5	5
Q10	5	5	5	5	5	5
Q11	SE	SE	SE	SE	—	DS*
Q12	<1	1–3	1–3	1–3	<1	<1
Q13	5	4	4	3	2	4
Q14	5	3	2	2	1	2
Q15	5	5	5	5	4	5

* Data Science

RQ_1: How usable is RAISE when performing tasks of varying complexity during realistic repository exploration scenarios? This question investigates the tool's usability as experienced by participants while executing experimental tasks designed to simulate real-world analysis workflows. Based on the experimental questions listed in Table 2, users engaged with the platform through filtering, basic, and assimilation tasks that varied in cognitive demand and interface interaction. To evaluate usability, we analyzed participants' performance metrics—such as task completion, accuracy, and time spent per section—summarized in Table 5.

RQ_2: How do users perceive the usefulness and applicability of RAISE in practical software engineering contexts? This question aims to capture participants' subjective evaluation of the tool's relevance, clarity, and alignment with their analytical needs. To address it, we consider responses from a structured post-session questionnaire (Table 3), along with aggregated user feedback (Table 5) that reflects perceived utility and ease of use. These perceptions are further analyzed through a visual summary of evaluation scores presented in Figure 4, allowing us to assess how participants position the tool in terms of effectiveness and real-world applicability.

These research questions reflect the dual goals of the study: to evaluate the usability of RAISE in action and to understand how it is perceived in terms of utility and applicability by its target audience.

5 Results and Discussion

This section presents the findings of our evaluation study, structured around the two aforementioned RQs. To analyze these results, we combined quantitative analysis of the Likert-scale responses

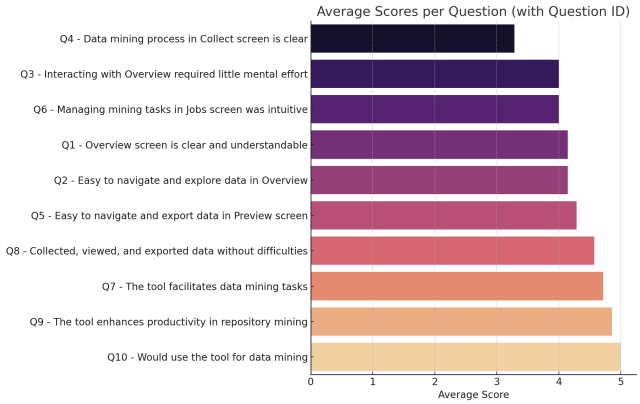

Figure 4: Average scores per question obtained from participant responses in the usability study.

with qualitative insights derived from open-ended feedback and observation of participant behavior during the experiment.

5.1 Usability of RAISE (RQ_1)

To address RQ_2, we analyzed participants' performance while completing the experimental tasks structured in Table 2. These tasks were grouped into three categories—filtering (FT), basic (BT), and assimilation (AT)—each with increasing levels of cognitive and operational complexity. Participant performance data, including task completion and time spent per session, is summarized in Table 5.

Filtering tasks (FT1–FT6) had the highest success rates overall. Most participants answered all filtering tasks correctly, and the time spent on this session was minimal—ranging from 2 to 6 minutes for most users, with P6 slightly above that. The task prompts in this group involved identifying straightforward repository metrics (e.g., number of pull requests, issues, or tasks), and the high accuracy suggests that RAISE's interface supports quick access to summary-level data. The consistency of performance across users indicates low cognitive demand and good clarity in the design of overview and data listing screens.

Basic tasks (BT1–BT4) also showed strong performance, with nearly all participants completing these tasks correctly. The only error observed was in BT2, which was incorrectly answered by participant P2. This isolated case suggests a potential misunderstanding or slip, rather than a systemic usability issue. Overall, these tasks were completed with high accuracy, indicating that users were able to interact effectively with the system's features involving date filtering, repository-specific queries, and data export functionalities. In terms of time investment, participants spent between 4 and 9 minutes on this session. P1, P2, and P4 completed the tasks relatively quickly (under 5 minutes), suggesting efficient interaction flows. In contrast, P3 and P6 took 9 minutes, which may reflect a more exploratory or cautious approach to navigation, rather than a barrier imposed by the system itself. Importantly, the consistency of task success across users, despite this variation in time, indicates that the interface was accessible and functional for users with differing exploration strategies.

Assimilation tasks (AT1–AT4) presented the greatest challenge, with AT1 and AT2 showing the highest error rates—two and three incorrect responses, respectively. These failures suggest potential usability and comprehension issues specific to those tasks.

In AT1, the errors may reflect difficulties in locating or cross-referencing information distributed across different parts of the interface. Participants might have struggled to identify consistent cues indicating user responsibility, or faced ambiguity in how contributors were represented. The absence of a more user-friendly and visually organized interface may have increased the cognitive load, making it harder for users to form complete or accurate associations.

AT2 required users to isolate activity within a specific time window. Errors in this task likely stemmed from challenges in configuring the temporal filter or interpreting commit data in a meaningful way. The interface may not have made it obvious how to extract relevant activity summaries or compare developer behavior efficiently. In both cases, the observed delays and errors point to possible issues in feedback clarity, data visibility, or alignment between user expectations and system affordances.

Together, these patterns suggest that while the tool offers access to the necessary information and may present challenges in supporting complex exploratory paths—particularly when users must interpret distributed data without guided support—it nonetheless demonstrates that successful completion of mining tasks is feasible. This indicates that the tool provides a solid foundation for exploratory analysis, even in the absence of extensive guidance or the availability of complex analysis/comparison tools.

5.2 RAISE's Perceived Usefulness and Applicability (RQ_2)

To answer RQ_3, we analyzed participants' subjective perceptions of the tool's usefulness and applicability based on structured survey responses (Table 3), individual ratings (Table 4), and the average evaluation scores visualized in Figure 4. Overall, participant responses reflect a highly positive perception of the tool, especially in aspects related to ease of navigation and clarity of visual elements. Questions Q1, Q2, and Q5, which concern the clarity and usability of the Overview and Preview screens, consistently received strong scores—typically ranging from 4 to 5—indicating that participants found the interface accessible and easy to interpret. Similarly, Q3 ("Interacting with the Overview screen required little mental effort") and Q6 ("Managing mining tasks in the Jobs screen was intuitive") reinforce the impression that the system does not impose significant cognitive load during standard interactions.

This trend is further corroborated by the data in Figure 4, where most average scores exceed 4. Notably, Q10 ("I would use the tool for data mining") received one of the highest scores overall, underscoring the system's perceived value in real-world professional contexts. At the same time, the lowest-scoring item was Q4 ("The process of mining new data on the Collect screen is clear"), suggesting that while the interface supports data exploration effectively, the experience of initiating new data collections may still lack clarity or guidance.

Additional insights come from Q7 and Q9, which respectively assess whether the tool facilitates data mining tasks and enhances productivity in repository analysis. High scores on both reinforce the idea that users not only navigated the system with ease but

Table 5: Participant Performance Data

TID	P1		P2		P3		P4		P5		P6	
	Time	Answer	Time	Answer	Time	Answer	Time	Answer	Time	Answer	Time	Answer
FT1	5min	✓	3min	✓	2min	✓	3min	✓	6min	✓	11min	✗
FT2		✓		✓		✓		✓		✓		✓
FT3		✓		✓		✓		✓		✓		✓
FT4		✓		✓		✓		✓		✗		✓
FT5		✓		✓		✗		✓		✓		✓
FT6		✗		✓		✓		✓		✓		✗
BT1	4min	✓	3min	✓	9min	✓	4min	✓	5min	✓	9min	✓
BT2		✓		✗		✓		✓		✓		✓
BT3		✓		✓		✓		✓		✓		✓
BT4		✓		✓		✓		✓		✓		✓
AT1	12min	✗	9min	✓	17min	✓	13min	✓	22min	✓	18min	✗
AT2		✓		✗		✗		✓		✓		✗
AT3		✓		✓		✗		✓		✓		✓
AT4		✓		✗		✓		✓		✓		✓

also recognized practical benefits in terms of workflow efficiency. Furthermore, the responses to Q8 ("Collected, viewed, and exported data without difficulties") complement the performance metrics in Table 5, indicating that despite variations in task complexity, users did not encounter major technical barriers when handling data outputs.

Taken together, these findings suggest that users perceive RAISE as both usable and applicable to practical software engineering scenarios. While the interface performs well across most dimensions, the consistently lower evaluation of the data collection component highlights an area where design refinements could improve transparency, reduce friction, and ultimately strengthen user confidence in executing end-to-end analytical workflows.

6 Threats to Validity

Construct and Internal Validity. A primary threat to construct validity stems from the reliance on external APIs for data collection. The RAISE platform requires personal access tokens to authenticate requests to services such as GitHub and Jira. These tokens are subject to platform-specific rate limits and quotas. When such limits are reached, the platform must wait for the quota to reset before resuming requests, which can temporarily delay data collection workflows. Additionally, the correct functioning of the system depends on the appropriate configuration of token permissions. If tokens are generated without the necessary scopes—such as access to repository metadata or issue tracking endpoints—specific mining operations may fail or return incomplete data, not due to a failure in the tool itself, but due to insufficient privileges granted at token creation.

External Validity and Conclusion. Another limitation concerns the scale of our usability evaluation, which involved only seven participants. While the sample included individuals with diverse areas of expertise and varying levels of professional experience, its size constrains the generalizability of our findings. These results should therefore be interpreted as exploratory and indicative, rather than definitive. To strengthen external validity, future replications of this study should include broader and more heterogeneous participant pools. Still, the infrastructure and methodology used provide a replicable foundation for ongoing evaluation and refinement of the platform.

7 Conclusion and Future Work

This paper introduced RAISE, a platform designed to support reproducible and multifaceted analyses of software development data by integrating artifacts from both GitHub and Jira. By offering a unified and structured environment for data collection and exploration, RAISE facilitates empirical investigations that span technical, organizational, and human-centered dimensions of software engineering. The usability experiment conducted as part of this study demonstrated the tool's capacity to assist users in completing practical tasks across multiple complexity levels.

Looking ahead, future work will focus on expanding the platform's functionality in two key directions. First, we plan to extend support for additional platforms and ecosystems beyond GitHub and Jira. This includes the development of new mining modules for services such as Bitbucket, Azure DevOps, GitLab, and Stack Overflow, enabling cross-platform repository mining and broader comparative studies. These modules will follow the same design principles adopted in the existing system—modular, containerized, and compatible with the unified schema.

Second, we aim to introduce analysis modules capable of processing the mined data to generate actionable insights. Planned features include sentiment analysis, toxicity detection, developer productivity estimation, and sensitive information leakage identification. These modules will also enable the automatic generation of visual summaries and metric-based dashboards, assisting both researchers and practitioners in interpreting repository data more effectively. By advancing these capabilities, RAISE seeks to evolve into a comprehensive observability platform for empirical software engineering and development analytics.

ARTIFACT AVAILABILITY

Our artifacts are available at https://doi.org/10.5281/zenodo.16997997.

ACKNOWLEDGMENTS

This research was partially funded by the Brazilian funding agencies CAPES (Grant 88881.879016/2023-01) and FAPESP (Grant 2023/00811-0). We also acknowledge the Brazilian company Stone[13] for their financial support.

[13] https://www.stone.com.br/

REFERENCES

[1] Mohammad Almarzouq, Abdullatif Alzaidan, and Jehad AlDallal. 2020. Mining GitHub for research and education: challenges and opportunities. *International Journal of Web Information Systems* 16, 5 (2020), 549–567. doi:10.1108/IJWIS-03-2020-0016

[2] Milica Avramovska, Elizabeta Hristovska, and Sonja Calamani. 2024. Applying Jira – A Tool for the Organization and Optimization of Work Processes in the Machine Industry Based on the Experience of the IT Industry. *SAR Journal* 7, 4 (2024), 289–295. doi:10.18421/SAR74-01

[3] Diego Castro and Marcelo Schots. 2018. Analysis of Test Log Information through Interactive Visualizations. In *Proceedings of the 26th IEEE/ACM International Conference on Program Comprehension (ICPC '18)*. ACM, Gothenburg, Sweden, 156–166. doi:10.1145/3196321.3196345

[4] K.K. Chaturvedi, V.B. Singh, and Prashast Singh. 2013. Tools in Mining Software Repositories. In *13th International Conference on Computational Science and Its Applications*. IEEE, 89–98. doi:10.1109/ICCSA.2013.22

[5] Sander Coremans, Jakob Krüger, and Dirk Fahland. 2023. Process Mining from Jira Issues at a Large Company. In *CAiSE Forum*. https://jacobkrueger.github.io/assets/papers/Coremans2023ProcessMiningTFS.pdf

[6] GitPython Developers. 2025. GitPython: Python library to interact with Git repositories. https://github.com/gitpython-developers/GitPython. Versão 3.1.44, acessado em 22 de abril de 2025.

[7] Sergio Dueñas, Jesus M. Gonzalez-Barahona, Gregorio Robles, Víctor Cosentino, Daniel Izquierdo-Cortázar, Dominguez Fernandez, and Andrea Capiluppi. 2021. GrimoireLab: A toolset for software development analytics. *PeerJ Computer Science* 7 (2021), e601. doi:10.7717/peerj-cs.601

[8] Jesus M. Gonzalez-Barahona and Gregorio Robles. 2023. Revisiting the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Information and Software Technology* 158 (2023), 107191. doi:10.1016/j.infsof.2023.107191

[9] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: GitHub's Data from a Firehose. In *Proceedings of the 9th Working Conference on Mining Software Repositories (MSR)*. 12–21. doi:10.1109/MSR.2012.6224294

[10] Georgios Gousios and Diomidis Spinellis. 2017. Mining Software Engineering Data from GitHub. In *Proceedings of the 39th International Conference on Software Engineering Companion*. 503–506. doi:10.1109/ICSE-C.2017.164

[11] Monika Gupta, Ashish Sureka, and Srinivas Padmanabhuni. 2014. Process Mining Multiple Repositories for Software Defect Resolution from Control and Organizational Perspective. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. 122–131. doi:10.1145/2597073.2597081

[12] Nicole Hoess, Carlos Paradis, Rick Kazman, and Wolfgang Mauerer. 2025. Does the Tool Matter? Exploring Some Causes of Threats to Validity in Mining Software Repositories. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 645–656.

[13] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071. doi:10.1007/s10664-015-9393-5

[14] Thomas Krismayer, Christoph Mayr-Dorn, Johann Tuder, Rick Rabiser, and Paul Grünbacher. 2019. Using Constraint Mining to Analyze Software Development Processes. In *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. 94–103. doi:10.1109/ICSSP.2019.00021

[15] Marek Macura. 2014. Integration of data from heterogeneous sources using ETL technology. *Computer Science* 15, 2 (2014), 109–124. doi:10.7494/csci.2014.15.2.109

[16] Rita Marques, Miguel Mira da Silva, and Diogo R. Ferreira. 2018. Assessing Agile Software Development Processes with Process Mining: A Case Study. In *20th Conference on Business Informatics*. doi:10.1109/CBI.2018.00021

[17] Lloyd Montgomery, Clara Lüders, and Walid Maalej. 2022. An alternative issue tracking dataset of public Jira repositories. In *Proceedings of the 19th International Conference on Mining Software Repositories*. 73–77. doi:10.1145/3524842.3528486

[18] Lloyd Montgomery, Clara Lüders, and Walid Maalej. 2024. Mining Issue Trackers: Concepts and Techniques. In *MSR 2024 Conference Proceedings*. doi:10.1007/978-3-031-73143-3_11

[19] Breno Neves, Daniel Coutinho, Eduardo Sardenberg, Arthur Alesi, Marcelo Machado, Johny Arriel, Ana Luísa Cavalcante, and Juliana Alves Pereira. 2025. Replication Package: RAISE. https://github.com/aisepucrio/raise-replication/.

[20] Mikko Raatikainen, Quim Motger, Clara Marie Lüders, Xavier Franch, Lalli Myllyaho, Elina Kettunen, Jordi Marco, Juha Tiihonen, Mikko Halonen, and Tomi Männistö. 2022. Improved management of issue dependencies in issue trackers of large collaborative projects. *IEEE Transactions on Software Engineering* 49, 4 (2022), 2128–2148. doi:10.1109/TSE.2022.3212166

[21] Ezequiel O Ramos and Rogério Rossi. 2023. Process Mining Applied in a Software Project Development with SCRUM and ProM. *European Journal of Engineering and Technology Research* 8, 5 (2023), 17–24. doi:10.24018/ejeng.2023.8.5.3089

[22] Daniela Rodriguez, Andy Zaidman, and Arie van Deursen. 2024. Sharing Software-Evolution Datasets: Practices, Challenges, and Recommendations. In *Proceedings of the 46th International Conference on Software Engineering (ICSE)*. ACM. doi:10.1145/3660798

[23] Tamanna Siddiqui and Ausaf Ahmad. 2017. Data Mining Tools and Techniques for Mining Software Repositories: A Systematic Review. 654 (2017), 717–726. doi:10.1007/978-981-10-6620-7_70

[24] Francisco Zigmund Sokol, Mauricio Finavaro Aniche, and Marco Tulio Valente Gerosa. 2013. MetricMiner: Supporting Researchers in Mining Software Repositories. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 125–134. doi:10.1109/SCAM.2013.6648195

[25] Davide Spadini, Maurício Aniche, and Alberto Bacchelli. 2018. PyDriller: Python Framework for Mining Software Repositories. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 908–911. doi:10.1145/3236024.3264598

[26] Melina Vidoni. 2022. A Systematic Process for Mining Software Repositories: Results from a Systematic Literature Review. *Information and Software Technology* 144 (2022), 106791. doi:10.1016/j.infsof.2021.106791

[27] Jiuang Zhao, Zitian Yang, Li Zhang, Xiaoli Lian, Donghao Yang, and Xin Tan. 2024. DRMiner: Extracting Latent Design Rationale from Jira Issue Logs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 468–480. doi:10.1145/3691620.3695019