Mining Pull Requests to Detect Process Anomalies in Open Source Software Development

Bohan Liu
State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University
Nanjing, Jiangsu, China
bohanliu@nju.edu.cn

He Zhang*
State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University
Nanjing, Jiangsu, China
hezhang@nju.edu.cn

Weigang Ma
State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University
Nanjing, Jiangsu, China
mf21320110@smail.nju.edu.cn

Hongyu Kuang
State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University
Nanjing, Jiangsu, China
khy@nju.edu.cn

Yi Yang
State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University
Nanjing, Jiangsu, China
522022320177@smail.nju.edu.cn

Jinwei Xu
State Key Laboratory of Novel Software Technology, Software Institute, Nanjing University
Nanjing, Jiangsu, China
jinwei_xu@smail.nju.edu.cn

Shan Gao
Huawei Technologies Co., Ltd.
China
gaoshan17@huawei.com

Jian Gao
Huawei Technologies Co., Ltd.
China
gaojian79@huawei.com

ABSTRACT

Trustworthy Open Source Software (OSS) development processes are the basis that secures the long-term trustworthiness of software projects and products. With the aim to investigate the trustworthiness of the Pull Request (PR) process, the common model of collaborative development in OSS community, we exploit process mining to identify and analyze the normal and anomalous patterns of PR processes, and propose our approach to identifying anomalies from both control-flow and semantic aspects, and then to analyze and synthesize the root causes of the identified anomalies. We analyze 17531 PRs of 18 OSS projects on GitHub, extracting 26 root causes of control-flow anomalies and 19 root causes of semantic anomalies. We find that most PRs can hardly contain both semantic anomalies and control-flow anomalies, and the internal custom rules in projects may be the key causes for the identified anomalous PRs. We further discover and analyze the patterns of normal PR processes. We find that PRs in the non-fork model (42%) are far more likely than the fork model (5%) to bypass the review process, indicating a higher potential risk. Besides, we analyzed nine poisoned projects whose PR practices were indeed worse. Given the complex and diverse PR processes in OSS community, the proposed approach can help identify and understand not only anomalous PRs but also normal PRs, which offers early risk indications of suspicious incidents (such as poisoning) to OSS supply chain.

*Corresponding author: He Zhang.

CCS CONCEPTS

• **Software and its engineering** → **Collaboration in software development**; • **Computing methodologies** → *Machine learning*; • **Security and privacy** → *Human and societal aspects of security and privacy*.

KEYWORDS

open source software development, process mining, pull request

ACM Reference Format:
Bohan Liu, He Zhang, Weigang Ma, Hongyu Kuang, Yi Yang, Jinwei Xu, Shan Gao, and Jian Gao. 2024. Mining Pull Requests to Detect Process Anomalies in Open Source Software Development. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639196

1 INTRODUCTION

Open Source Software (OSS) has now been widely accepted and integrated into various commercial software, forming the so-called *OSS supply chain*. Consequently, the security and trustworthiness of the OSS supply chain has gradually become a critical and fatal concern [13]. OSS poisoning refers to the act of intentionally implanting malicious code into OSS software, which has the characteristics of unexpected, imperceptible, and huge impact in a short period of time. Therefore, commercial software companies have an urgent need to assess the risks of OSS projects and turn on early warning alarms as early as possible when they happen. Approaches like code security scanning can only reduce the damage already made by poisoning, but offer little help to detect it at the earliest time. For instance, Squires purposefully corrupted and deleted his independently developed and maintained project "faker.js" [25]. By that time, the damage to the projects referencing this package was irreparable. The new team standardized the Pull Request (PR) process to eliminate risks, involving multiple people in each PR. Another study [19] surveyed 17 domain experts and 134 developers, and found that PR process is one of the most useful safeguards for OSS supply chain poisoning. From the perspective of consumers in the OSS chain, the standardization of the implementation of the safeguard such as PR can indicate the degree of risk of a project.

PR is a typical collaborative development model commonly adopted by the OSS community [15]. As of 2021, there have been more than 170 million PRs on GitHub[1]. Hence, when evaluating the trustworthiness of a software project, the foundation of its trustworthiness lies in the trustworthiness of its development process [34]. Therefore, both the rationality and the standardization of the PR process are crucial in OSS development. A standardized and efficient PR collaboration process can attract more community members to actively contribute to OSS projects, thus promoting its healthy and development stability [27]. The goal of this work is to study such widely used and critically important safeguard for OSS poisoning, specifically investigating the normal patterns of the PR process and analyzing anomalies in the PR process. Specifically, we derive three Research Questions (RQs) as follows:

RQ1: What are the control-flow anomalies in the PR process? The control-flow anomalies refer to the differences between a PR's event flow and the normal pattern. Typically, they occur rarely in historical data.

RQ2: What are the semantic anomalies in the PR process? The semantic anomalies refer to the contradictions in the process, e.g., a PR that was recommended for rejection was merged.

RQ3: What are the normal patterns of PR processes? As opposed to anomalies, this study considers process patterns discovered after excluding low-frequency event flows as normal patterns.

Due to the openness of OSS community, nevertheless, the PR process presents a high degree of complexity, like "spaghetti", in which the anomalies in developers' behaviors and the associated potential risks to the OSS projects are difficult to detect. We propose a process-mining-based approach to analyze the event logs of PR and identify both the control-flow (global) anomalies and semantic (local) anomalies. Compared with the general anomaly detection methods based on deep learning [30, 31], process mining is able to comprehensively analyze the root causes of anomalies and provide higher explainability. It is especially helpful for researchers and practitioners to understand and analyze the real development workflows, reveal the rules and patterns in the processes, and support process improvement and optimization [43].

Specifically, we constructed the process models of the normal patterns for four scenarios. With the 18 projects chosen from the OSS supply chain of our industry partner, the proposed approach is able to identify four types of semantic anomalies, and four types of control-flow anomalies as well. We further analyze and synthesize the root causes of these anomalies. Finally, after excluding all the anomalous PRs, we are able to identify normal patterns and recover the process models for each scenario. All these models performed well in terms of fitness [43], accuracy [1], generalization [7], and simplicity [46]. In addition, we used these process models to conduct anomaly detection on four OSS projects where poisoning incidents were reported. They all have a higher proportion of anomalies. The main contributions of this paper are as follows:

- We propose a process-mining-based approach to detecting anomalous PRs from two perspectives, namely control-flow and semantic.
- We identify four types of semantic anomalies and their 19 root causes, as well as four types of control-flow anomalies and their 26 root causes.
- We model and analyze the normal patterns for four different collaboration scenarios. We also discuss the potential of using normal patterns to identify project risks.

2 RELATED WORK

This section introduces the research related to pull request and software process mining respectively.

2.1 Related Work on Pull Request

There are five popular research fields related to PR, namely the analysis and prediction of whether a PR will be merged, analysis of reasons why PRs were closed, analysis of influencing factors of PR response time, analysis of user behavior in PRs, as well as PR practices and smells.

PR merge. Tsay et al. [41] found PRs with more comments are less likely to be merged. Azeem et al. [2] found PRs with clear titles and changes are more likely to be merged by project managers. Saini et al. [40] found that the number of code conflicts contained in a PR affects the responsiveness of reviewers and managers to the PR, and also affects the probability of the PR being merged. Mohamed et al. [28] found that the historical PR acceptance rate of contributors is also an important consideration for managers when making PR decisions. Dey et al. [10] found 15 metrics significantly affect the probability of PR being merged in NPM ecosystem.

PR closure. Gousios et al. [15] manually reviewed 350 closed PRs and found that 27% were closed due to multiple contributors submitting similar PRs, 16% were closed due to contributors submitting PRs that managers were not interested in or thought to be of little value, and 13% were closed due to problems with the code in the PR. Li et al. [23] found two other reasons for PR closure, either the contributor did not respond to the last change request or the reviewers did not provide feedback on the last revision.

PR response time. Kononenko et al. [18] found that the number of commits and file changes included in a PR would affect the response time for review and decision-making. Weißgerber et al. [48] found that the number of changes contained in a PR is an important factor affecting the response time to accept or reject PRs. Gousios et al. [15] found that the more historical contributions a contributor has made to a project, the shorter the response time of maintainers. Veen et al. [45] regarded PRs that receive user updates in 1

[1]https://octoverse.github.com/2021/

day as important PRs, and prioritized PRs by predicting the probability of PR being updated in time.

PR behaviors. Ben et al. [4] provided a comprehensive analysis of developer behavior and compared the developers' individual work to the project's overall work to evaluate their contributions. Onoue et al. [32] analyzed the technical characteristics of developers and the activeness of user behavior to help classify developers. Yu et al. [50] constructed a developers' follower network on GitHub and discovered four social behavior patterns from it.

PR practices and smells. There are a good number of studies that investigated specific code review best practices and smells. Macleod et al. [26] conducted semi-structured interviews with 15 developers within Microsoft, and proposed 35 best practices of code review from the perspectives of contributors, reviewers, and organizations. Li et al. [23] conducted a mixed-methods empirical study and revealed 14 code review practices for requesting changes to PRs. Chouchen et al. [8] defined five anti-patterns of modern code review. The study [11] identified 7 review smells through a multivocal literature review. However, these smells mainly come from the subjective opinions and knowledge of developers and researchers rather than the characteristics presented in historical data. In addition, their studies did not cover the entire PR process.

2.2 Related Work on Software Process Mining

Process mining was introduced into software process research by Rubin et al. [36] in 2007. Poncin et al. [35] further demonstrate the application of process mining in specific scenarios, such as developer roles and bug life cycles. The position paper [3] suggests four perspectives of mining requirements, i.e. time, case, organization, and control-flow.

Process mining was used by several studies to discover the normal and risky patterns of the software process. However, they almost all focused on processes at a high level of abstraction, rather than going deep into specific practices such as pull requests. Rubin et al. [37, 38] reported their experiences in applying process mining in the analysis of the development process of industrial software systems. The study [51] demonstrates a tool capable of identifying risks by examining the consistency of actual development processes and human-defined processes. The study [12] collected contribution guidelines of 53 GitHub projects and compared them with actual processes mined from historical event logs.

3 DEFINITIONS

This section describes research objects and explains related terms.

We aim to study *anomalies* in PR processes using process mining. With reference to the definition of process anomalies suggested by Bezerra et al. [6], *anomaly* in this paper can be defined as the process pattern with low frequency and deviation from the normal patterns. The *normal patterns* depict the processes conforming to the business process specification, that is, the process should meet the goal of ensuring that the PR is trustworthy and manageable. The deviation can be measured by the fitness between a specific process and the normal *process pattern*. A fitness of 1 indicates the specific process is fully consistent with one of the normal patterns, while a lower value (always > 0) indicates a larger discrepancy.

The *process patterns* can be described by a *process model*, which is the formal expression of a part of the process. The *process model* is mined from *event logs* and built based on a certain language such as Petri net, BPMN, etc. Table 1 presents an example of the *event log* used for process mining. A case is a complete PR process, consisting of a set of events that occur in chronological order.

Table 1: An example of the preprocessed event log.

Case ID	Event ID	Activity	Timestamp	Actor
1	20230301	SubmitCommit	2021-01-04 06:30:46	Jack
	20230302	OpenPR	2021-01-04 06:32:46	Jack
	...	...	...	...
...	...	...	...	...

We identify and analyze anomalies in the PR process from a *control-flow (global)* perspective and a *semantic (local)* perspective, respectively. The *control-flow anomaly* indicates that an event flow has lower overall consistency with the normal pattern. The common methods for detecting *control-flow anomalies* in the field of process mining is to use the conformance checking algorithm to replay the case on the process model to calculate the fitness [5, 6, 29, 39]. A low fitness indicates a large deviation between the case and the process model, and the case would be considered as anomaly. Deviations may be due to some activities in the case that are not included in the process model, or incorrect execution order of activities in the case compared with the process model. The *semantic anomaly* indicates that the local event *transition* of the event stream does not conform to the process specification. For example, *transition ReviewRejected* $\rightarrow$ *MergePR* should indicate a *semantic anomaly* as the PR was merged despite the reviewer's suggestion for rejection. Sec. 4 will explain the types of anomalies in detail.

4 APPROACH

Figure 1 illustrates the overview of our approach.

4.1 Data Acquisition and Preprocessing

4.1.1 Data Acquisition. The PR logs can be crawled via GitHub Timeline API[2]. The official GitHub documentation[3] lists all event types that can be obtained through the API.

4.1.2 Data Cleaning. We eliminated the PR logs for which the process is not yet complete, i.e. PRs that are currently open. Besides, we treated earlier closed and then reopened PRs as two separate processes. To avoid excessive loop structures, we merged the executions repeated multiple times by the same actor into one activity.

4.1.3 Selection of Key Activities. Within the 46 types of PR activities on GitHub, we only included the activities related to code commit, code review, and decisions on the PR since introducing too many other activities would make the model unintelligible. The descriptions of the selected key activities are presented in Table 2. Based on the different attitudes expressed in the review comments (represented as statuses on GitHub), we refined *Reviewed* as *ReviewComment*, *ReviewRejected*, and *ReviewApproved*. Similarly,

[2]https://docs.github.com/en/rest/issues/timeline?apiVersion=2022-11-28
[3]https://docs.github.com/en/webhooks-and-events/events/issue-event-types

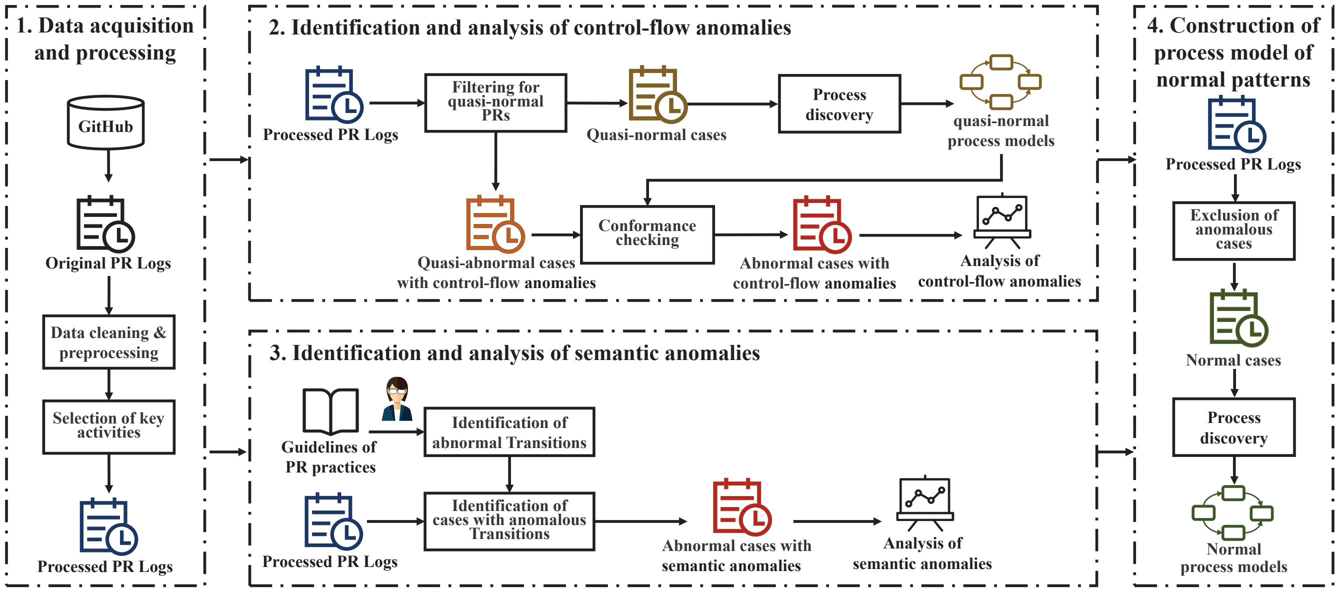
1. Data acquisition and processing
GitHub
Original PR Logs
Data cleaning & preprocessing
Selection of key activities
Processed PR Logs
2. Identification and analysis of control-flow anomalies
Processed PR Logs
Filtering for quasi-normal PRs
Quasi-normal cases
Process discovery
quasi-normal process models
Quasi-abnormal cases with control-flow anomalies
Conformance checking
Abnormal cases with control-flow anomalies
Analysis of control-flow anomalies
3. Identification and analysis of semantic anomalies
Guidelines of PR practices
Identification of abnormal Transitions
Processed PR Logs
Identification of cases with anomalous Transitions
Abnormal cases with semantic anomalies
Analysis of semantic anomalies
4. Construction of process model of normal patterns
Processed PR Logs
Exclusion of anomalous cases
Normal cases
Process discovery
Normal process models

Figure 1: Overview of PR anomaly detection approach.

Committed can be refined as *SubmitCommit* and *Revise* according to the stage in which the activity occurs in the PR process.

Table 2: Identified key activities in PR process.

Activities	Descriptions	Related Actors
OpenPR	Create a PR.	Contributor
SubmitCommit	Submit code changes before *OpenPR*.	Contributor
Revise	Submit code changes after *OpenPR*.	Contributor
ReviewRequested	Request a PR review.	Contributor
ReviewRequest-Removed	Remove a PR review request.	Contributor
ReviewApproved	Agree to merge PR.	Reviewer
ReviewRejected	Refuse to merge PR until the problem is resolved.	Reviewer
ReviewComment	Be neutral and only give review comments.	Reviewer
IssueComment	General comment.	All roles
ClosePR	Close PR.	Maintainer/Contributor
MergePR	Merge PR.	Maintainer
DeleteBranch	Delete the head branch associated with the PR.	Maintainer
Referenced	Reference an issue (PR) in a commit message.	All roles

4.2 Control-flow Anomalies

The definition of control-flow anomalies is based on the assumption that the pattern of anomalies is of low-frequency. However, low-frequency does not exactly mean anomalous, that is, the consistency of the low-frequency cases with the normal pattern is not necessarily low. We consider low-frequency cases as candidate anomalies, since low-frequency cases are more likely to contain anomalous patterns [5], and thus are more likely to have a lower fitness. Therefore, we first proposed a Minimum Overall Coverage-based Log Partition Algorithm (MOCLPA) to filter out high frequency cases and used them to discover a quasi-normal process model. Then, we apply the conformance-checking algorithm to measure the fitness between the low-frequency cases and the quasi-normal model. Those with low fitness are considered anomalous, and those with high fitness are considered normal.

4.2.1 Filtering for Quasi-normal PRs. The event log can be denoted as a set of traces. A trace is an activity sequence of all events in a case sorted by timestamp, which is a more compact representation of a case. Different cases may have the same trace, and the same trace is regarded as a *process variant*. Hence, there is a many-to-one relationship between process cases and process variants. High-frequency process variants contain frequent patterns of business processes, whilst frequent patterns tend to follow normal processes [5]. The MOCLPA, which is shown in the Algorithm 1, aims to split the event log into two groups of cases according to the occurrence frequency of process variants. One group (V_{high}) is composed of high-frequency process variants, which are used to discover a quasi-normal process model. Another group (V_{low}) is composed of low-frequency process variants, which are regarded as a set of candidate anomaly processes.

Algorithm 1 Minimum Overall Coverage based Log Partition Algorithm (MOCLPA).

```
Require: Event Log $L = \{t_1, t_2, \cdots, t_n\}$, Minimum Overall Coverage thr
Ensure: Quasi-normal Case Set $L_{qnormal}$, Candidate Anomaly Case Set $L_{suspicious}$
 1: $V_{high} = \emptyset$, $V_{low} = \emptyset$, cnt = 0, cover_rate = 0
 2: total_case_num = get_case_num(L)
 3: variants = get_variants(L)
 4: order_variants = sort_variants_by_freq(variants, desc=true)
 5: for variant_name, variant_freq in order_variants do
 6:     if cover_rate $\leq$ thr then
 7:         $V_{high}$.add(variant_name)
 8:     else
 9:         $V_{low}$.add(variant_name)
10:     end if
11:     cnt = cnt + variant_freq
12:     cover_rate = cnt/total_case_num
13: end for
14: $L_{qnormal}$ = filter_case_by_variants(L, $V_{high}$)
15: $L_{suspicious}$ = filter_case_by_variants(L, $V_{low}$)
```

4.2.2 Process Discovery. We applied Inductive Miner-infrequency (IMi) [22] to discover the quasi-normal process model, which is an improved variant of the Inductive Miner (IM) [21]. We also tried other well-known methods such as Alpha Miner [44] and Heuristic Miner [47], but IMi can construct a more understandable model.

A model discovered using all logs remains elusive. Hence we model four different PR collaboration scenarios separately. To be specific, there are two different application scenarios of the PR in the open source community, namely *fork and pull model* (abbreviated as *fork*)[4] and *shared repository model* (abbreviated as *non-fork*)[5]. In the *fork model*, the head branch of the PR comes from the forked repository. Whilst in the *shared repository model*, the head branch of the PR comes from the base repository. Furthermore, PR has two final states, *merged* and *closed*. According to the two types of development models and the final states of PRs, we classified event logs into four scenarios, which are as follows:

fork & merge: fork and pull model & PR is merged.
fork & close: fork and pull model & PR is closed.
non-fork & merge: shared repository model & PR is merged.
non-fork & close: shared repository model & PR is closed.

We used IMi to construct quasi-normal process models for these four scenarios respectively.

4.2.3 Conformance Checking. There are two commonly used methods, which are Token-based Replay [43] and Alignments [43]. We used Alignments as it can provide more detailed and easy to understand diagnostic information, which clearly indicates where inconsistencies occur between the case and process model. For each of the four scenarios, we performed the conformance checking respectively. We checked the consistency between each candidate anomaly case and the quasi-normal process model of the scenario it belongs to. The Alignments method would produce a fitness value as an indication of whether the case is anomalous. Besides, the specific location of the deviation between the case and the model would be located for subsequent in-depth analysis.

4.2.4 Analysis of Control-Flow Anomalies. We referred to the five types of control-flow anomalies defined by Nolle et al. [30] for analysis. We do not include the type "attribute" because there is no applicable data for this study. The descriptions of all the control-flow anomaly types are shown in Table 3. Two researchers further identified the root causes of anomalies by analyzing comments and context in PR. Any discrepancies between them were discussed to reach a consensus.

Table 3: Types of control-flow anomalies.

Types	Explanations
skip	A necessary activity has not been executed.
insert	An activity that has never been executed has been executed.
rework	An activity has been executed too many times.
early	An activity has been executed early.
late	An activity has been executed late.

[4]https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/getting-started/about-collaborative-development-models#fork-and-pull-model
[5]https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/getting-started/about-collaborative-development-models#shared-repository-model

4.3 Semantic Anomalies

It can be difficult to detect anomalies when the overall control-flow of the process appears consistent with normal patterns, but there may still be individual instances of anomalous behavior. Therefore, we further identify the semantic anomalies. To address this issue, we analyzed all transitions of activities in the form of "Activity A → Activity B" to identify the anomalous transitions based on the semantic understanding of the PR process. Then we applied a rule-based method to detect all the cases with anomalous transitions.

4.3.1 Identification of Anomalous Transitions. We referred to the official GitHub documentation[6] as well as relevant research on PR best practices [16, 23, 26] and analyzed all the transitions in the form of "Activity A → Activity B". As a result, we identified four types of semantic anomalies as shown in Table 4. We identified these four semantic anomalies according to the two ground rules, namely the review request should be responded to, and the the decision on the PR should be consistent with the review results. In fact, we also found other potential anomalous scenarios where the request for review was not initiated and the PR was merged or closed even without review, i.e. OpenPR → MergePR and OpenPR → ClosePR. These two scenarios are very common, accounting for about 11.3% of all 17531 cases. The study [17] indicates that it is reasonable for maintainers to directly review the code then decide to approve or reject the PR.

Specifically, after a PR is created, contributors or maintainers can request a review from core members. If the review request is initiated by the contributor, it indicates that the contributor expects to receive approval and feedback on the PR from core members. If the review request is initiated by the maintainer, it indicates that the maintainer values the reviewer's opinion and the quality of the PR [16]. In either case, once a review request is initiated, the requested reviewer should give feedback in a timely manner, and the maintainer should also wait for the reviewer to give a clear review comment before deciding whether to merge it [26]. we found two types of PRs that did not meet the above conditions, 1) the reviewer did not respond to the review request, but the PR was closed, 2) the reviewer did not respond to the review request, but the PR was merged.

If there is any problem, the contributor should make revisions until the reviewer confirms that the PR is correct. The maintainer should generally refer to the last comment given by the reviewer to decide whether to merge the PR. That is, the decision of whether to merge should be consistent with the final comment [23]. We found two types of PRs that did not meet the above conditions, 3) the final review comment was approved, but the PR was closed, 4) the final review comment was rejected, but the PR was merged.

4.3.2 Identification of Anomalous Cases with Anomalous Transitions. We used a rule-based method to detect anomalous cases. For each case, we iterate through all transitions it contains. As long as it contains one of the four types of anomalous transitions, it would be detected as an anomalous PR.

[6]https://docs.github.com/en/pull-requests

Table 4: Types of semantic anomalies.

Types	Explanations
ReviewRequested ⟶ *ClosePR*	The reviewer has not responded to the review request, but the PR was closed by the maintainer.
ReviewRequested ⟶ *MergePR*	The reviewer has not responded to the review request, but the PR was merged by the maintainer.
ReviewApproved ⟶ *ClosePR*	The reviewer approved the PR in the last review, but the PR was closed by the maintainer.
ReviewRejected ⟶ *MergePR*	The reviewer rejected the PR in the last review, but the PR was merged by the maintainer.

4.3.3 Analysis of Semantic Anomalies. In order to explore the root causes of these four types of semantic anomalies, we manually analyzed the complete process, review comments, and discussions of anomalous PRs. We referred to the reasons why PRs were closed summarized by Gousios et al. [15]. Additionally, we identified three more reasons. The detailed analysis results are discussed in Sec. 5. For the PRs with semantic anomalies that were finally merged, we identified root causes by analyzing comments and consulting maintainers since there is no relevant research to provide reference.

4.4 Construction of Process Models

After identifying the semantic anomalous PRs and control-flow anomalous PRs, we can eliminate all the anomalous cases so as to recover normal patterns only with the normal PR cases.

4.4.1 Exclusion and Evaluation of Anomalous Cases. We excluded all the anomalous cases with control-flow anomalies or semantic anomalies from the complete case set, and obtained a set that contains and only contains all normal cases.

4.4.2 Process Discovery. We first classified the cases based on the four scenarios discussed in Sec. 4.2.2. Then we used the IMi algorithm to construct the process model for each scenario.

4.5 Multi-Case Study

This research is carried out in collaboration with a global Information and Communication Technology (ICT) enterprise. With the 18 most frequently used OSS projects in their software supply chain, we conducted a multi-case study to evaluate our approach. The replication package is available on GitHub[7].

4.5.1 Projects. The information on these 18 projects is shown in Table 5. They involve the main development languages (Java, C++, etc.) at a variety of project scales. The event logs recorded in 2021 and 2022 that were crawled via the Timeline API resulted in 17531 PRs after data cleansing, of which 11820 PRs were merged and 5423 PRs were closed. We did not use the all available historical data for the following reasons: 1) the manual analysis of the anomalies is an effort-consuming task (over two man-months in this study); 2) we selected the most recent data with a sufficient volume for process mining; 3) our experience indicates that the practices of the GitHub community keep improving (e.g., GitHub Action is available in 2019, and it could impact the PR process [9, 42, 49]), and adding excessive earlier data may introduce even more noise.

[7]https://github.com/ICSE2024-905/Replication-Package-of-Mining-PRs

Table 5: Information of selected projects.

Project Name	Lan.	#Stars	#PRs	#Merged	#Closed
apache/dubbo	Java	38.1k	1817	1519	298
apache/hadoop	Java	13.1k	2078	1597	481
apache/httpd	C++	3.1k	79	0	79
apache/netbeans	Java	2.1k	1460	1310	150
apache/zookeeper	Java	11k	225	11	214
gpac/gpac	C++	2k	60	45	15
ImageMagick/ImageMagick	C++	8.2k	75	67	8
libexpat/libexpat	C++	827	154	142	12
madler/zlib	C++	4.1k	35	0	35
opencv/opencv	C++	65k	1222	1055	167
openzipkin/zipkin	Java	15.8k	43	36	7
phoenixframework/phoenix	Elixir	18.8k	514	376	138
redis/redis	C++	7.9k	1744	1456	288
spring-cloud/spring-cloud-function	Java	923	71	22	49
spring-projects/spring-framework	Java	50.2k	641	109	532
stefanberger/swtpm	C++	387	241	223	18
tensorflow/tensorflow	C++	169k	5208	3852	1356
vim/vim	C++	29.1k	1576	0	1576

4.5.2 Parameter Settings. Simplicity may be compromised if uncommon behavior is included in the process model whilst fitness may be compromised if uncommon behavior is excluded from the model. It is a common approach [22] to adopt the Pareto Principle in the process discovery. According to the Pareto Principle, a suitable process model should cover 80% of the behavior in the event log, so we set the minimum overall coverage in the MOCLPA as 0.8. The IMi algorithm has a single hyper-parameter, K, which is utilized to differentiate between *frequent* and *infrequent* behaviors. Its value can range from 0 to 1. We tried different values and set it to 0.1 based on the trade-off between the intelligibility and the degree of information retention of the model.

To identify control-flow anomalous PRs, a fitness threshold needs to be set in the conformance checking. The PR with the fitness lower than the threshold would be regarded as the PR with control-flow anomalies. As explained in Sec. 4.2.1, we treat the set selected with MOCLPA as quasi-normal cases based on the assumption that fitness between the quasi-normal cases and the quasi-normal process model built with them should be beyond the threshold. Therefore, we performed the conformance checking on each case in the quasi-normal case set. Finally, we set 0.6 as the threshold.

4.5.3 Model Assessment Metrics. There are mainly four common dimensions for assessing recovered process models, namely fitness [43], accuracy [1], generalization [7], and simplicity [46]. The value of each metric ranges from 0 to 1, and the closer of the result to 1 means the better performance.

5 RESULTS AND ANALYSIS

5.1 Control-Flow Anomalous PRs (RQ1)

The identification of control-flow anomalies is based on the conformance checking, that is, cases with a low degree (less than 0.6) of fitness to the quasi-normal patterns are judged as anomalous PRs. In other words, normal cases are not necessarily consistent with the quasi-normal pattern, and anomalous cases are those PRs that are quite different from the quasi-normal pattern. We only detected control-flow anomalies in three PR collaboration scenarios, which are *fork & close*, *non-fork & merge*, and *non-fork & close*. There is a wealth of variants (8179 cases with 2660 variants) in the *fork & merge* scenario. The review phase of the process presents

Table 6: Root causes of control-flow anomalies.

Types	Related Activities	Root Causes	Scenarios	Total
rework	*SubmitCommit*	Code synchronization between different branches in the base repository.	non-fork & merge	33
		Large code patch or the code is greatly refactored.	non-fork & merge	3
		The PR created by the contributor is informal.	non-fork & close	15
		Unknown.*	non-fork & close	12
		The PR created by the contributor does not meet the project specifications.	non-fork & close	9
	ReviewComment ⟶ *Revise*	The reviewer made multiple comments, and the contributor made multiple revisions based on the comments.	non-fork & merge	12
	IssueComment	**The reviewer misunderstood the PR and discussed it with the contributor**	non-fork & merge	2
		Multiple reviewers approved merging the PR in general comment	non-fork & merge	1
		The reviewer discussed the reason for CI failures with the contributor.	non-fork & merge	1
		The bot generated a test report every time once a new commit was submitted	non-fork & merge	1
	ReviewRejected ⟶ *Revise*	The reviewer made multiple comments, and the contributor made multiple revisions based on the comments.	fork & close	2
	ReviewRequested ⟶ *ReviewApproved*	The contributor did not submit new changes, but was continuously initiating review requests. Reviewers were continuously providing approval comments.	fork & close	1
skip	*DeleteBranch*	The associated branch of the PR is the resident branch of the base repository.	non-fork & merge	29
	SubmitCommit	**Due to the contributor using *force push* command, some commit records are missing.‡**	non-fork & merge	10
	IssueComment	**The reviewer did not respond to the review request.**	non-fork & close	1
	Revise	**The contributor did not respond to the requested changes.**	non-fork & close	3
		The contributor did not respond to requested changes.	fork & close	1
early	*ReviewApproved*	**The reviewer agreed to merge the PR without carefully reviewing it.**	non-fork & merge	4
insert	*ReviewRequestRemoved*	The maintainers assigned themselves as reviewers by mistake and then deleted the review request.	non-fork & merge	1
		The maintainer updated the review request initiated to assign another reviewer.	non-fork & close	1
		The maintainer updated the review request initiated to notify the reviewer to review again.	non-fork & close	1
		The maintainer assigned inappropriate reviewers, causing every review request to be rejected.	fork & close	3
		The contributor updated the review request initiated to notify the reviewer to review again.	fork & close	1
		The bot automatically removed some of the review requests initiated by the maintainer.	non-fork & close	1
		The bot assigned inappropriate reviewers, so the maintainer remove the review request.	fork & close	1
	ReviewApproved	**The reviewer agreed to merge the PR without carefully reviewing it.**	non-fork & close	1
	ReviewComment	The bot divided the test report into multiple segments and reported sequentially.	non-fork & close	1

[1] Bold font indicates obviously unreasonable behaviors.
[2] "*" indicates an unknown reason due to the lack of context and relevant personnel did not respond to emailed inquiries.
[3] "‡" indicates a lack of context in the PR to identify the root cause, but relevant personnel responded.

diversity and inclusiveness, whilst overall there is significant certainty from *OpenPR* to review related activities to *MergePR*. This may be the reason why the fitness of the cases in the *fork & merge* scenario is generally high.

Table 6 presents the root causes of each type of control-flow anomalies. The Total column indicates the total number of cases from all projects that have anomalies caused by the same root cause. Due to the lack of detection of *late* anomalies on our selected projects, only four types of control-flow anomalies are shown in the table 6. An anomalous PR usually contains multiple anomalies, hence the root cause of a single anomaly may be reasonable, but the anomalous PRs are quite different from normal patterns. The root causes for control-flow anomalies are diverse, and we highlight the root causes that demonstrate obviously unreasonable (non-standard) behaviors. In addition, the root causes of 22 control-flow anomalies are unknown due to the lack of context. The unknown is meaningful as it reveals an anti-pattern, namely the lack of discussion in PRs. We emailed 5 participants of these PRs and received one response. The respondent confirmed the anomaly we pointed out with a specific explanation, i.e. the use of force push command.

In the *non-fork & merge* scenario, there are two main manifestations of control-flow anomalies, namely, the *SubmitCommit* was executed multiple times, and the *DeleteBranch* was skipped. We found that these two types of anomalies mainly occur in conjunction with each other in "netbeans". Specifically, before and after the release of a new version in the "netbeans", the code synchronization would result in repeated execution of *SubmitCommit*. The branch associated with the new release is generally the resident branch of the base repository, it would not be deleted after the version's release, therefore the *DeleteBranch* would be skipped. From an intra-project perspective, it seems like a reasonable process pattern. However, from a cross-project perspective, this is an anomalous process because it only happens in "netbeans". In addition, we found two obviously unreasonable behaviors in this scenario. Specifically, the contributor uses **force push** commands[8], which can result in some commits submitted by others being overwritten (e.g., libexpat#550); the reviewer agreed to merge the PR without carefully reviewing it (e.g., phoenix#4431), which is an obviously unreasonable behavior that may cause malicious code to be merged into the repository; the reviewer misunderstood the PR

[8] https://git-scm.com/docs/git-push#Documentation/git-push.txt--f

and discussed it with the contributor (e.g., phoenix#4309). The reviewer's misinterpretation of the PR may lead to the contributor's invalid revisions [23], and could even cause the contributor's doubt about the reviewer's abilities and destroyed motivation for working on the project [20, 23].

In the *non-fork & close* scenario, the main manifestation is the repeated execution of *SubmitCommit*. There are two root causes that reflect obviously unreasonable behaviors. First, the PR created by the contributor is informal (e.g., spring-framework#26333). Second, the PR created by the contributor does not meet the project specifications. For example, the associated branch of PR does not meet project requirements (e.g., tensorflow# 48181); the PR contains many commits that are not related to project requirements (e.g., dubbo#10162). Furthermore, projects like "zookeeper" and "tensorflow" utilize bots to automate certain processes, which could lead to unexpected situations. For example, the bot automatically removed the review request (e.g., tensorflow#49499); the bot generated multiple test reports for intensive code submissions (e.g., zookeeper#1707). In addition, we also observed some anti-patterns with a lack of commonality. Specifically, the contributor did not respond to the requested changes (e.g., tensorflow#49562), which could waste significant effort and time invested by the reviewer [24]; the reviewer did not respond to the review request (e.g., tensorflow#50390). Lack of responsiveness from reviewers can demotivate contributors from engaging [20, 24]; and the reviewer agreed to merge the PR without proper review (e.g., tensorflow#54425), which could increase the risk of successful project poisoning.

In the *fork & close* scenario, there are various causes for the anomalies, but they mainly occurred in the "tensorflow". We found three obviously unreasonable behaviors in this scenario. Specifically, the contributor did not respond to requested changes; the maintainer or the bot assigned inappropriate reviewers, causing every review request to be rejected. Team members within a project can have different technical backgrounds and expertise [33]. If inappropriate reviewers are assigned, it may result in PR not receiving timely responses or reviewers providing inaccurate feedback [23], thereby reducing the participation of contributors.

Main findings for RQ1: there are various reasons for control-flow anomalies in different PR collaboration scenarios, and quite a few of them are reasonable (harmless). The contributors' main unreasonable behaviors are 1) creating informal PRs, 2) failing to respond to reviewers' comments in a timely manner, 3) creating PRs that do not comply with project specifications. The reviewers' main unreasonable behaviors are 1) not responding to the review request in a timely manner, 2) agreeing to merge the PR without careful review. The maintainers' main unreasonable behavior is assigning inappropriate reviewers. In addition, the bots may lead to some unexpected situations.

5.2 Semantic Anomalous PRs (RQ2)

5.2.1 Root Causes of Semantic Anomalies of Closed PRs. For the PRs with semantic anomalies that were finally closed (*ReviewRequested* → *ClosePR* and *ReviewApproved* → *ClosePR*), we manually analyzed and synthesized the comment information related to the PR decision and classified the root causes referring to the study [15]. The identified root causes are shown in Table 7.

Table 7: Root causes of semantic anomalies of closed PRs.

Types	Root Causes	Total
obsolete	The PR is no longer relevant, as the project has progressed.	24
conflict	There feature is currently being implemented by other PR or in another branch.	17
superseded	A new PR solves the problem better.	15
duplicate	The functionality had been in the project prior to the submission of the PR.	14
superfluous	PR does not solve an existing problem or add a needed feature	13
deferred	Proposed change delayed for further investi-gation in the future.	17
tests failed	Tests failed to run.	3
incorrect implementation	The implementation of the feature is incorrect, missing or not following project standards.	21
merged	The PR was identified as merged by the human examiner.	173
low priority	The current PR integration priority is low.	21
mistake	A contributor or project member made a mistake and accidentally closed the current PR.	7
invalid	Sample PRs created by contributors who did not contribute any valuable code.	8
unknown	Reason unknown due to lack of discussion information.	77

Table 8: Root causes of semantic anomalies of merged PRs.

Types	Root Causes	Total
ReviewRequested → *MergePR*	**The developers communicate on other platforms.‡**	3
	Fix urgent issue and PR passes all tests.‡	1
	A backport of another PR.‡	3
	Unknown.*	123
ReviewRejected → *MergePR*	The contributor explains why changes are not needed.	7
	Wrong operation of the bot.	2
	Wrong operation of the maintainer.	1
	To avoid delays, the reviewer made revisions by themselves	2
	Unknown.*	1

[1] Bold font indicates obviously unreasonable behaviors.
[2] "*" indicates an unknown reason due to the lack of context and relevant personnel did not respond to emailed inquiries.
[3] "‡" indicates a lack of context in the PR to identify the root cause, but relevant personnel responded.

The most common reason for a PR to be closed abnormally is that it has been merged indirectly through other means (e.g., using the cherry-pick command[9]), which is categorized as the *merged* type. We emailed 12 participants of these 173 anomalous PRs to ask about the root causes for such an unusual pattern. We received one response from one of the maintainers of "zookeeper" and they confirmed that this is an obviously unreasonable practice. Their team used a script to automatically process PR decisions in the early stage of the project. Specifically, when the maintainers decide to merge the PR, they would use a script to create a new commit and push it to the base repository. The new commit is associated with the current PR and contains all the changes in it. When the new commit is successfully pushed to the base repository, the associated PR will be automatically closed. The main reason for using such a script is twofold. First, to compress multiple commits into one, with the aim of keeping a mostly linear history. Second, to generate standardized commit messages, with the aim of facilitating project management. However, unexpected situations may occur when using scripts, e.g., errors in the execution of the script may cause the status of the PR not to be updated (e.g., zookeeper#1607). To avoid unexpected situations, the maintainer

[9]https://git-scm.com/book/en/v2/Distributed-Git-Maintaining-a-Project#_rebase_cherry_pick

recommends using the GitHub button to merge the PR instead of the script.

There were 77 anomalous PRs, for which we were unable to identify the cause due to lack of context. We emailed 24 maintainers of these PRs but received no response.

All the other root causes (*obsolete*, *conflict*, *superseded*, *superfluous*, *duplicate*, *deferred*, *tests failed* and *incorrect implementation*) were found, but their proportions are at low levels. In addition, we found three other root causes of why PRs were closed abnormally, which are denoted as *low priority*, *mistake* and *invalid*.

The *low priority* type is mainly found in project "dubbo", where PRs are characterized by few changes and most of them are related to code style, such as modifying variable names, removing redundant spaces, etc. For PRs with low priority, the maintainer usually temporarily closes them and labels them as "low priority" for easy management (e.g., dubbo#7198). The maintainer would periodically merge the changes in these low priority PRs into a new PR and then merge it. This is indeed a reasonable pattern. However, it should be an anomaly judging based on the original PR alone, because the original PR presents *ReviewApproved* → *ClosePR*.

All the anomalies of *mistake* type are found in project "tensorflow", where PRs are characterized by contributors' mistakes, such as creating a PR by mistake (e.g., tensorflow#51563), choosing the wrong branch when creating a PR (e.g., tensorflow#57800), etc.

The anomalies of *invalid* type are also mainly found in project "tensorflow", where PRs are characterized by informality behaviors such as contributors' pranks (e.g., tensorflow#51714), contributors' individual code testing (e.g., tensorflow#53560), etc.

5.2.2 Root Causes of Semantic Anomalies of Merged PRs.

For PRs with semantic anomalies of type *ReviewRequested* → *MergePR*, almost all maintainers did not provide comments to explain the reasons for merging PRs. So we attempted to contact the maintainers in these PRs via email. We sent 20 emails and received 3 replies. In all replies, the maintainers provided detailed explanations and confirmed our results.

There are 130 PRs with semantic anomalies of *ReviewRequested* → *MergePR*. We reached out to jcchavezs, a maintainer in the 'zipkin' project, and asked about anomalies with regard to zipkin#3411. To keep the project from being affected by the Log4Shell vulnerability[10], jcchavezs submitted a PR for an urgent bug fix. Since the test suite was robust and the PR passed all test cases, jcchavezs merged the PR without waiting for the reviewer to reply. In addition, we also contacted AlbumenJ, a maintainer of the project "dubbo", and asked questions about the anomalies in dubbo#7440. According to AlbumenJ's explanation, he communicated with the reviewer on other apps and reached a consensus, so AlbumenJ directly merged the PR. We contacted lkishalmi, a maintainer in the project "netbeans", and asked questions about anomalies in netbeans#2669. According to lkishalmi's explanation, netbeans#2669 is a backport of netbeans#2259. Since netbeans#2259 has passed the review and was merged into NetBeans 12.1, lkishalmi skipped the code review in netbeans#2669 and directly merged it into NetBeans 12.0.

There are 13 PRs with semantic anomalies of type *ReviewRejected* → *MergePR*. By analyzing the comments in these PRs, we found that seven PRs were merged because the contributor explained why changes were not needed and his response was approved by the maintainer. Two PRs were merged due to the misoperation by the bot. One PR were merged due to the misoperation by the maintainer. Two PRs were merged due to the reviewers made revisions by themselves. The reasons why the remaining one PR were merged by exception is unknown. We tried consulting the maintainer but with no response.

[10]https://avd.aliyun.com/detail?id=AVD-2021-920285

Main findings for RQ2: Semantic anomalies are mainly caused by the maintainers' irregular merging or closing of PRs. The main reason why the PR was closed abnormally was that the maintainer merged into the PR through other means. Although this is an anomaly to the current PR, the maintainers' motivation is to facilitate project management. The reasons why PRs are incorporated anomalously are mostly unknown due to a lack of context information. The survey feedback suggests that maintainers may skip the review process and directly merge the PR when encountering urgent issues. Additionally, maintainers and reviewers sometimes communicate PRs through other channels; bot may also causes some PRs to be merged by mistake.

5.3 Normal Patterns of PR Process (RQ3)

We eliminated all PRs containing control-flow anomalies or semantic anomalies to obtain a set of normal cases. Then, the case sets were divided into four groups according to the four PR collaboration scenarios. The IMi algorithm was applied to each group of case sets to construct a corresponding process model. Table 9 shows the quality assessment results of process models mined from different collaborative sub-scenarios. It can be seen that all process models perform well on the four quality assessment metrics. Given the page limit, all models were shared in the replication package.

The complexity and diversity of the normal patterns are mainly reflected in activities such as review requests, review and revision since these activities may either not occur or may involve multiple participants and cycle multiple times. We manually reviewed these normal patterns and found no new semantic anomalies, which illustrates the effectiveness of our anomaly identification.

We counted the percentage of cases in which each activity transition occurred. Table 10 shows activity transitions that occur frequently (>20%) in at least one scenario. Frequency is the proportion of normal cases containing one kind of transition in a scenario to all normal cases in that scenario. We use different colors to distinguish between high and low frequencies. From the differences in the high frequency transitions across scenarios, we can summarize some findings as follows.

Initiating a review request immediately after a PR is created is more likely to get a positive response, and the PR is more likely to be merged. In the fork & merge and non-fork & merge scenarios, the proportions of cases containing *OpenPR* → *ReviewRequested* are 34% and 36%, respectively, whilst it is less than 10% of the other two scenarios. There is also such an obvious gap in the frequency of *ReviewRequested* → *ReviewApproved*. In addition, we found **the proportion (47.3%) of *ReviewRequested* contained in merged PRs is significantly higher than that of the closed PRs (14.3%)**.

The *non-fork model* tends to skip the review process more than the *fork model*. In the *non-fork & merge* scenario, the ratio of *OpenPR* ⟶ *MergePR* is as high as 42%, whilst it is only 5% in the *fork & merge*. The ratio of *OpenPR* ⟶ *ClosePR* in the *non-fork & close* is 28%, which is almost twice that in the *fork & close*. Furthermore, **there are more multiple rounds of reviews and revisions in the *fork model*** as both *IssueComment* ⟶ *IssueComment*, *IssueComment* ⟶ *Revise*, and *Revise* ⟶ *IssueComment* are significantly more frequent than the *non-fork model*. We have concerns about the risks of such a pattern, i.e. skipping reviews. It is worth studying how to balance efficiency and risk control in the *fork model*.

PRs that are spontaneously and frequently revised are more likely to be closed in the *non-fork model*. In the *non-fork & close*, the percentage of PRs where *OpenPR* ⟶ *Revise* and *Revise* ⟶ *Revise* occurred are 56% and 53% respectively. In other scenarios, the frequency of *OpenPR* ⟶ *Revise* is around 10%, and the frequency of *Revise* ⟶ *Revise* is less than 5%. In the *fork model*, revisions are usually associated with reviews. That is, revisions are usually the result of interactions between contributors and reviewers, rather than spontaneous behavior by contributors. In the *non-fork model*, most of the merged PRs were not revised.

Main findings for RQ3: the process model of the normal pattern we built for different scenarios performed well. A notable difference between non-fork model and fork model is that the former tends to bypass the review process more often. This may also mean that the non-fork model is potentially more risky. In addition, PRs with timely reviews have a higher probability of being merged, while PRs with repeated revisions have a lower probability of being merged.

Table 9: Quality assessment results of normal process models under different PR collaboration scenarios.

Scenario	# Cases	# Variants	Fit.	Acc.	Gen.	Sim.
fork & merge	10083	4611	0.88	0.81	0.98	0.65
fork & close	4329	1528	0.88	0.89	0.95	0.68
non-fork & merge	1536	191	0.98	0.95	0.89	0.67
non-fork & close	624	72	1.00	0.77	0.77	0.67

6 DISCUSSION

6.1 Control-flow and Semantic Anomalies

By comparing the fitness distribution of PRs with semantic anomalies against that of normal PRs, we found that the difference between the two distributions is slim. There are 84.2% of the semantic anomalous PRs with a fitness greater than 0.8, which indicates that most of them have little deviation from the normal pattern. The fitness of nearly half (44.4%) of the control-flow anomalous PRs is lower than 0.3. There is a significant deviation between these PRs and the normal model.

In terms of quantity, we identified 99 PRs with control-flow anomalies and 553 PRs with semantic anomalies, among which only 13 PRs contained both control-flow and semantic anomalies. Overall, there is no obvious correlation between control-flow anomalous PRs and semantic anomalous PRs.

Table 10: Frequencies of activity transitions in each scenario.

	fork & merge	fork & close	non-fork & merge	non-fork & close
OpenPR ⟶ *IssueComment*	0.33	0.54	0.10	0.06
IssueComment ⟶ *IssueComment*	0.27	0.32	0.03	0.04
IssueComment ⟶ *Revise*	0.24	0.25	0.04	0.02
Revise ⟶ *IssueComment*	0.28	0.24	0.03	0.02
IssueComment ⟶ *MergePR*	0.27	0.00	0.10	0.00
IssueComment ⟶ *ClosePR*	0.00	0.61	0.00	0.11
MergePR ⟶ *DeleteBranch*	0.29	0.00	0.90	0.00
ClosePR ⟶ *DeleteBranch*	0.00	0.30	0.00	0.43
SubmitCommit ⟶ *OpenPR*	0.80	0.80	0.93	0.45
OpenPR ⟶ *Revise*	0.10	0.12	0.09	0.56
Revise ⟶ *Revise*	0.03	0.02	0.01	0.53
Revise ⟶ *ClosePR*	0.00	0.14	0.00	0.58
OpenPR ⟶ *ReviewRequested*	0.34	0.10	0.36	0.08
ReviewRequested ⟶ *ReviewApproved*	0.29	0.02	0.33	0.01
ReviewApproved ⟶ *IssueComment*	0.20	0.06	0.02	0.01
IssueComment ⟶ *ReviewApproved*	0.30	0.04	0.02	0.00
ReviewApproved ⟶ *MergePR*	0.59	0.00	0.38	0.00
OpenPR ⟶ *MergePR*	0.05	0.00	0.42	0.00
OpenPR ⟶ *ClosePR*	0.00	0.15	0.00	0.28

6.2 Other Potential Semantic Anomalies

We identified a total of four semantic anomalies according to the two ground rules, namely the review request should be responded to, and the decision on the PR should be consistent with the review results. We also found other potential anomalous scenarios where the request for review was not initiated and the PR was merged or closed even without review, i.e. *OpenPR* ⟶ *MergePR* and *OpenPR* ⟶ *ClosePR*. These two scenarios are very common, accounting for about 11.3% of all 17531 cases. It is reported that it is reasonable for maintainers to directly review the code and then decide to approve or reject the PR [17].

Among the 1137 cases with the pattern of *OpenPR* ⟶ *MergePR*, 219 contained comments, 209 of which were about expressing gratitude to the contributors. However, the PR netbeans#3500 was merged before the reviewer provided comments, which is not a standardized process. The PR netbeans#3564 is labelled as "*don't merge*" but it was merged due to the internal mechanism of Git. The maintainer of the PR netbeans#3649 clearly pointed out that it is unwise to merge directly without a proper review. These cases illustrate the potential risks of merging PRs without review.

Given that PRs without review seem common phenomena, this work did not treat them as anomalies, whereas we still have concerns, even those PRs that were recognized by maintainers.

6.3 How to View Anomalies?

Anomalies are the opposite of normal, but they do not necessarily imply risk. What are normal patterns? As a famous saying goes, "*for actually the earth had no roads to begin with, but when many men pass one way, a road is made*". Risks may also lurk in seemingly normal activities, such as the *OpenPR* ⟶ *MergePR* we discussed earlier. There may also be rationality in anomalies, for example, *SubmitCommit* repeated many times may be the behavior before a new version is released soon.

Some anomalies may be normal within a certain project, but they are indeed anomalous from the perspective of cross-projects. For example, there are a large number of *ReviewApproved* ⟶ *ClosePR* type anomalies in "zookeeper" due to the use of bots. The problems of using bots is also indicated in [14]. In the future, establishing standardized processes may be a promising direction to improve the trustworthiness of OSS supply chains as well as to reduce the risks of each project itself. The normal patterns identified in this study can help establish the standardized process.

6.4 Examples of poisoning alarms

Table 11: Amounts and proportions of anomalous PRs in poisoned and un-poisoned projects

Project	Control-flow	Semantic	Total Ratio
Un-poisoned projects*			
apache/dubbo	3	49	2%
apache/hadoop	3	28	1%
apache/httpd	0	5	5%
apache/netbeans	64	35	5%
apache/zookeeper	4	81	26%
gpac/gpac	2	4	8%
ImageMagick/ImageMagick	0	1	1%
libexpat/libexpat	8	0	4%
madler/zlib	0	1	2%
opencv/opencv	3	26	1%
openzipkin/zipkin	0	2	3%
phoenixframework/phoenix	11	6	2%
redis/redis	2	16	4%
spring-cloud/spring-cloud-function	0	6	6%
spring-projects/spring-framework	6	27	3%
stefanberger/swtpm	4	0	1%
tensorflow/tensorflow	40	261	5%
vim/vim	0	5	0%
Poisoned projects			
parse-community/parse-server	59	13	41%
heroku/cli	26	38	20%
php/php-src	0	390	17%
faisalman/ua-parser-js	2	1	8%

[1] "*" projects that have not been publicly exposed for poisoning.

We searched "(software supply chain or OSS software) AND poisoning" on Google and Bing. We excluded general security attack cases, and cases where the repository is not accessible. As a result, we found nine projects (see the replication package for details), and five of which have less than 10 PRs in 2021 and 2022. The four projects left are "parse-community/parse-server"[11], "heroku/cli"[12], "php/php-src"[13], "faisalman/ua-parser-js"[14].

As shown in Table 11, we detected a reasonable number of anomalies in these four projects, and the proportion of their anomalies in the total number of PRs is higher (0.22) than the average of the 18 un-poisoned projects (0.06), and higher than all the other projects except "apache/zookeeper". We performed the Wilcoxon signed-rank test to compare whether there is a significant difference in the data distribution between un-poisoned and poisoned projects. The p-value is 0.0028, which indicates a statistically significant difference in the proportion of anomalous PRs in the two groups.

The "php/php-src" has as many as 390 (17%) anomalous PRs, and 382 of them present the same anomaly, i.e. ReviewApproved $\rightarrow$ ClosePR. The code was directly merged through other commits, and then the PR was closed. There is a number of research focusing on code review practices (referring to Sec. 2), but the real problem with poisoned projects may be that their processes outside of review is risky. These cases imply that the process anomaly detection can alarm the risk of being poisoned, whilst it also suggests that some poisoning methods cannot be discovered through the PR processes as some projects rarely use PR practices. Not using PR may itself indicate risks.

[11] https://nvd.nist.gov/vuln/detail/CVE-2022-24760
[12] https://www.securityweek.com/heroku-shares-details-recent-github-attack/
[13] https://securityaffairs.co/116088/hacking/php-git-server-hack.html
[14] https://portswigger.net/daily-swig/popular-npm-package-ua-parser-js-poisoned-with-cryptomining-password-stealing-malware

7 THREATS TO VALIDITY

To select a reasonable threshold of fitness for filtering control-flow anomalies, we construct a quasi-normal model before the anomaly detection. We experimented multiple process mining algorithms and different hyper-parameters, and selected the best performing configuration for constructing the normal model.

We sent questionnaires via email to participants of anomalous PRs to confirm the accuracy of anomaly detection. Unfortunately, it was difficult to conduct a quantitative evaluation due to low response rates (5/61). However, all the received feedbacks indicate that the anomalies we identified do indeed exist.

To mitigate the potential subjective bias during the data analysis, all the findings were discussed by two authors together, and the results and disagreements were further discussed in weekly meetings involving all authors to reach a final consensus.

We analyzed 18 projects in this study that our industry partner was interested in. The general significance on more projects remains to be studied. The root cause identification is based on our manual analysis of review comments and discussions. Therefore, there are a large number of PRs for which the root causes cannot be identified due to the lack of such information. Further dedicated empirical research is warranted in the future. The future work could carefully select a larger set of PRs to build more general normal patterns.

8 CONCLUSION

This study proposes a process-mining-based approach to identify and analyze anomalous PRs. Our approach can not only help practitioners and researchers to understand the PR process patterns, but also support to identify and assess the normality and risks in the OSS development processes. The trustworthiness of the development process then lays the foundation for the long-term trustworthiness of a project. Our evaluation also reveals that the processes of large and well-known OSS projects may not be as standardized as those of smaller projects. For example, we identified far more anomalies in *tensorflow* than in other projects. The further identified normal patterns in this study may encourage future studies to progressively establish best practices for PR so as to standardize the PR process.

9 ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China (No.62072227, No.62202219, No.62302210, No.72372070), the Jiangsu Provincial Key Research and Development Program (No.BE2021002-2), the National Key Research and Development Program of China (No.2019YFE0105500), and the Innovation Project and Overseas Open Projects of State Key Laboratory for Novel Software Technology (Nanjing University) (ZZKT2022A25, KFKT2022A09, KFKT2023A09, KFKT2023A10). We would also like to thank Huawei Technologies Co., Ltd. for its tremendous support.

REFERENCES

[1] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F Van Dongen, and Wil MP Van Der Aalst. 2015. Measuring precision of modeled behavior. *Information systems and e-Business Management* 13 (2015), 37–67.

[2] Muhammad Ilyas Azeem, Sebastiano Panichella, Andrea Di Sorbo, Alexander Serebrenik, and Qing Wang. 2020. Action-based recommendation in pull-request development. In *Proceedings of the International Conference on Software and System Processes*. 115–124.

[3] Saimir Bala and Jan Mendling. 2018. Monitoring the software development process with process mining. In *Business Modeling and Software Design: 8th International Symposium, BMSD 2018, Vienna, Austria, July 2-4, 2018, Proceedings 8*. Springer, 432–442.

[4] Xu Ben, Shen Beijun, and Yang Weicheng. 2013. Mining developer contribution in open source software using visualization techniques. In *Proceedings of the 3rd International Conference on Intelligent System Design and Engineering Applications*. IEEE, 934–937.

[5] Fábio Bezerra and Jacques Wainer. 2013. Algorithms for anomaly detection of traces in logs of process aware information systems. *Information Systems* 38, 1 (2013), 33–44.

[6] Fábio Bezerra, Jacques Wainer, and Wil MP van der Aalst. 2009. Anomaly detection using process mining. In *Enterprise, Business-Process and Information Systems Modeling: 10th International Workshop, BPMDS 2009, and 14th International Conference, EMMSAD 2009, held at CAiSE 2009, Amsterdam, The Netherlands, June 8-9, 2009. Proceedings*. Springer, 149–161.

[7] Joos CAM Buijs, Boudewijn F van Dongen, and Wil MP van der Aalst. 2014. Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems* 23, 01 (2014), 1440001.

[8] Moataz Chouchen, Ali Ouni, Raula Gaikovina Kula, Dong Wang, Patanamon Thongtanunam, Mohamed Wiem Mkaouer, and Kenichi Matsumoto. 2021. Anti-patterns in modern code review: Symptoms and prevalence. In *Proceedings of the 28th International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 531–535.

[9] Alexandre Decan, Tom Mens, and Hassan Onsori Delicheh. 2023. On the outdatedness of workflows in the GitHub Actions ecosystem. *Journal of Systems and Software* 206 (2023), 111827.

[10] Tapajit Dey and Audris Mockus. 2020. Effect of technical and social factors on pull request quality for the npm ecosystem. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–11.

[11] Emre Doğan and Eray Tüzün. 2022. Towards a taxonomy of code review smells. *Information and Software Technology* 142 (2022), 106737.

[12] Omar Elazhary, Margaret-Anne Storey, Neil Ernst, and Andy Zaidman. 2019. Do as i do, not as i say: Do contribution guidelines match the github contribution process?. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 286–290.

[13] William Enck and Laurie Williams. 2022. Top five challenges in software supply chain security: Observations from 30 industry and government organizations. *IEEE Security & Privacy* 20, 2 (2022), 96–100.

[14] Mehdi Golzadeh, Damien Legay, Alexandre Decan, and Tom Mens. 2020. Bot or not? Detecting bots in GitHub pull request activity based on comment similarity. In *Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops*. 31–35.

[15] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*. 345–355.

[16] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: The integrator's perspective. In *Proceedings of the 37th International Conference on Software Engineering*, Vol. 1. IEEE, 358–368.

[17] Jing Jiang, David Lo, Jiateng Zheng, Xin Xia, Yun Yang, and Li Zhang. 2019. Who should make decision on this pull request? Analyzing time-decaying relationships and file similarities for integrator prediction. *Journal of Systems and Software* 154 (2019), 196–210.

[18] Oleksii Kononenko, Tresa Rose, Olga Baysal, Michael Godfrey, Dennis Theisen, and Bart De Water. 2018. Studying pull request merges: A case study of shopify's active merchant. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*. 124–133.

[19] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. 2022. Taxonomy of attacks on open-source software supply chains. *arXiv preprint arXiv:2204.04008* (2022).

[20] Amanda Lee, Jeffrey C. Carver, and Amiangshu Bosu. 2017. Understanding the impressions, motivations, and barriers of one time code contributors to FLOSS projects: a survey. In *Proceedings of the 39th International Conference on Software Engineering*. ACM, 187–197.

[21] Sander JJ Leemans, Dirk Fahland, and Wil MP van Der Aalst. 2013. Discovering block-structured process models from event logs - A constructive approach. In *Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency*. Springer, 311–329.

[22] Sander JJ Leemans, Dirk Fahland, and Wil MP van Der Aalst. 2014. Discovering block-structured process models from event logs containing infrequent behaviour. In *Proceedings of the 11th International Workshop on Business Process Management*. Springer, 66–78.

[23] Zhixing Li, Yue Yu, Tao Wang, Shanshan Li, and Huaimin Wang. 2022. Opportunities and Challenges in Repeated Revisions to Pull-Requests: An Empirical Study. *Proceedings of the ACM on Human-Computer Interaction* 6, CSCW2 (2022), 1–35.

[24] Zhixing Li, Yue Yu, Tao Wang, Gang Yin, Shanshan Li, and Huaimin Wang. 2021. Are you still working on this? An empirical study on pull request abandonment. *IEEE Transactions on Software Engineering* 48, 6 (2021), 2173–2188.

[25] Marcus Lucero. 2022. The story behind colors.js and faker.js. https://www.revenera.com/blog/software-composition-analysis/the-story-behind-colors-js-and-faker-js/.

[26] Laura MacLeod, Michaela Greiler, Margaret-Anne Storey, Christian Bird, and Jacek Czerwonka. 2017. Code reviewing in the trenches: Challenges and best practices. *IEEE Software* 35, 4 (2017), 34–42.

[27] Ivan Mistrík, John Grundy, Andre Van der Hoek, and Jim Whitehead. 2010. *Collaborative software engineering: challenges and prospects*. Springer, Berlin.

[28] Abdillah Mohamed, Li Zhang, Jing Jiang, and Ahmed Ktob. 2018. Predicting which pull requests will get reopened in github. In *Proceedings of the 25th International Conference on Asia-Pacific Software Engineering Conference*. IEEE, 375–385.

[29] David Myers, Suriadi Suriadi, Kenneth Radke, and Ernest Foo. 2018. Anomaly detection for industrial control systems using process mining. *Computers & Security* 78 (2018), 103–125.

[30] Timo Nolle, Stefan Luettgen, Alexander Seeliger, and Max Mühlhäuser. 2022. BINet: Multi-perspective business process anomaly classification. *Information Systems* 103 (2022), 101458.

[31] Timo Nolle, Alexander Seeliger, and Max Mühlhäuser. 2018. BINet: Multivariate business process anomaly detection using deep learning. In *Proceedings of the 16th International Conference on Business Process Management*. Springer, 271–287.

[32] Saya Onoue, Hideaki Hata, and Ken-ichi Matsumoto. 2013. A study of the characteristics of developers' activities in GitHub. In *Proceedings of the 20th International Conference on Asia-Pacific Software Engineering Conference*. IEEE, 7–12.

[33] Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information needs in contemporary code review. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 1–27.

[34] Sachar Paulus, Nazila Gol Mohammadi, and Thorsten Weyer. 2013. Trustworthy Software Development. In *Communications and Multimedia Security*. Springer Berlin Heidelberg, Berlin, Heidelberg, 233–247.

[35] Wouter Poncin, Alexander Serebrenik, and Mark Van Den Brand. 2011. Process mining software repositories. In *2011 15th European conference on software maintenance and reengineering*. IEEE, 5–14.

[36] Vladimir A. Rubin, Christian W. Günther, Wil M. P. van der Aalst, Ekkart Kindler, Boudewijn F. van Dongen, and Wilhelm Schäfer. 2007. Process Mining Framework for Software Processes. In *Software Process Dynamics and Agility, International Conference on Software Process, ICSP 2007, Minneapolis, MN, USA, May 19-20, 2007, Proceedings*, Vol. 4470. Springer, 169–181.

[37] Vladimir A. Rubin, Irina A. Lomazova, and Wil M. P. van der Aalst. 2014. Agile development with software process mining. In *International Conference on Software and Systems Process 2014, ICSSP '14, Nanjing, China - May 26 - 28, 2014*. ACM, 70–74.

[38] Vladimir A. Rubin, Alexey A. Mitsyuk, Irina A. Lomazova, and Wil M. P. van der Aalst. 2014. Process mining can be applied to software too!. In *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014*. ACM, 57:1–57:8.

[39] Mahdi Sahlabadi, Ravie Chandren Muniyandi, and Zarina Shukur. 2014. Detecting abnormal behavior in social network websites by using a process mining technique. *Journal of Computer Science* 10, 3 (2014), 393.

[40] Nishrith Saini and Ricardo Britto. 2021. Using machine intelligence to prioritise code review requests. In *Proceedings of the 43rd International Conference on Software Engineering: Software Engineering in Practice*. IEEE, 11–20.

[41] Jason Tsay, Laura Dabbish, and James Herbsleb. 2014. Influence of social and technical factors for evaluating contribution in GitHub. In *Proceedings of the 36th international conference on Software engineering*. 356–366.

[42] Pablo Valenzuela-Toledo and Alexandre Bergel. 2022. Evolution of github action workflows. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 123–127.

[43] Wil Van Der Aalst. 2016. *Process mining: data science in action*. Vol. 2. Springer, Berlin.

[44] Wil Van der Aalst, Ton Weijters, and Laura Maruster. 2004. Workflow mining: Discovering process models from event logs. *IEEE transactions on knowledge and data engineering* 16, 9 (2004), 1128–1142.

[45] Erik Van Der Veen, Georgios Gousios, and Andy Zaidman. 2015. Automatically prioritizing pull requests. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 357–361.

[46] Borja Vázquez-Barreiros, Manuel Mucientes, and Manuel Lama. 2015. ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. *Information Sciences* 294 (2015), 315–333.

[47] AJMM Weijters, Wil MP van Der Aalst, and AK Alves De Medeiros. 2006. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP* 166, July 2017 (2006), 1–34.

[48] Peter Weißgerber, Daniel Neu, and Stephan Diehl. 2008. Small patches get in!. In *Proceedings of the 5th International Working Conference on Mining Software Repositories*. ACM, 67–76.

[49] Mairieli Wessel, Joseph Vargovich, Marco A Gerosa, and Christoph Treude. 2023. GitHub Actions: the impact on the pull request process. *Empirical Software Engineering* 28, 6 (2023), 1–35.

[50] Yue Yu, Gang Yin, Huaimin Wang, and Tao Wang. 2014. Exploring the patterns of social behavior in GitHub. In *Proceedings of the 1st international workshop on crowd-based software development methods and technologies*. 31–36.

[51] Nico Zazworka, Victor R Basili, and Forrest Shull. 2009. Tool supported detection and judgment of nonconformance in process execution. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*. IEEE, 312–323.