

VILNIUS GEDIMINAS TECHNICAL UNIVERSITY

FACULTY OF FUNDAMENTAL SCIENCES

dEPARTMENT OF INFORMATION SYSTEMS

Davina Jackson

Student ID: 20241921

**Development Project Quality Assessment using event-based methods**

Master Graduation Thesis

Supervised by: Prof. Dr. Titas Savickas

Vilnius, 2025

**Table** **of Contents**

[1. Introduction 3](#_Toc200994974)

[1.1. Investigation Object 4](#_Toc200994975)

[1.2. The Aim and Tasks of the Thesis 4](#_Toc200994976)

[1.3. Novelty of the Topic 4](#_Toc200994977)

[1.4. Relevance of the Topic 5](#_Toc200994978)

[1.5. Research Methodology 5](#_Toc200994979)

[1.6. Scientific Value of the Thesis 6](#_Toc200994980)

[1.7. Main Results of the Thesis 6](#_Toc200994981)

[1.8. Structure of the Work 6](#_Toc200994982)

[2. Related Works Analysis 6](#_Toc200994983)

[2.1. Detailed Review of OSS Project Quality Key Related Works 8](#_Toc200994984)

[2.2. OSS Project Quality Related Works 13](#_Toc200994985)

[2.3. Analysis of GitHub Data Events 19](#_Toc200994986)

[2.4. Process Mining 22](#_Toc200994987)

[2.5. Statistical Analysis Methods 26](#_Toc200994988)

[2.6. Main Results of the 2nd Section 28](#_Toc200994989)

[3. Proposed Approach 30](#_Toc200994990)

[3.1. Main Results of the 3rd Section 33](#_Toc200994991)

[4. Conclusions 33](#_Toc200994992)

[5. References 34](#_Toc200994993)

**List of Images**

Figure 2.2.1. Types of approaches used in column 2 within Table 2.2.1. 18

Figure 2.4.1. 3 basic types of process mining in terms of input and output 23

Figure 2.4.2. Process models discovered for a purchase-to-pay process showcasing only the frequent desired outcomes 25

Figure 3.1. GitHub Repository Reliability Analysis Process. 31

Figure 3.2. Repository Data Collection and Pre-Processing Sub Process. 31

Figure 3.3. Data Extraction Sub Process. 32

Figure 3.4. Repository Process Evaluation Sub Process. 32

**List of Tables**

Table 2.2.1. Summary of 27 research papers based on OSS project repository reliability issues identification methods and results of good and bad OSS project development practices. 13

Table 2.3.1. Important common shared properties within all GitHub event objects inside of the GitHub Events API. 21

Table 2.3.2. Issues event payload object properties 21

Table 2.4.1. Open-Source Process Mining Tools Comparison. 25

Table 2.4.2. Process Mining Algorithms Summary. 26

Table 2.5.1. Predictive Analysis Models/Methods Summary. 28

**Abbreviations**

OSS: Open-source software

SLR: Scientific literature review

HEP Software: High Energy Physics software

BERT: Bidirectional Encoder Representations from Transformers

ITS: Issue Tracking System

NPM Packages: Node Package Manager packages

RBF Neural Network: Radial Basis Function network

API: Application Programming Interface

DFG: Directly-Follows Graph

PR: Pull Request

BPMN: Business Process Model and Notation standard

XES: eXtensible Event Stream

CSV: Comma-Separated Values

GUI: Graphical User Interface

# Introduction

The problem of repository reliability issues identification within open-source software (OSS) projects has become a more relevant topic within the last few years (Macak et al., 2022). Methods of identifying reliability issues or project quality are sparse or unknown. Due to its widespread use, OSS serves as a critical social infrastructure requiring reliability and availability, but unlike traditional software, it lacks a formal guarantor for its reliability, while its open nature enables potentially superior development capabilities (Miyamoto et al., 2023). Data mining involves extracting meaningful patterns and insights from large datasets, enabling software engineers to identify patterns, detect potential defects, and support key tasks like programming, testing, and maintenance by transforming data into actionable knowledge (Canaparo & Ronchieri, 2018).

This research answers the question how can we identify reliability issues within OSS project repositories?

## Investigation Object

The investigation object is methods for ensuring reliability and project quality of OSS project repositories.

## The Aim and Tasks of the Thesis

The aim of the research is to improve open-source software project management quality using event-based analysis methods.

The tasks were performed to achieve the aim of the thesis. They are as follows:

1. To analyze and compare the methods used to identify software project repository reliability issues from the perspective of expanding on good project development practices.
2. To propose a method to assess quality of open-source projects based on their repository history.
3. To perform experiments to evaluate the performance of the proposed method.

## Novelty of the Topic

Research on OSS reliability is often constrained by small datasets, poor sampling methods, and inconsistent methodologies, making findings difficult to generalize or replicate (Cosentino et al., 2017). Existing metrics for evaluating OSS project health are not standardized, resulting in limitations of their practical application (Linåker et al., 2022). Additionally, socio-technical factors, such as commit timing, developer roles, and interaction patterns, also play a critical role in project outcomes but remain underexplored (Eiroa-Lledo et al., 2023).

## Relevance of the Topic

Ensuring the reliability and quality of OSS project repositories is critical because of their foundational role in modern software infrastructure. They support a significant portion of modern technology, powering applications ranging from web services to enterprise solutions and IoT devices (Brittain & Darwin, 2003). However, OSS faces significant challenges, including high failure rates, with fewer than 50% of projects surviving beyond five years, and erratic development patterns (Ait et al., 2022). Fault management varies widely, as repair times depend on developer expertise and issue severity (Miyamoto et al., 2023). Developers encounter practical issues like error-prone configuration files, insufficient testing before pull requests, and suboptimal use of continuous integration tools (Saroar & Nayebi, 2023). Additionally, many bug reports lack essential details, such as crash reproduction steps and stack traces, which delay resolutions and affect reliability (Soltani et al., 2020).

Addressing these challenges is essential to enhance OSS sustainability, reliability, and adoption. Tackling standardizing bug reporting, streamlining fault management, and refining predictive models, data mining, and process mining would significantly augment OSS project outcomes. OSS project repositories contain rich data on code commits, bug reports, issue tracking, and developer interactions, which can be analyzed to identify patterns, predict defects, and improve project management (Brittain & Darwin, 2003). By improving open-source software project management quality using event-based analysis methods, this research will help ensure OSS projects continue to serve as a reliable and essential part of the global software ecosystem.

## Research Methodology

Information analysis such as a systematic literature review showcasing a comparative table of research papers has been used to analyze OSS project repository reliability issues identification methods and results of good OSS project development practices.

## Scientific Value of the Thesis

The top methods found that identify software project repository reliability issues from the perspective of expanding on good project development practices were data or process mining, empirical studies, deep machine learning, and systematic mapping studies.

## Main Results of the Thesis

Analysis of related literature and existing identification methods of software project repository reliability issues shows that predictive models, data mining, and process mining for defect detection and reliability assessment show potential but require further development to improve accuracy and accessibility.

## Structure of the Work

The second section deals with summarizing 27 research papers using a table, highlighting 10 important papers from the 27 total research papers, analyzing GitHub events, and researching process discovery.

# Related Works Analysis

In this section, a scientific literature review (SLR) of 27 research papers is performed based on open-source software (OSS) project repository reliability issues identification methods and results of good and bad OSS project development practices. There are multiple terms involved such as OSS project repository, bug, HEP software, software reliability, machine learning, deep learning, data mining, literature review, empirical study or research, and systematic mapping study.

Open-source software is released with a license allowing anyone to view, modify, and distribute its source code (Brittain & Darwin, 2003). This approach contrasts with proprietary software, where the source code is typically closed to the public, and users are restricted in how they can modify or share it. An OSS project repository refers to the online storage location where the codebase, documentation, and other resources for an open-source software project are maintained and shared (Brittain & Darwin, 2003). A popular OSS project repository is GitHub. Key components of OSS project repositories are source code, version control/commits, labels, README, license, issues and pull requests, documentation, continuous integration, and community guidelines (Brittain & Darwin, 2003). A bug is a defect in software applications (Miyamoto et al., 2023).

Bug tracking systems allow users to log, report, track, and resolve issues that arise during the development and maintenance of a project (Sriya & Srinivas, 2019). Popular bug tracking systems are Bugzilla and Jira. HEP software stands for High Energy Physics Software. It's a specialized field of software development focused on addressing the computational and data analysis needs of high energy physics research (Canaparo & Ronchieri, 2018).

Software reliability refers to the probability that software will work properly in a specified environment and for a given amount of time (Nassar, 1986).

Machine learning, a branch of artificial intelligence, focuses on creating statistical algorithms that learn from data to perform tasks without explicit programming, with deep learning advancements enabling neural networks to achieve superior performance over earlier methods (Takahashi & Takahashi, 2024).

Deep learning, a subset of machine learning inspired by biological neuroscience, utilizes neural networks with multiple layers to perform tasks like classification, regression, and representation learning through supervised, semi-supervised, or unsupervised methods (Singh et al., 2021).

Data mining, an interdisciplinary field combining machine learning, statistics, and database systems, involves extracting patterns and transforming large datasets into comprehensible structures as part of a knowledge discovery (Bunge & Judson, 2004).

A literature review provides an overview of existing knowledge on a topic, situating the current study within relevant research while guiding the development of appropriate questions, theoretical frameworks, and methodologies, often preceding the methodology and results sections in scholarly works (Rodriguez, 2003).

Empirical research involves gaining knowledge through direct or indirect observation and experience, using qualitative, quantitative, or mixed methods to analyze data and answer clearly defined, evidence-based questions tailored to the research field and context (Given, 2012).

A systematic mapping study categorizes, analyzes, and synthesizes existing literature on a specific research topic to offer a comprehensive overview, highlight research gaps, and inform future directions in computer science (Salama et al., 2017).

These concepts and approaches collectively form the foundation for understanding and improving the reliability and quality of OSS project repositories. By exploring these interconnected areas, this analysis aims to address the challenges faced in managing, maintaining, and enhancing OSS projects in a rapidly evolving technological landscape.

## Detailed Review of OSS Project Quality Key Related Works

This section provides an in-depth review of 10 key articles selected from the SLR that address critical aspects of OSS project quality, focusing on methodologies, challenges, and best practices in the field. By examining diverse approaches and findings from these studies, this review highlights the evolving strategies for assessing, maintaining, and enhancing the quality of OSS project repositories.

In Alshara et al. (2023), a new dataset named PI-Link composed of 50369 links that explicitly connect 34732 issues with 50369 pull-requests is proposed. The links were extracted from 907,139 Android projects in GitHub created between January 1st, 2011 and January 1st, 2021. No data set to the authors’ knowledge has yet been proposed that can train machine learning models. To organize and store the collected data, a metamodel based on issues and pull-requests concepts is proposed. The stored data’s relationships are analyzed based off the length and similarities between their titles, bodies, labels, and comments using box-whisker plots of Jaccard, Cosine, Levenshtine, and BERT similarity metrics. Their results showed promising lexical and semantic similarities between the issues and their linked pull-requests, and sensitivity between the analyzed features are discovered such as text-length and term frequency. The main limitation of this solution is that other domain-based projects are not included in their dataset, which authors plan to extend their dataset in the future to provide inclusivity.

In Reszka et al. (2023), authors mined software repository content within Jira and Bugzilla ITS systems in relation to the individual repositories’ specificities and proposed a different data mining approach or analysis, in comparison to classical data mining, that presents a holistic image of the project development process. Developed analysis schemes are usually based on simplified data models while issue report details are neglected, so instead, an approach is proposed that involves original evaluation profiles and metrics related to issue reporting aspects and the activity of project actors. The structural and semantic exploration of repositories, characteristic features in value and time perspectives, and the space of project monitoring goals are three aspects targeted. They verified their original and universal analysis using open-source and some commercial software project repositories. Their study found valuable information in areas like attributes that are neglected in other literatures. However, their main limitation was the lack of feedback from the stakeholders of the open-source projects analyzed and data confidence. In the future, the authors would like to extend their analysis methodology to be enhanced with other repositories like GitHub.

In Cosentino et al. (2017), 80 GitHub publications from 2009 to 2016 were assessed using a systematic mapping study with four research questions. With data mining of software repositories on GitHub becoming a common theme, the authors think it is worthwhile to reflect on these research papers and report their findings. Their four research questions used to analyze the publications were the following: what topics/areas have been addressed, what empirical methods have been used, what technologies have been used to extract and build datasets from GitHub, and what is the research community behind these works like. First, they found their publications using six digital libraries, performed a breadth-first search using backward and forward snowball methods, and then applied a title/abstract pruning process to come up with 342 publications. Second, they took those publications and applied a two-phased selection process to determine which publications were relevant in answering their research questions and ended with 80 publications. They found from their study that topics: development, projects, users, and the GitHub ecosystem itself have been addressed, empirical methods: metadata observation, surveys, interviews, and a mixture of methods have been used, technologies addressing data collection process and dataset size and availability have been used, and the community is composed of researchers and publication fora. Their main limitation is the threat to validity due to the search process and selection criteria.

In Ait et al. (2022), authors empirically study 1,127 GitHub repositories created in 2016 from NPM packages, R packages, WordPress plugins, and Laravel packages, which are ecosystems devoted to create packages or plugins for well-known platforms and in this study helps restrict the number of toy projects collected, for early project development dynamics. The quantity of OSS projects is constantly growing, and many are not regularly maintained with some even abandoned shortly after creation. The authors stored their chosen GitHub repositories’ activity in a time series database and analyzed the activity using state machines and labels from 2016 to the creation of this study in 2022. They found that the prototypical development process consists of intensive coding-driven active periods followed by long periods of inactivity, a significant number of projects die in the first year of creation with the survival rate decreasing year after year, and the probability of survival being longer than five years is less than 50%. The main limitations from this study would be the right-censoring of the data since the exact survival time of the dataset is unknown due to study limits as a project analyzed could reactivate after the six-year period considered in the study and the ecosystems / dataset chosen since the identification of actual software developing projects is a challenging task. In the future, authors hope to expand the number of projects considered in their study to cover other ecosystems and construct an intelligent system able to forecast the survivability of OSS projects and to flag those projects at risk of dying soon so that corrective community actions can be put in place.

In Kim & Lee (2021), authors empirically study GitHub multiple and custom labels on Issues to provide a better understanding of their usage in software development. Previous studies on labels were limited to simple statistics or label recommendations for issue types. They collected 14,415 projects with 13 million issues and 0.3 million labels from the GHTorrent dataset and updated data with recent GitHub information via GitHub REST API, quantitatively investigated the performance of projects with multi-label features, and qualitatively investigated the categories of multi labels and the usage of multi-labels based on these categories. Their analysis results showed that multi-labels are commonly used in most OSS projects, OSS projects using multi-label features manage their issues more effectively, and different types of information represented by custom labels are related to features, development, and issue concepts. The main limitation is the subject refinement and original analysis methods used that may have introduced bias to their dataset.

In Eiroa-Lledo et al. (2023), leveraging the ManySStuBs4J dataset, authors applied a custom traversal algorithm to commits made for OSS project repositories to determine when “simple stupid bugs” were first introduced to GitHub projects and to explore the factors relating to the time it takes to fix them. A prominent issue in OSS projects is buggy code and even though a check-and-balance structure effectively minimizes the amount of buggy code, this issue remains. Authors used the commit history from the main development branch to identify the commit that first introduced 13 different types of simple stupid bugs in 617 top Java projects on GitHub. Then using statistical survival analysis methods and other non-parametric statistical tests such as the Mann-Whitney U Tests, Hodges-Lehman Median Difference and Kaplan-Meier estimates, and the Cox Proportional-Hazards model, the authors found that Time Factors and Author Factors were the main categorical variables that affect a bug’s life in which bugs are fixed 1.71 times quicker if they are introduced and resolved by the same developer, bug fixing is slower for projects with more than 40 code authors and more than 25 code committers, and the day and time a buggy code was written and fixed affects resolution time. The main limitation is the ManySStubBs4J dataset due to some projects missing important information such as the project owner, project references, and bug type definitions.

In Macak et al. (2022), authors conduct a systematic literature review covering 35 relevant research approaches to examine how process mining is currently used in relation to cybersecurity and software reliability and to identify the current research gaps and promising research directions. Their belief is that studies exist that state how process mining can help and be practically used in this context, however an overview is needed and coverage is limited. Authors collected existing process mining applications using bias free methods like filtering by multiple researchers, snowballing, and manual non-systematic search using search engines. They then discussed current trends and promising research directions that can be used to tackle the current cybersecurity and software reliability challenges. The results showed numerous research gaps, especially in software reliability, and nine major research directions such as security of industrial control systems, security of smartphones, network traffic security, web application security, attack inspection, outlier user behavior detection, fraud detection, quality assurance, and error detection. Even though the systematic literature review strategy proposed by Zhou et al. (2016) was followed, the authors’ main limitation is the search string query, exclusion criteria, and inclusion criteria while scanning amongst the top 6 digital databases.

In Linåker et al. (2022), authors conduct a snowball literature review based on a start set of 9 papers and identify 146 relevant papers, with two iterations of forward and backward snowballing, that create an overview of characteristics that affect the health of an OSS project and enable the assessment thereof. 107 health characteristics, addressing the socio-technical spectrum of the community of actors maintaining the OSS project, the software and other deliverables being maintained, and the orchestration facilitating the maintenance, are elicited and coded using structured and axial coding into a framework structure that is divided, based on the level of abstraction they address or the project’s ecosystem of related OSS projects, among 15 themes. The identified studies confirm the importance of not analyzing an OSS project in isolation, that its dependencies and ties to other projects play an important part in terms of resilience and security. The main limitation is that the framework provides limited guidance in terms of which characteristics to consider and how. In future research, the authors aim to address this gap through further iterations to design a more mature framework with related processes that can be tailored based on organizational context and requirements.

In Canaparo & Ronchieri (2018), authors compare diverse data mining techniques using NASA Defect, Eclipse, Android, and Elastic Search datasets within three OSS tools: Weka, Scikit Learn and R for developing effective software quality prediction models. Software quality monitoring and analysis are among the most productive topics in software engineering research and OSS constitutes a valid test case for the assessment of software characteristics. The authors first conducted research about software quality prediction to identify defect-prone software modules and then they attempted to reproduce and expand previous data mining technique studies. They collected data mining techniques such as Support Vector Machine, Decision Tree, Naïve Bayes, Ensemble Classifier, Multi Layer Perceptor, AdaBoost, Bagging, RandomForest, J48, K-Nearest Neighbor, RBF Neural Network, and Deep Learning from existing literature, collected software metrics from OSS repositories such as McCabe, Halstead, Size, and Chidamber and Kemerer, assessed prediction models to detect software issues, and adopted statistical methods to evaluate data mining techniques. By analyzing the results, they conclude that Bagging and Random Forest have the best average accuracy over all the datasets analyzed in their study. They also specify that in previous literatures, datasets related to OSS projects and Deep Learning techniques are not considered often, so they made sure to include them due to their growing popularity.

In Perez-Castillo & Piattini (2021), authors present an empirical research multiple case study, using guidelines from PER RUNESON et al. (2012), that follows a repository mining approach based on statistical methods and analyses 13 OSS projects from GitHub and SonarCloud, which retrieves more than 95,000 commits, 782 builds (with more than 90 different metrics measured by each build), and more than 25,000 quality measures. There is no thorough research about how code quality is affected by the software development projects’ contexts. This study analyses how the evolution of the development effort (the number of developers and their contributions) influences the code quality (the number of bugs, code smells, cloning). First, GitHub supports the code repositories of these projects and second Sonarcloud is queried to get information about builds and associated software metrics such as bugs, code smells, violations, duplications and once the data was collected and pre-processed, some correlation (the Spearman correlation test) and clustering (K-means based off of machine learning) algorithms were applied to figure out relationships between software development effort and software quality measures. The authors found that more developers or higher number of commits does not necessarily influence worse quality levels, after applying the clustering algorithm an inverse correlation in some cases were detected where specific efforts were made to improve code quality, and the size of commits and the relative weight of developers in their teams might also affect complexity or cloning measures which shows that the relationship between the team efforts and software quality can vary. The main limitation of this study is that SonarCloud was the only source of code quality measures and the correctness of these measures must be supposed.

## OSS Project Quality Related Works

The main results of the SLR are presented in Table 2.2.1. It consists of 6 columns which are the following: Reference (Column 1), Used Approach / Method (Column 2), Evaluation of Approach (Column 3), Data Set / Algorithm Used (Column 4), OSS Type (Column 5), and Main Results (Column 6).

**Table 2.2.1.** Summary of 27 research papers based on OSS project repository reliability issues identification methods and results of good and bad OSS project development practices.

| **Reference** | **Used Approach / Method** | **Evaluation of Approach** | **Data Set / Algorithm Used** | **OSS Type** | **Main Results** |
| --- | --- | --- | --- | --- | --- |
| (Miyamoto et al., 2023) | Proposes a reliability assessment method based on deep learning using software repository analysis by estimating the time to failure. | Reliability  evaluation using the information in issues | RReLU and Radam algorithms | GitHub | Software repair time is dependent on severity of the fault incurred and the software development capability of the developer.  Software development capability improves with the popularity of OSS because project volunteers can propose improvements to the source code, and repair time becomes shorter.  OSS users can estimate the number of days to fix a fault and the time to be exposed to a fault, and thus understand the reliability of OSS. |
| (Cosentino et al., 2017) | A systematic mapping study was conducted with four research questions and assessed 80 publications from 2009 to 2016. | Cross analysis | N/A | GitHub | Proposed some actions to mitigate reliability concerns that papers used small data sets, poor sampling techniques, employed a scarce variety of methodologies, and/or were hard to replicate. |
| (Sayago-Heredia et al., 2021) | A systematic mapping study to find, evaluate  and investigate the mechanisms, methods and techniques used for the analysis of information from code repositories that allow the understanding of the evolution of software. | Quality assessment and review protocol | N/A | N/A | Identified the main information used as input for the analysis of code repositories (commit data and source code), as well as the most common methods and techniques of analysis (empirical/experimental and automatic). |
| (Ait et al., 2022) | An empirical study on 1,127 repositories from four different ecosystems created in 2016 and stored their activity in a time series database and analyzed their activity evolution along their lifespan, from 2016 to now. | Existing works comparison | NPM packages, R packages, WordPress plugins, and Laravel packages | GitHub | Prototypical development process consists of intensive coding-driven active periods followed by long periods of inactivity.  A significant number of projects die in the first year of existence with the survival rate decreasing year after year.  Probability of surviving longer than five years is less than 50% though some types of projects have better chances of survival. |
| (Kim & Lee, 2021) | An empirical study on performance of projects with multi-label or custom features and qualitatively investigated the categories of multi-labels, and the usage of multi-labels based on these categories. | Existing works comparison | GHTorrent dataset | GitHub | Multi-labels are common in the majority of software projects and allows projects to manage their issues more effectively.  Different types of information represented by labels, which are related to features, development, and issues. |
| (Saroar & Nayebi, 2023) | We conducted a survey study with 90 Action users and developers. | Statistical and qualitative analysis | N/A | GitHub | Developers prefer Actions with verified creators and more stars when choosing between similar Actions, and often switch to alternative Actions when faced with bugs or a lack of documentation.  Developers find the composition of YAML files challenging and error-prone. They primarily rely on forums to fix issues with these YAML files.  Developers would not likely adopt Actions when there are concerns around complexity and security risks. |
| (Reszka et al., 2023) | Analysis methodology including structural and semantic exploration of repositories, deriving characteristic features in value and time perspectives, and defining the space of project monitoring goals. | Existing works comparison | N/A | Jira and Bugzilla | Holistic image of the project development process, which is useful in the assessment of its efficiency and identification of imperfections. |
| (Eiroa-Lledo et al., 2023) | Custom traversal algorithm to commits made for open-source repositories to determine when “simple stupid bugs” were first introduced to projects and explore the factors that drive the time it takes to fix them. | Statistical survival analysis methods and non-parametric statistical tests | ManySStuBs4J dataset | GitHub | Able to identify the commit that first introduced 13 different types of simple stupid bugs in 617 of the top Java projects on GitHub.  Two main categories of categorical variables that affect a bug’s life; Time Factors and Author Factors.  Bugs are fixed quicker if they are introduced and resolved by the same developer.  Day of the week and time of day buggy code was written and fixed affects its resolution time. |
| (Puangjaktha et al., 2024) | Proposed Deep learning-based algorithm for automatically matching scholarly articles with their corresponding official code repositories. | Model comparison | Papers With Code and S2AG dataset | GitHub | Most common linking information includes the paper title and BibTeX entries, typically found in the repository’s readme document.  Utilizing these embedding representations with the Light Gradient Boosting Machine (LGBM), achieved an F1 score of 0.94.  Model comparison with a rule-based approach improved performance by 5.31%. |
| (Alshara et al., 2023) | Proposed PI-Link, a new significant and reliable ground-truth dataset composed of 50369 links that explicitly connect 34732 Issues with 50369 Pull-Requests. | Analyzed relationships | PR-Issue-Scraper Algorithm | GitHub | Promising similarities between Issues and their linked PRs at the lexical and semantic levels. Some feature similarities are sensitive to the text length, whereas other feature similarities are sensitive to the term frequency. |
| (Garcia et al., 2019) | Systematically map the active research topics of process mining and their main publishers by country, periodicals, and conferences. | Existing works comparison | N/A | N/A | Overview regarding process mining is presented.  Identification of the most applied process mining algorithms  Application domains among different business segments are reported on.  Most active research topics are associated with process discovery algorithms, conformance checking, and architecture and tools improvements. |
| (Macak et al., 2022) | A systematic literature review covering 35 relevant research approaches to examine how process mining is currently used for these tasks and what are the research gaps and promising research directions in the area. | Existing works comparison | N/A | N/A | Coverage is still rather limited, with numerous research gaps, especially in software reliability.  Nine major research directions, discussed how they fit in the overall landscape and presented how they utilize process mining for these purposes. |
| (Sattler et al., 2023) | Proposed SEAL, maps repository information, mined from the development history of a project, onto a low-level intermediate program representation, making it available for state-of-the-art program analysis. | 13 open-source projects | IDE algorithm | N/A | Determines which code changes modify central parts of a given software project, how authors interact (indirectly) with each other through code, and demonstrate that putting static analysis’ results into a socio-technical context improves their expressiveness and interpretability. |
| (Nizam, 2022) | Researched software engineering literature and case studies to gather information about critical incidents and repeating behaviors of teams in failed projects into a novel dataset. Grounded theory was employed to build a theoretical foundation for failure phase definitions from the collected data. | Cross-validation | N/A | N/A | Common behavioral patterns occurred in approximately 89 percent of the case studies, supporting the decision to consider software project failure as a process.  A simple structure that uses everyday concepts for phase names and reveals the critical behaviors leading a software project to failure. |
| (Li et al., 2022) | A comprehensive study of the source code, code change history, and issue reports of ten open-source Java projects, combining quantitative and qualitative analysis. | Cross-validation | Scott-Knott clustering algorithm | Java | Analysis of the models indicates the important factors for determining the logging of exception stack traces.  The current practices of logging exception stack traces, recommendations for developers to consider when determining whether to log the stack trace of an exception, and insights for future research and practices to derive global or company-wide guidelines for the logging of exception stack traces. |
| (Soltani et al., 2020) | Interviewed 35 developers to gain insights into their perceptions on the importance of various contents in bug reports. Surveyed 305 developers to assess findings. Mined issue repositories of 250 most popular projects to evaluate the quality of currently available bug reports. | Surveys and mining | N/A | GitHub | Crash reproducing steps, stack traces, fix suggestions, and user contents, have statistically significant impact on bug resolution times, for ∼70%, ∼76%, ∼55%, and ∼33% of the projects.  Over 70% of bug reports lack these elements.  Developers find it highly important that bug reports include crash description, reproducing steps or test cases, and stack traces.  Software version, fix suggestions, code snippets, and attached contents have lower importance for software debugging. |
| (Romanov et al., 2024) | Creating time series models and forecasting their behavior. A proposed approach for software project analyses based on fuzzy logic principles. | Fuzzy logic | Computational Intelligence in Forecasting (CIF) 2015 and SMAPE datasets | GitHub and GitLab | The completed experiments showed the effectiveness of the suggested forecast method; the time cost for repository analysis was shortened by a factor of 4 on average.  Various repository evaluation methods were described and researched.  A method that allows the repository evaluation based on analyzing the dynamic of its metrics was developed. |
| (Wrobel et al., 2023) | An empirical study on repositories to explore the use of continuous integration techniques in open-source projects. | Existing works comparison | N/A | GitHub | Most of the projects studied have not yet developed a mature approach to using continuous integration techniques.  Developers do not thoroughly test code before submitting pull requests.  Developers tend to submit pull requests with small amounts of new or modified code. |
| (La Cholter et al., 2021) | Mined 1,835 repositories for already-compiled malicious files and data suggesting whether the repository is malware-related. Extracted all Portable Executable files from all commits and queried VirusTotal for analysis. | VirusTotal | N/A | GitHub | Of the 24,395 files, 4,335 are suspicious, with at least one detection; 440 could be considered malicious, with at least seven detections.  Topic tags identified suggesting malware or offensive security content, to differentiate from seemingly benign repositories.  197 of 440 malicious executables were in 27 ostensibly benign repositories. |
| (Tamura & Yamada, 2023) | Analsis using deep machine learning based on fine-tuning for reliability/quality evaluation. | Model comparison | Large quantity fault data sets | Bugzilla | Useful to use the fine-tuning from the standpoint of cost reduction and time saving in deep machine learning.  The transfer-learning and fine-tuning have been compared in order to assess the reliability/quality of the deep machine-learning model. |
| (Mahmood et al., 2021) | Proposes fault prediction approach using a data-mining technique to find good predictors for high-quality software. | An experimental study | Apriori algorithm | Hazelcast | The evaluation showed that the results of proposal are promising.  Practitioners and developers can utilize these rules for defect prediction during early software development. |
| (Haider et al., 2023) | Two systematic literature reviews are employed for the identification of potential risk factors in OSS and identification of practices for softening the effect of risk-factors in OSS development. | Questionnaire survey | N/A | N/A | Identified 14 risk factors and 31 practices for mitigating the critical risk factors.  Focusing on the identified risk factors would minimize the risks associated with OSS. |
| (Wang et al., 2024) | Proposed a multi-release OSS reliability model based on three-parameter lifetime distribution. | Mean square error, predictive sum of squares error, R2, theil statistic, and variance | OMID 0.9.0.0, OMID 1.0.0, OMID 1.0.1, GOBBLIN 0.12.0, GOBBLIN 0.13.0,  and GOBBLIN 0.14.0 datasets | Apache Omid and Gobblin | Has better fitting and predictive performance compared with other multi-release OSS reliability models.  Can better adapt to the variety of OSS fault detection environment, and assist developers to evaluate the reliability of OSS. |
| (Linåker et al., 2022) | A snowball literature review conducted based on a start set of 9 papers, and 146 relevant papers over two iterations of forward and backward snowballing are identified. Health characteristics are elicited and coded using structured and axial coding into a framework structure. | Existing works comparison | N/A | N/A | An overview of characteristics that affect the health of an OSS project and enable the assessment thereof.  Final framework consists of 107 health characteristics divided among 15 themes. |
| (Canaparo & Ronchieri, 2018) | Compared data mining techniques derived from machine learning, the HEP software, for developing effective software quality prediction models. | Assessed prediction models | NASA Defect, Eclipse, Android, and Elastic Search | Weka, Scikit Learn and R | Collected software metrics from open-source repositories.  Assessed prediction models to detect software issues.  Adopted statistical methods to evaluate data mining techniques. |
| (Hassani, 2018) | Performed an empirical study on log-related issues in two large-scale, open-source software systems. Then conducted a manual study and identified seven root-causes of the log-related issues. | Manual verification and existing works comparison | N/A | Hadoop and Camel | Files with log-related issues have undergone statistically significantly more frequent prior changes, and bug fixes.  Developers fixing these log-related issues are often not the ones who introduced the logging statement nor the owner of the method containing the logging statement.  Most of the defective logging statements remain unreported for a median 320 days.  Once reported, the issues are fixed, median of five days.  Developed an automated tool that detects four types of log-related issues. |
| (Perez-Castillo & Piattini, 2021) | An empirical multiple case study that analyses 13 open-source projects: the evolution of the development effort influences the code quality. | Spearman correlation test | K-means clustering algorithm | GitHub and SonarCloud | More developers or higher number of commits does not necessary influence worse quality levels.  After applying a clustering algorithm, it is detected an inverse correlation in some cases where specific efforts were made to improve code quality.  The size of commits and the relative weight of developers in their teams might also affect measures like complexity or cloning. |

Table 2.2.1 Column 1 uniquely references each article analyzed. The articles were chosen in a way that ensured no two articles had the same author with similar abstracts and all were not older than 10 years.

Table 2.2.1 Column 2 shows the various types of approaches and is visualized using a bar graph in Figure 2.2.1. It was observed that the most common approach used in 10 out of 27 articles was proposing a new method, like in Wang et al. (2024). Developing new methods often drives innovation by addressing limitations of existing techniques and paving the way for improved applications. The second most common approach used in 5 out of 27 articles was an empirical study, like in Kim & Lee (2021). Empirical studies are valuable for providing real-world evidence and validating theoretical concepts, making them essential for establishing practical applicability. The third most common approach used in 4 out of 27 articles was mining, like in La Cholter et al. (2021). Data mining enables researchers to identify hidden patterns and trends within complex datasets, offering critical insights that can enhance decision-making and model development.

Figure 2.2.1. Types of approaches used in column 2 within Table 2.2.1.

Table 2.2.1 Column 3 shows that 10 out of 27 articles either used existing works as a comparison, like in Wrobel et al. (2023), or cross-validation, like in Li et al. (2022), to evaluate their approach. Using existing works as a benchmark provides a clear context for assessing the effectiveness and novelty of a proposed method, ensuring that new contributions are meaningful. Similarly, cross-validation is crucial for evaluating the robustness and generalizability of results which enhances the credibility of findings (Wikipedia Foundation, 2016). Together, these evaluation methods enhance the reliability of research outcomes.

Table 2.2.1 Column 4 shows that 14 out of 27 articles used a dataset or algorithm in their approach, like in (Canaparo & Ronchieri, 2018; Perez-Castillo & Piattini, 2021). The use of datasets allows researchers to validate their methods against real-world or simulated scenarios, ensuring their applicability to practical problems (Bright Data, 2023). Algorithms, on the other hand, are integral to implementing and testing novel methodologies, acting as the backbone of computational research (Deshila Technology Research Institute, 2024). These resources not only establish a foundation for replicability, but also foster innovation through standardized tools and frameworks.

Table 2.2.1 Column 5 shows that 13 out of 27 articles used GitHub within their approach, like in Soltani et al. (2020). GitHub's popular collaborative platform facilitates the sharing of code, datasets, and research artifacts, promoting transparency and enabling peer validation (GitHub, 2023c). By leveraging GitHub, researchers can improve the accessibility of their work and encourage community engagement.

Table 2.2.1 Column 6 shows that all articles have unique results and conclusions. The diversity of results reflects the richness of approaches and perspectives. Each conclusion contributes to a more comprehensive understanding, offering insights that can guide future investigations. This uniqueness underscores the importance of diverse methodologies in advancing knowledge and addressing complex challenges.

Assessing project quality within OSS projects can involve multiple metrics across various dimensions. Common metrics used within these articles were code quality (such as the number of bugs or the number of commits focused on improving code) as seen in Perez-Castillo & Piattini (2021), community health (such as the number of and the variety of contributors, the average time taken to respond to issues, the average time taken to review and merge pull-requests, or the number and quality of responses to community queries) as seen in Miyamoto et al. (2023), and project activity (such as the number of commits indicating ongoing development, the ratio of closed issues to total issues, the percentage of inactive unresolved issues or pull-requests, or the frequency/consistency of releases or updates) as seen in Ait et al. (2022).

## Analysis of GitHub Data Events

This section analyzes GitHub data events and properties to understand how they correspond to developer actions. The systematic review of 27 articles led to selecting GitHub as the primary OSS repository for this research, by reason of its popularity and accessibility.

Miyamoto et al. (2023) state software repositories facilitate collaboration by managing version control, issue tracking, and dependency management. Focusing on GitHub, the authors mention this software repository allows users to clone source code for local use, submit requests to merge changes or improvements, suggest enhancements or report issues, and fork repositories and publish them as their own open-source projects. The authors focused on issue data in their research. Issues allow users to discuss problems and contribute ideas, guiding development. Visualizing change history helps users understand code changes, evaluate developers, and promote learning. The properties that they found useful within GitHub issues were title, created\_at and closed\_at timestamps, number (number of issue), locked (whether it is possible to post), comments (number of comments), user\_id (user that created the issue), user\_type (type of user), and user\_site\_admin (user is admin or not).

Alshara et al. (2023) state GitHub, a Git-based hosting platform, offers version control, access control, bug tracking, task management, and continuous integration. Focusing on GitHub Flow, the authors mention this distributed branch-based workflow designed to work with Git and GitHub allows teams to manage development and deployments efficiently by using isolated branches for changes, which are later merged into the main deployable branch. The authors focused on events like repositories, issues, pull requests, comments, labels, and commits in their research. Repositories store project files and revision history, each with a unique name. Issues track tasks, bugs, or feedback and are identified by properties such as a unique number, URL, title, body, created\_at and closed\_at timestamps, comments, and labels. Comments facilitate discussions on issues and solutions. Labels categorize issues, popular labels being “bug”, “enhancement”, “duplicate”, “documentation”, and “wontfix”. Pull requests group changes for review before merging and are identified with the same properties as issues, but with the addition of code commits. Commits contain URLs for file changes, version differences, and brief descriptions of updates. The authors created a GitHub scraper tool that automatically gathers all explicit pull request-issue links and introduced a metamodel for these links, providing detailed information about the associated issues and pull requests.

GitHub (2023a) states the GitHub Events API provides various event types triggered by activity, each with shared properties and a unique payload based on the event type. It is focused on tracking and retrieving activity events and primarily used for monitoring activities and providing notifications or feeds about recent changes. The API's consistent event object structure supports building scripts and applications for automation, integration, or extending GitHub features. Examples include triaging issues, creating analytics dashboards, or managing releases. Issue events triggered by activity in issues and pull requests are accessible via the REST API (GitHub, 2023b). This API provides extensive functionality for managing and interacting with the full range of GitHub resources. It enables create, read, update, and delete operations on resources within GitHub, allowing for more comprehensive control over GitHub projects and user interactions. Pull requests are treated as issues in GitHub's REST API, but not all issues are pull requests. Both types share a sequential numbering system within a repository, with pull requests including a “pull\_request” property in the issue object. All issue events follow a common object structure, except those exclusive to timeline events, with some events including additional resource-specific properties. Event documentation details any variations in object format or properties.

The type and description of the important common shared properties within the GitHub Events API for event objects are presented in Table 2.3.1. It consists of 3 columns which are the following: Property (Column 1), Type (Column 2), and Description (Column 3). The type and description of the properties within the GitHub Events API for the issues event unique payload are presented in Table 2.3.2. It consists of 3 columns which are the following: Property (Column 1), Type (Column 2), and Description (Column 3).

Table 2.3.1. Important common shared properties within all GitHub event objects inside of the GitHub Events API.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| id | integer | Unique identifier for the event. |
| type | string | Type of event. |
| actor | object | User that triggered the event. |
| repo | object | Repository object where the event occurred. |
| payload | object | Unique payload object per event type. |
| public | boolean | Whether the event is visible to all users. |
| created\_at | string | The date and time when the event was triggered. |
| org | object | Organization that was chosen by the actor to perform action that triggers the event. |

**Table 2.3.2.** Issues event payload object properties inside of the GitHub Events API.

| **Property** | **Type** | **Description** |
| --- | --- | --- |
| action | string | Action performed. Can be “opened”, “edited”, “closed”, “reopened”, “assigned”, “unassigned”, “labeled”, or “unlabeled”. |
| issue | object | The issue. |
| changes | object | The changes to the issue if the action was “edited”. |
| changes[title][from] | string | Previous version of the title if the action was “edited”. |
| changes[body][from] | string | Previous version of the body if the action was “edited”. |
| assignee | object | Optional user who was assigned or unassigned the issue. |
| label | object | Optional label that was added or removed from the issue. |

To ensure meaningful and well-rounded insights for this research, three to five GitHub repositories should be selected for analysis of data events. This range strikes a balance between depth and manageability, allowing for comparative evaluation while keeping data collection and processing practical. Including multiple repositories helps capture variations in development practices, activity levels, and collaboration patterns, contributing to a more comprehensive and credible study outcome.

## Process Mining

Process mining is a field that combines data mining and process analytics to apply algorithms or techniques to event log data, uncovering trends, patterns, and process execution details which enables organizations to understand and gain insights to the performance of their processes (IBM, 2021). Event logs can contain data such as timestamps, event types, unique identifiers, statuses, actions, and attributes. There are 3 basic types of process mining: discovery, which generates a process model solely from event log data without prior models and is the most widely used; conformance, which verifies if actual processes align with intended models by identifying deviations; and enhancement, which uses additional data, such as conformance outputs, to improve existing models by addressing issues like bottlenecks (IBM, 2021). Figure 2.4.1 shows discovery, conformance checking, and enhancement in terms of input and output (Aalst et al., 2012).

A diagram of a process

Description automatically generated

Figure 2.4.1. 3 basic types of process mining in terms of input and output(Aalst et al., 2012).

According to IBM (2021), the advantages of process mining are enhanced transparency, simplified process analysis and enhanced efficiency, data-driven decision making, process optimization, and process standardization. Process mining offers enhanced transparency by providing a data-driven view of operational processes, allowing businesses to identify inefficiencies and compliance issues more effectively than traditional mapping methods. It simplifies process analysis by leveraging event-log data to quickly visualize and streamline operations, reducing cycle times and costs. With data-driven insights, process mining supports objective decision-making, helping to pinpoint and resolve issues like bottlenecks. By continuously monitoring performance metrics, it facilitates ongoing process optimization and automation. Additionally, it promotes process standardization by identifying variations and aligning them with optimal models, ensuring consistent performance and quality.

Process discovery is a type of process mining with intentions of extracting process related knowledge from event logs and using this knowledge to create a process model (Wikipedia Foundation, 2024). The model created from this method is developed without outside influence as no previous process model is referred to (IBM, 2021). Currently, there are over 100 process mining algorithms capable of discovering process models such as genetic process discovery techniques, heuristic mining algorithms, region-based mining algorithms, and fuzzy mining algorithms (Wikipedia Foundation, 2024). The representation and accuracy of the graphical models developed are dependent on the technique and visualization chosen. There are many ways to graphically represent process discovery methods, but 3 process models are commonly used. The directly-follows graph (DFG) is the simplest, showing activities and their direct sequential relationships (Wikipedia Foundation, 2024). Petri nets offer a more complex representation, capturing sequential, parallel, and looping behaviors through token flows (Wikipedia Foundation, 2024). The business process model and notation standard (BPMN) is a commonly used flat control-flow perspective process model that can capture subprocesses, data flows, and resources in one diagram, which makes it an appealing tool for process miners and business users (Wikipedia Foundation, 2024). Using a process discovery technique on GitHub projects will provide a better oversight of project quality.

Figure 2.4.2 shows 3 common graphical representations of process discovery: a petri net model (a), a business process model and notation model (b), and a directly-follows graph (c). All 3 of these models are representing a purchase-to-pay process showcasing only the frequent desired outcomes, with the petri net and BPMN models specifying the same behavior while the DFG specifies paths not allowed in the petri net and BPMN models (Van Der Aalst, 2019).

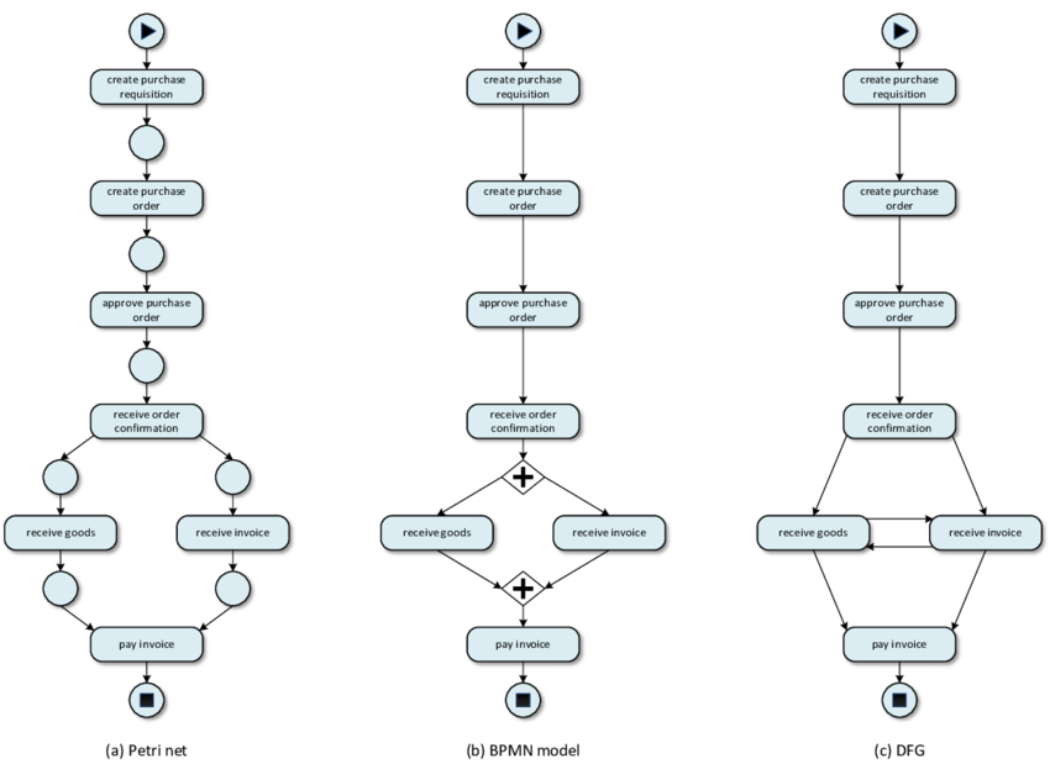


Figure 2.4.2. Process models discovered for a purchase-to-pay process showcasing only the frequent desired outcomes (Van Der Aalst, 2019).

Open-source process mining tools were explored to see which one would be the most beneficial for this research and a summary is presented in Table 2.4.1. Apromore is a user friendly web based tool focused on business users, offering intuitive dashboards and BPMN model support (Apromore, 2025). PM4Py is a Python based library ideal for developers and automation, enabling scripting of process mining tasks without a graphical user interface (GUI) (Process Intelligence Solutions, 2025). Due to Apromore’s algorithm variety and PM4Py’s programming skills requirement, ProM was selected. ProM is a free, Java-based, platform-independent framework often referred to as ProM Tools that supports a wide range of discovery, conformance, and enhancement techniques through an extensible plugin system (ProM Tools, 2025). It supports the eXtensible Event Stream (XES) event log format which ensures compatibility with standard process mining workflows. Even though it has a steep learning curve, its flexibility and depth make it ideal for academic research. Advanced algorithms such as Alpha, Heuristics, Inductive, and Fuzzy Miner are available for use which offer detailed deep insights (ProM Tools, 2025).

**Table 2.4.1.** Open-Source Process Mining Tools Comparison.

|  |  |  |  |
| --- | --- | --- | --- |
| **Tool** | **Key Features** | **Strengths** | **Limitations** |
| ProM | Plugin-based, supports many algorithms, XES format | Highly extensible / deep analysis, research focused, algorithm rich | Complex UI, steep learning curve |
| Apromore | BPMN support, dashboarding, event log analysis | User friendly, collaborative, good business alignment | Limited algorithm variety |
| PM4Py | Scriptable, supports discovery & conformance via Python code | Lightweight, reproducible, great for automation and custom pipelines | No GUI, requires programming skills |

Discovery process mining algorithms that could be useful to repository reliability analysis were explored further to grasp a better understanding and a summary is presented in Table 2.4.2. The Alpha Miner is the first process discovery algorithm, designed to identify causal relationships between activities in event logs and convert them into workflow nets (Wikipedia Foundation, 2025). While foundational and easy to understand, it works best with clean, structured data and has influenced many later algorithms. The Heuristics Miner builds on Alpha Miner by filtering out noise and infrequent behaviors, making it more suitable for real-life, complex event logs (Turdibayeva, 2024). It generates robust models using causal nets that emphasize frequent and relevant activity paths. The Inductive Miner discovers process models by recursively splitting event logs into smaller parts, ensuring the resulting models are structured and sound (Turdibayeva, 2024). It handles loops, parallelism, and complex behavior well, making it ideal for scalable and flexible analysis. The Fuzzy Miner focuses on simplifying complex or unstructured processes by dynamically adjusting model detail based on significance and correlation metrics (Turdibayeva, 2024). It provides interactive, user driven views that highlight the most relevant parts of the process depending on the level of detail needed.

**Table 2.4.2.** Process Mining Algorithms Summary.

| **Algorithm** | **Short Description** | **Diagram Output Type** | **Handles Noise?** | **Good for GitHub Logs?** |
| --- | --- | --- | --- | --- |
| Alpha Miner | Finds activity patterns; best for simple and clean logs | Petri Net | Low Noise Only | Simple Logs Only |
| Heuristics Miner | Incorporates frequency-based heuristics, more robust to noise, can handle real-life event logs | Heuristics Net | Yes | Yes |
| Inductive Miner | Builds clear, block-structured models; suitable for complex logs | Process Tree / Petri Net | Yes | Yes |
| Fuzzy Miner | Simplifies models by abstracting less significant behavior; good for exploratory analysis | Fuzzy Model (Simplified Graph) | Yes | Yes |

After reviewing these discovery process mining algorithms, it was found that Heuristics Miner and Inductive Miner algorithms will provide the most beneficial output for this research.

## Statistical Analysis Methods

Statistical analysis methods including survival analysis, prescriptive analysis, resource analysis, and predictive analysis were explored to see which one would be the most beneficial for this research. Survival analysis focuses on time-to-event data, measuring how long it takes for a specific event to occur, such as system failure (Dovetail Editorial Team, 2024). It is useful for identifying patterns over time and evaluating the impact of certain factors on event duration. Prescriptive analysis recommends specific actions by combining data, algorithms, and business rules to guide decision-making (Dovetail Editorial Team, 2024). It answers “What should we do next?” using optimization techniques to suggest the best possible outcomes. Resource analysis evaluates the availability and usage of people, tools, or time to support planning and decision-making in projects (Vyas, 2025). It helps estimate timelines, manage workloads, and optimize resource allocation. While survival analysis, prescriptive analysis, and resource analysis offer valuable insights, they were not selected due to their complexity, specific domain focus, or limited relevance to pull request event data, whereas predictive analysis was chosen for its simplicity, practicality, and alignment with forecasting patterns in GitHub repository activity.

Predictive analysis uses historical data and statistical models to forecast future trends or behaviors, such as demand or risk (Dovetail Editorial Team, 2024). While not exact, it enables informed planning by highlighting likely outcomes based on past patterns. Predictive analysis was chosen as the most suitable method because event data from GitHub repositories, such as pull request data, is inherently time-stamped and sequential, making it ideal for statistical modelling and forecasting. Metrics such as time to merge, number of comments or reviews, and approval or rejection patterns can be extracted, analyzed, and used to predict future repository behavior. This allows for identifying potential process bottlenecks and trends and forecasting future repository activity, making it both practical and effective to implement with available data and tools.

Various statistical and machine learning models to forecast future outcomes based on past data can be used within predictive analysis. Common methods include linear regression, logistic regression, and time series analysis. Linear regression helps estimate continuous outcomes like how long a pull request takes to merge (Halton, 2025). Logistic regression is used to predict binary results such as whether a pull request will be approved or rejected (Halton, 2025). Time series analysis is useful for identifying trends over time, such as predicting how many pull requests will be opened next week (insightsoftware, 2023). More advanced models like decision trees, random forests, and support vector machines can identify patterns in more complex datasets, such as flagging pull requests likely to be delayed (insightsoftware, 2023). The choice of model depends on the type of data available and the specific prediction goals. These methods can be applied directly to GitHub pull request data like number of comments, review time, file changes, and contributor activity. A summary of these findings is presented in Table 2.5.1 below.

**Table 2.5.1.** Predictive Analysis Models/Methods Summary.

| **Model/Method** | **Purpose** | **Example GitHub Use** | **Output Type** |
| --- | --- | --- | --- |
| Linear Regression | Predicts a numeric value | Estimate time to merge a pull request | Numeric (example: number of days) |
| Logistic Regression | Predicts a yes/no or true/false outcome | Predict if a pull request will be approved | Binary (example: yes or no) |
| Time Series Analysis | Forecasts trends based on time data | Predict number of pull requests expected next month | Numeric (example: time series) |
| Decision Tree | Creates rules based on data splits | Classify pull requests as high or low risk for delays | Categories / Classes |
| Random Forest | Combines multiple trees for better accuracy | Predict probability of pull request rejection | Probability / Class |
| Support Vector Machine | Separates complex data into categories | Identify pull requests likely to need multiple reviews | Class (example: review needed) |

After reviewing these predictive analysis models and methods, it was found that logistic regression will provide the most beneficial output for this research due to its ability to predict binary outcomes from structured GitHub pull request data. Additional usage other than predicting pull request acceptance for this research could be forecasting merge delay by predicting whether a pull request will be merged within a threshold time, classifying pull requests with issues by using features like comment sentiment, file types changed, or change size to predict if a pull request will be associated with future bug reports or reopened issues, and detecting reviewer engagement by predicting the likelihood of reviewer response within a given time frame based on pull request metadata and historical team activity. Python, specifically the scikit-learn library, will be used to apply logistic regression as it allows full control, customization, and easy evaluation. Afterwards the results would then be interpreted.

## Main Results of the 2nd Section

The analysis of 27 articles reveals diverse methodologies for understanding open-source software (OSS) development, reliability, and quality. Studies like (Miyamoto et al., 2023) demonstrated the importance of OSS reliability by showing how software repair time depends on fault severity and developer capability, while (Kim & Lee, 2021) highlighted the efficiency of multi-labels in managing software issues. Empirical studies were prominent, with findings from (Ait et al., 2022) showing the lifecycle patterns of OSS projects, where many die within a year, and less than 50% survive beyond five years. Similarly, (Eiroa-Lledo et al., 2023) used survival analysis to identify factors affecting bug resolution times, noting that issues resolved by their original authors were addressed more quickly. Additionally, (Soltani et al., 2020) emphasized that high-quality bug reports significantly reduced resolution times, yet over 70% of such reports lacked essential details.

The studies also highlighted methodological advances, such as the use of advanced algorithms like RReLU and Radam in OSS reliability assessments (Miyamoto et al., 2023) or clustering techniques for evaluating code quality (Perez-Castillo & Piattini, 2021). (Tamura & Yamada, 2023) showcased cost-effective reliability evaluation using transfer learning. Several studies emphasized leveraging GitHub for practical insights such as (Romanov et al., 2024) that demonstrated the effectiveness of fuzzy logic models for predicting repository dynamics. Furthermore, systematic reviews provided critical insights into gaps in the literature; for example, (Macak et al., 2022) identified research gaps in process mining applications for software reliability, while (Cosentino et al., 2017) proposed actions to address methodological limitations, such as small datasets and poor sampling. Collectively, these findings highlight the importance of tailored approaches, robust evaluations, and community engagement for advancing OSS research.

GitHub data events and properties were examined to understand their connection to developer actions. GitHub is highlighted as the primary OSS repository in our research due to its popularity and features supporting collaboration, such as version control, issue tracking, and dependency management (Miyamoto et al., 2023). Key focus areas include issues, which facilitate problem-solving and idea-sharing, with useful properties like timestamps, comments, and user details. Alshara et al. (2023) emphasize GitHub Flow, a branch-based workflow enhancing development efficiency, and explore events such as repositories, issues, pull requests, comments, labels, and commits. GitHub’s Events API (GitHub, 2023b, 2023a) provides a structured framework for automation and integration, enabling tasks like triaging issues or managing releases. Shared and unique event properties are categorized and detailed in Table 2.3.1 and Table 2.3.2, showcasing properties, types, and descriptions essential for analyzing GitHub events.

Process mining combines data mining and process analytics for the analysis of event logs to uncover patterns and improve process performance through 3 basic types of process mining (IBM, 2021). The 3 basic types of process mining: discovery, conformance, and enhancement, are depicted in Figure 2.4.1 in terms of input and output (Aalst et al., 2012). Process mining offers transparency, simplifies analysis, boosts efficiency, supports data-driven decisions, optimizes processes, and promotes standardization by identifying inefficiencies, bottlenecks, and variations for consistent quality (IBM, 2021). Process discovery, a process mining technique, extracts knowledge from event logs to create graphical models of processes, with over 100 algorithms available for model generation (Wikipedia Foundation, 2024). The accuracy of these models depends on the chosen technique and visualization, with DFGs, petri nets, and BPMNs being commonly used (Wikipedia Foundation, 2024) and depicted in Figure 2.4.2.

Through comparative evaluation of open-source process mining tools, ProM was selected for its wide algorithm support, extensibility, and suitability for in-depth academic research despite its complexity (ProM Tools, 2025). Four key discovery algorithms such as Alpha, Heuristics, Inductive, and Fuzzy Miner were reviewed for their strengths in handling event logs, with Inductive Miner and Heuristics Miner noted as especially effective for GitHub event log data due to their noise-handling capabilities and clear model output (Turdibayeva, 2024). Additionally, multiple statistical analysis methods were explored, and predictive analysis was identified as the most suitable for this research due to its ability to model time stamped pull request data and forecast repository behavior using accessible metrics such as merge time, reviews, and approval patterns (Dovetail Editorial Team, 2024). Predictive analysis methods and models were further compared for their suitability with this data, and logistic regression was selected as the most effective due to its simplicity, interpretability, and strong performance with binary classification tasks (Halton, 2025).

# Proposed Approach

This section provides a description of the proposed method or approach using a BPMN diagram. The GitHub Repository Reliability Analysis process starts with the Repository Data Collection & Pre-Processing sub process that outputs extracted data formatted as an eXtensible Event Stream (XES) file which is then input into the Repository Process Evaluation sub process that outputs Results which get transformed into a Report. This flow can be seen below in Figure 3.1.

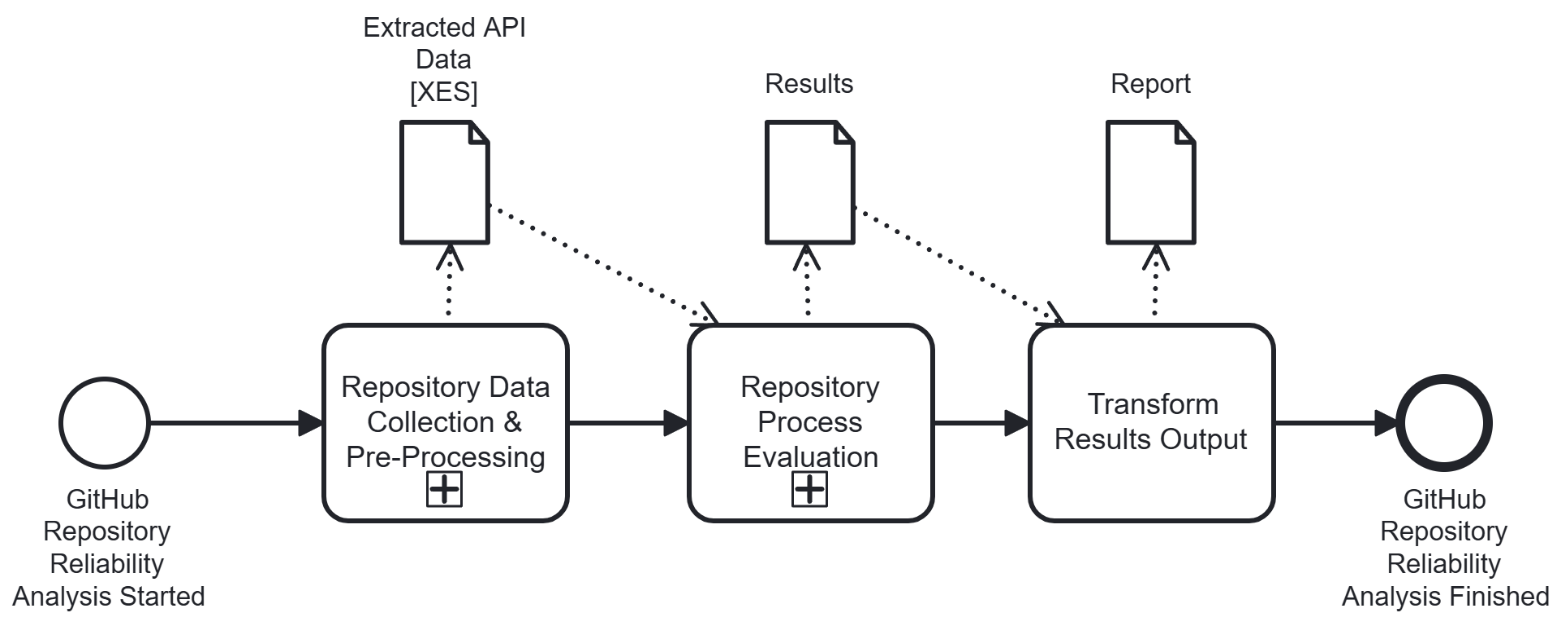


Figure 3.1. GitHub Repository Reliability Analysis Process.

Within the Repository Data Collection & Pre-Processing sub process, a repository is manually selected to have its data extracted in the form of a Comma-Separated Values (CSV) file via the Data Extraction sub process and that extracted data is transformed into a XES file to be used for the Repository Process Evaluation sub process. This flow can be seen below in Figure 3.2.

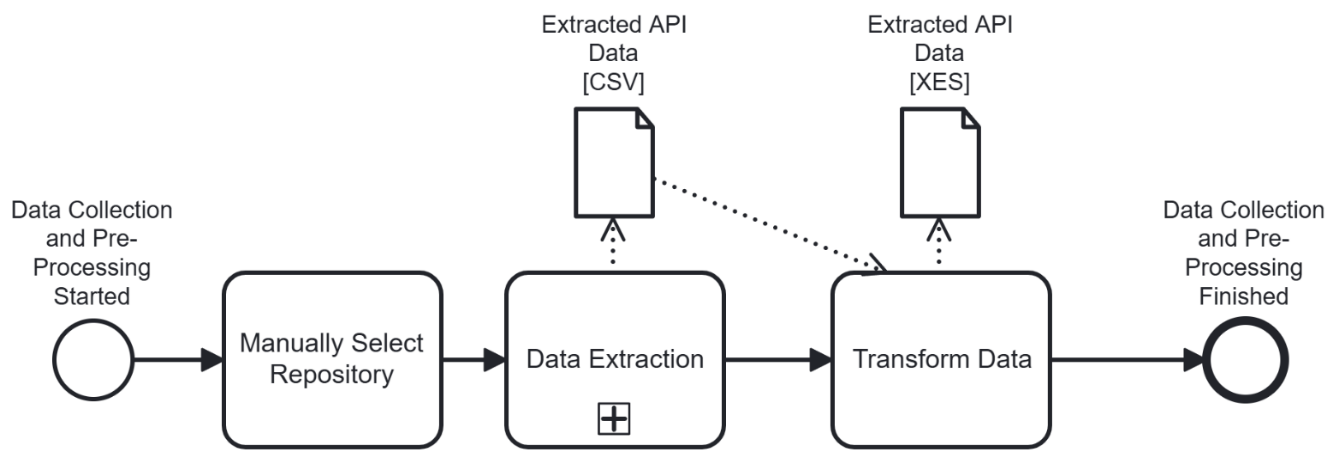


Figure 3.2. Repository Data Collection and Pre-Processing Sub Process.

Within the Data Extraction sub process, data events such as pull requests, pull requests issues comments, pull requests reviews, pull requests review comments, and branch deletion events are pulled from the repository for analysis and merged in a CSV file format. This flow can be seen below in Figure 3.3.

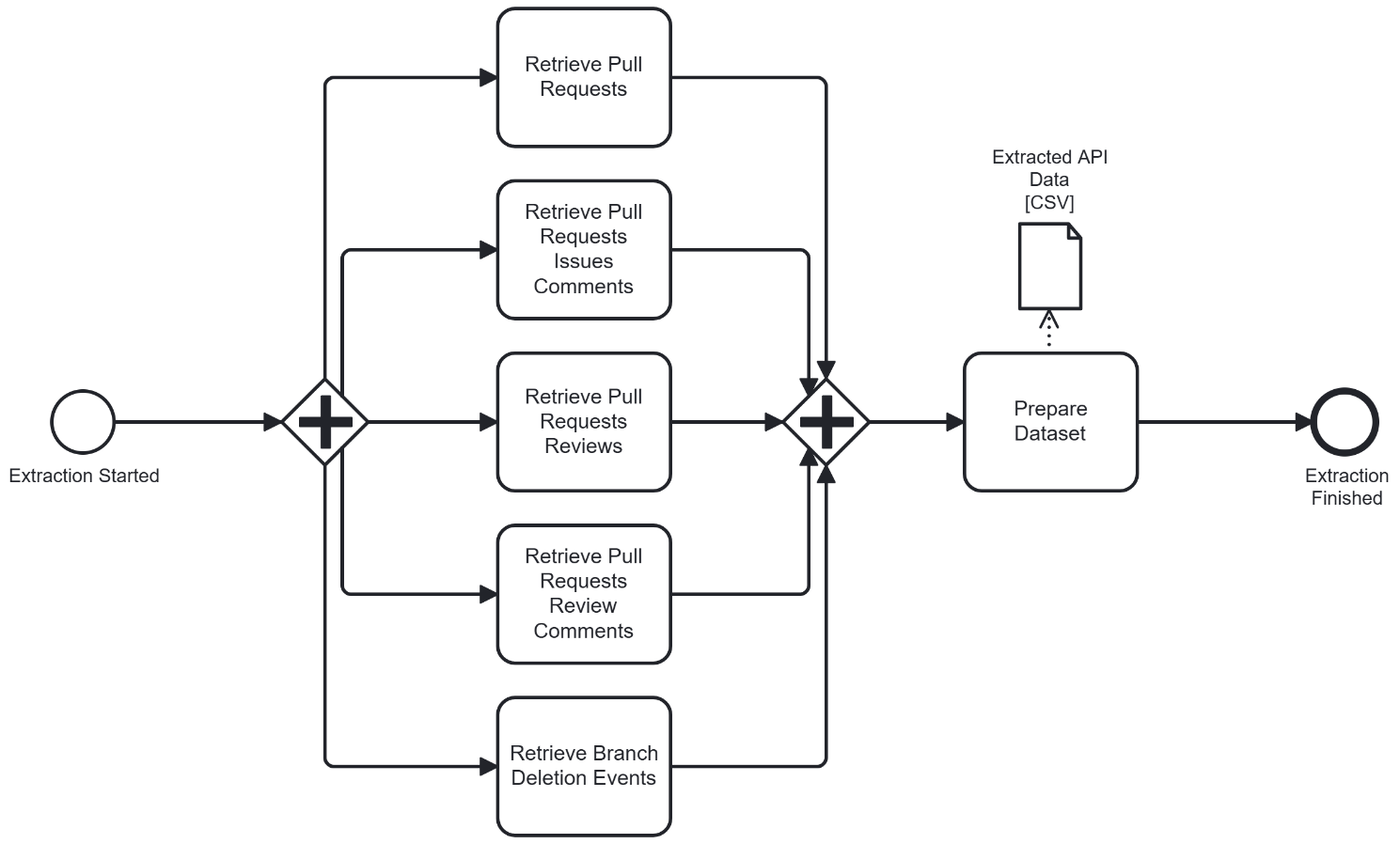


Figure 3.3. Data Extraction Sub Process.

Within the Repository Process Evaluation sub process, the extracted data transformed into an XES file format is then evaluated by applying statistical analysis to calculate metrics and applying process mining methods to generate process models. These outputs are then merged and once the Results are satisfiable, the process is over. If the Results are not satisfiable, statistical analysis and process mining methods are applied again to the extracted data until the Results are satisfiable. This flow can be seen below in Figure 3.4.

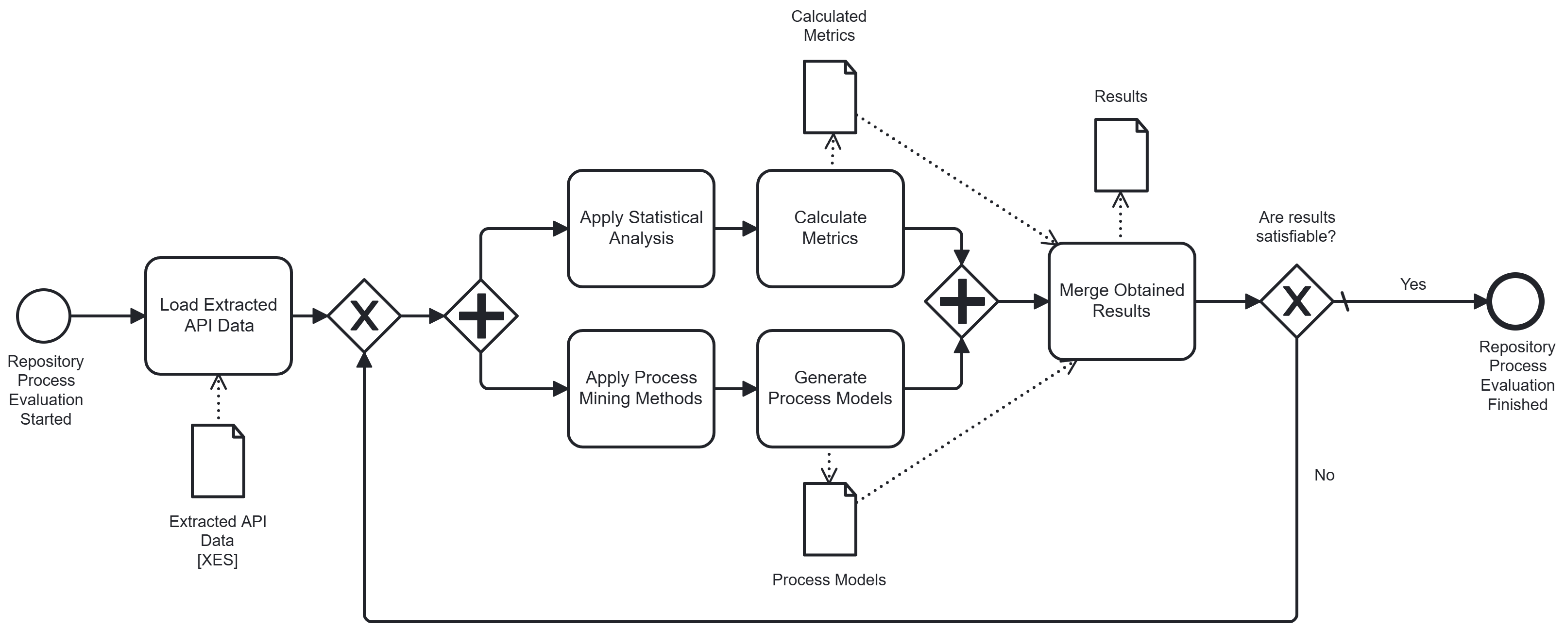


Figure 3.4. Repository Process Evaluation Sub Process.

Overall, the BPMN diagram gives a clear and organized view of the proposed method. It shows the main steps for handling the data, how the evaluation is repeated if needed, and where key decisions are made. This structured approach helps make sure the method is consistent, easy to follow, and can be repeated for analyzing the reliability of any repository.

## Main Results of the 3rd Section

The GitHub Repository Reliability Analysis process was modelled using BPMN to show the key steps involved in collecting, preparing, and evaluating repository data. The process is divided into two main sub processes: Repository Data Collection & Pre-Processing and Repository Process Evaluation.

In the Repository Data Collection & Pre-Processing stage, a repository is manually selected, and relevant data is extracted. This data is combined into a single CSV file and then converted into an XES file, which is suitable for process mining analysis. The Repository Process Evaluation stage uses the XES file to calculate reliability metrics through statistical analysis and to create process models using process mining methods. These results are reviewed, and if they are not acceptable, the evaluation is repeated until the outcomes meet the expected criteria. Once the results are satisfactory, they are used to create a final report.

The BPMN diagrams clearly outline each part of the process, showing how data is transformed and evaluated. This structured approach helps ensure that the method is easy to follow, repeatable, and useful for assessing the reliability of repositories. What follows from the results is a strong foundation for integrating process mining techniques into software repository quality evaluation.

# Conclusions

In this section, the general conclusions are presented as follows:

1. Based on the performed analysis and comparison of methods used to identify software project repository reliability issues from the perspective of expanding on good project development practices, it was found that top methods were data or process mining, empirical studies, deep machine learning, and systematic mapping studies, and that methods like predictive models, data mining, and process mining for defect detection and reliability assessment have potential but require further development to improve accuracy and accessibility.
2. Based on the performed modelling of the GitHub Repository Reliability Analysis process using BPMN, it was found that a clearly defined and repeatable workflow for data extraction, transformation, and evaluation shows that the proposed method effectively supports structured and consistent assessment of repository quality, and that the benefits of these results include improved clarity in the analysis process and easier identification of key data points.

# References

Aalst, W. M. P., Adriansyah, A., Karla, A., Medeiros, A. de, Arcieri, F., Baier, T., Blickle, T., Bose, R., Brand, P. Van Den, Brandtjen, R., Guzzo, A., Harmon, P., Hofstede, A., Hoogland, J., Ingvaldsen, J. E., Kato, K., Malerba, D., Mans, R., Manuel, A., … Song, M. (2012). Process Mining Manifesto. *Lecture Notes in Business Information Processing*, *99*(2), 169–194. www.win.tue.nl/ieeetfpm/

Ait, A., Izquierdo, J. L. C., & Cabot, J. (2022). An Empirical Study on the Survival Rate of GitHub Projects. *Proceedings - 2022 Mining Software Repositories Conference, MSR 2022*, 365–375. https://doi.org/10.1145/3524842.3527941

Alshara, Z., Shatnawi, A., Eyal-Salman, H., Seriai, A. D., & Shatnawi, M. (2023). PI-Link: A Ground-Truth Dataset of Links Between Pull-Requests and Issues in GitHub. *IEEE Access*, *11*(January), 697–710. https://doi.org/10.1109/ACCESS.2022.3232982

Apromore. (2025). *Unlock the True Potential of Your Transformation*. https://apromore.com/

Bright Data. (2023). *What Is a Dataset? Definition, Use Cases, Benefits, and Example*. https://medium.com/@Bright-Data/what-is-a-dataset-definition-use-cases-benefits-and-example-9aaf5ecc301e#:~:text=Improved Decision-Making,the data%2C and measure performance.

Brittain, J., & Darwin, I. F. (2003). *Open Source*. IEEE Micro. https://doi.org/10.1109/MM.2003.1240200

Bunge, J. A., & Judson, D. H. (2004). *Data Mining*. Encyclopedia of Social Measurement, Three-Volume Set. https://doi.org/10.1016/B0-12-369398-5/00159-6

Canaparo, M., & Ronchieri, E. (2018). Data mining techniques for software quality prediction. *Software Design and Development: Concepts, Methodologies, Tools, and Applications*, *1–4*, 1–9. https://doi.org/10.1051/epjconf/201921405007

Cosentino, V., Izquierdo, J. L. C., & Cabot, J. (2017). A Systematic Mapping Study of Software Development with GitHub. *IEEE Access*, *5*, 7173–7192. https://doi.org/10.1109/ACCESS.2017.2682323

Deshila Technology Research Institute. (2024). *Algorithms: The importance, advantages & tips to writing them*. https://www.linkedin.com/pulse/algorithms-importance-advantages-tips-kqzpc/

Dovetail Editorial Team. (2024). *From ANOVA to regression: 10 key statistical analysis methods explained*. https://dovetail.com/research/key-statistical-analysis-methods-explained/

Eiroa-Lledo, E., Ali, R. H., Pinto, G., Anderson, J., & Linstead, E. (2023). Large-Scale Identification and Analysis of Factors Impacting Simple Bug Resolution Times in Open Source Software Repositories. *Applied Sciences (Switzerland)*, *13*(5). https://doi.org/10.3390/app13053150

Garcia, C. dos S., Meincheim, A., Faria Junior, E. R., Dallagassa, M. R., Sato, D. M. V., Carvalho, D. R., Santos, E. A. P., & Scalabrin, E. E. (2019). Process mining techniques and applications – A systematic mapping study. *Expert Systems with Applications*, *133*, 260–295. https://doi.org/10.1016/j.eswa.2019.05.003

GitHub. (2023a). *GitHub event types*. Github.Com. https://docs.github.com/en/webhooks-and-events/events/github-event-types#pullrequestreviewcommentevent

GitHub. (2023b). *Issue event types*. Github.Com. https://docs.github.com/en/rest/using-the-rest-api/issue-event-types?apiVersion=2022-11-28

GitHub. (2023c). *The tools you need to build what you want.* Github.Com. https://github.com/features

Given, L. (2012). *Empirical Research*. The SAGE Encyclopedia of Qualitative Research Methods. https://doi.org/10.4135/9781412963909.n133

Haider, S., Khalil, W., Al-Shamayleh, A. S., Akhunzada, A., & Gani, A. (2023). Risk Factors and Practices for the Development of Open Source Software from Developers’ Perspective. *IEEE Access*, *11*(June), 63333–63350. https://doi.org/10.1109/ACCESS.2023.3267048

Halton, C. (2025). *Predictive Analytics: Definition, Model Types, and Uses*. Investopedia. https://www.investopedia.com/terms/p/predictive-analytics.asp#:~:text=The most common predictive models,deep learning methods and technologies.

Hassani, M. (2018). *Studying and Detecting Log-related Issues Concordia University*. *March*.

IBM. (2021). *What is process mining?* IBM. https://www.ibm.com/think/topics/process-mining

insightsoftware. (2023). *Top 5 Predictive Analytics Models and Algorithms*. https://insightsoftware.com/blog/top-5-predictive-analytics-models-and-algorithms/#:~:text=next ten days.-,What are Predictive Analytics Models,strategic planning and operational efficiency.

Kim, J., & Lee, S. (2021). An empirical study on using multi-labels for issues in github. *IEEE Access*, *9*, 134984–134997. https://doi.org/10.1109/ACCESS.2021.3116061

La Cholter, W., Elder, M., & Stalick, A. (2021). Windows Malware Binaries in C/C++ GitHub Repositories:Prevalence and Lessons Learned. *International Conference on Information Systems Security and Privacy*, *Icissp*, 475–484. https://doi.org/10.5220/0010237904750484

Li, H., Zhang, H., Wang, S., & Hassan, A. E. (2022). Studying the Practices of Logging Exception Stack Traces in Open-Source Software Projects. *IEEE Transactions on Software Engineering*, *48*(12), 4907–4924. https://doi.org/10.1109/TSE.2021.3129688

Linåker, J., Papatheocharous, E., & Olsson, T. (2022). How to characterize the health of an Open Source Software project? A snowball literature review of an emerging practice. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3555051.3555067

Macak, M., Daubner, L., Fani Sani, M., & Buhnova, B. (2022). Process mining usage in cybersecurity and software reliability analysis: A systematic literature review. *Array*, *13*(November 2021), 100120. https://doi.org/10.1016/j.array.2021.100120

Mahmood, N., Hafeez, Y., Iqbal, K., Hussain, S., Aqib, M., Jamal, M., & Song, O. Y. (2021). Mining software repository for cleaning bugs using data mining technique. *Computers, Materials and Continua*, *69*(1), 873–893. https://doi.org/10.32604/cmc.2021.016614

Miyamoto, S., Tamura, Y., & Yamada, S. (2023). A Method of OSS Reliability Assessment Based on Public Repository Analysis. *International Journal of Reliability, Quality and Safety Engineering*, *30*(5), 1–12. https://doi.org/10.1142/S0218539323500213

Nassar, S. M. (1986). *Software reliability*. Computers and Industrial Engineering. https://doi.org/10.1016/0360-8352(86)90164-6

Nizam, A. (2022). Software Project Failure Process Definition. *IEEE Access*, *10*, 34428–34441. https://doi.org/10.1109/ACCESS.2022.3162878

PER RUNESON, M. H., OST, AUSTEN RAINER, B. ¨, REGNELL, O., & Runeson, P. (2012). *in Software in Software*. http://www.worldcat.org/title/case-study-research-in-software-engineering-guidelines-and-examples/oclc/828789615&referer=brief\_results

Perez-Castillo, R., & Piattini, M. (2021). Understanding the Impact of Development Efforts in Code Quality. In *JUCS - Journal of Universal Computer Science* (Vol. 27, Issue 10). https://doi.org/10.3897/jucs.72475

Process Intelligence Solutions. (2025). *Process Mining Unlocked*. https://processintelligence.solutions/pm4py

ProM Tools. (2025). *ProM Tools*. https://promtools.org/

Puangjaktha, P., Choetkiertikul, M., & Tuarob, S. (2024). “Paper, Meet Code”: A Deep Learning Approach to Linking Scholarly Articles With GitHub Repositories. *IEEE Access*, *12*(May), 68410–68426. https://doi.org/10.1109/ACCESS.2024.3399767

Reszka, Ł., Sosnowski, J., & Dobrzyński, B. (2023). Enhancing Software Project Monitoring with Multidimensional Data Repository Mining. *Electronics (Switzerland)*, *12*(18). https://doi.org/10.3390/electronics12183774

Rodriguez, A. R. (2003). *Literature review*. American Journal of Medical Quality. https://doi.org/10.1177/106286060301800507

Romanov, A., Yarushkina, N., Filippov, A., Sergeev, P., Andreev, I., & Kiselev, S. (2024). Time Series Forecasting during Software Project State Analysis. *Mathematics*, *12*(1). https://doi.org/10.3390/math12010047

Salama, M., Bahsoon, R., & Bencomo, N. (2017). Managing Trade-offs in Self-Adaptive Software Architectures. *Managing Trade-Offs in Adaptable Software Architectures*, 249–297. https://doi.org/10.1016/B978-0-12-802855-1.00011-3

Saroar, S. G., & Nayebi, M. (2023). Developers’ Perception of GitHub Actions: A Survey Analysis. *ACM International Conference Proceeding Series*, 121–130. https://doi.org/10.1145/3593434.3593475

Sattler, F., Böhm, S., Schubert, P. D., Siegmund, N., & Apel, S. (2023). SEAL: Integrating Program Analysis and Repository Mining. *ACM Transactions on Software Engineering and Methodology*, *32*(5). https://doi.org/10.1145/3585008

Sayago-Heredia, J., Pérez-Castillo, R., & Piattini, M. (2021). A Systematic Mapping Study on Analysis of Code Repositories. *Informatica (Netherlands)*, *32*(3), 619–660. https://doi.org/10.15388/21-INFOR454

Singh, R., Gehlot, A., Kumar Prajapat, M., & Singh, B. (2021). *Deep Learning*. Artificial Intelligence in Agriculture. https://doi.org/10.1201/9781003245759-5

Soltani, M., Hermans, F., & Bäck, T. (2020). The significance of bug report elements. *Empirical Software Engineering*, *25*(6), 5255–5294. https://doi.org/10.1007/s10664-020-09882-z

Sriya, V., & Srinivas, P. V. S. (2019). *Bug tracking system*. Journal of Advanced Research in Dynamical and Control Systems. https://doi.org/10.55041/ijsrem32737

Takahashi, K., & Takahashi, L. (2024). *Machine Learning*. Materials Informatics and Catalysts Informatics. https://doi.org/10.1007/978-981-97-0217-6\_7

Tamura, Y., & Yamada, S. (2023). Deep Learning Based on Fine Tuning with Application to the Reliability Assessment of Similar Open Source Software. *International Journal of Mathematical, Engineering and Management Sciences*, *8*(4), 632–639. https://doi.org/10.33889/IJMEMS.2023.8.4.036

Turdibayeva, K. (2024). *Process mining algorithms simply explained*. ProcessMaker. https://www.processmaker.com/blog/process-mining-algorithms-simply-explained/

Van Der Aalst, W. M. P. (2019). A practitioner’s guide to process mining: Limitations of the directly-follows graph. *Procedia Computer Science*, *164*(January), 321–328. https://doi.org/10.1016/j.procs.2019.12.189

Vyas, R. (2025). *Step-By-Step Guide to Resource Analysis in Project Management*. https://www.eresourcescheduler.com/blog/step-by-step-guide-to-resource-analysis-in-project-management#:~:text=Resource Analysis is the process,and calculate the associated costs.

Wang, J., Li, P., Hu, J., & Zhang, C. (2024). A multi-release reliability model of open source software with fault detection obeying three-parameter lifetime distribution. *Scientific Reports*, *14*(1), 1–18. https://doi.org/10.1038/s41598-024-70536-x

Wikipedia Foundation. (2016). *Cross-validation (statistics)*. Wikipedia The Free Encyclopedia. https://en.wikipedia.org/wiki/Cross-validation%7B\_%7D(statistics)

Wikipedia Foundation. (2024). *Business process discovery*. Wikipedia The Free Encyclopedia. https://en.wikipedia.org/wiki/Business\_process\_discovery

Wikipedia Foundation. (2025). *Alpha Algorithm*. https://en.wikipedia.org/wiki/Alpha\_algorithm

Wrobel, M. R., Szymukowicz, J., & Weichbroth, P. (2023). Using Continuous Integration Techniques in Open Source Projects - An Exploratory Study. *IEEE Access*, *11*(September), 113848–113863. https://doi.org/10.1109/ACCESS.2023.3324536

Zhou, X., Jin, Y., Zhang, H., Li, S., & Huang, X. (2016). A map of threats to validity of systematic literature reviews in software engineering. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, *0*, 153–160. https://doi.org/10.1109/APSEC.2016.031